INTERNATIONAL STANDARD

ISO/IEC 10514-1

First edition
1996-06-01

Information technology — Programming languages —

Part 1:
Modula-2, Base Language

Technologies de l'information — Langages de programmation —
Partie 1: Modula-2, langage de base

Reference number
ISO/IEC 10514-1:1996(E)

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

Contents

Page

Foreword ... xiv

Introduction ... xv

Section 1 : Scope ... 1
- **1.1** Goals ... 1
- **1.2** Specifications included in this part of ISO/IEC 10514 ... 1
- **1.3** Specifications not within the scope of this part of ISO/IEC 10514 ... 1

Section 2 : Normative References ... 3

Section 3 : Definitions, Structure, and Conventions ... 4
- **3.1** Definitions ... 4
- **3.2** Structure of the Formal Definition ... 6
- **3.3** Conventions ... 8
  - **3.3.1** Informative text ... 8
  - **3.3.2** Typographical conventions ... 8
  - **3.3.3** References to pervasive identifiers ... 9
  - **3.3.4** Interpretation of non-VDM-SL notations ... 9
  - **3.3.5** References to formal parameter names ... 9

Section 4 : Requirements for Implementations ... 10
- **4.1** Translation ... 10
- **4.2** Source Code Representation ... 10
- **4.3** Ordering of Declarations ... 11
- **4.4** Predefined Entities ... 11
- **4.5** Library Modules ... 11
- **4.6** Errors ... 12
- **4.7** Exceptions ... 12
- **4.8** Implementation-dependencies ... 13
- **4.9** Documentation ... 13
- **4.10** Statement of Compliance ... 14
- **4.11** Minimum requirements ... 14

© ISO/IEC 1996
All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.
ISO/IEC Copyright Office • Case Postale 56 • CH-1211 Genève 20 • Switzerland
Printed in Switzerland

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Section 5 : The Lexis 16
5.1 Source Code Structure 16
5.2 Word tokens 16
5.2.1 Identifiers 16
5.2.2 Portable Identifiers 16
5.2.3 Full Identifiers 16
5.2.4 Pervasive Identifiers 16
5.2.5 Keywords 17
5.3 Symbols and Operators 17
5.3.1 Required Symbols 17
5.3.2 Symbols with alternatives 18
5.3.3 Operators 18
5.4 Constant Literals 20
5.4.1 Whole Number Literals 20
5.4.2 Real Literals 20
5.4.3 String Literals 20
5.5 Separators 21
5.5.1 White Space 21
5.5.2 Comments 21
5.5.3 Source Code Directives 21
5.6 Letters and National Characters 21

Section 6 : The Language 23
6.1 Programs, Program Modules, and Separate Modules 23
6.1.1 Programs and Compilation Modules 23
6.1.2 Program Modules 27
6.1.3 Definition Modules 28
6.1.4 Implementation Modules 30
6.1.4.1 Sourced Implementation Modules 31
6.1.4.2 Abstract Implementation Modules 33
6.1.5 Program Module and Separate Module Consistency 34
6.1.5.1 Definition Module and Implementation Module Consistency 34
6.1.5.2 Module Compilation Order 36
6.1.5.3 Module Dependencies 37
6.1.6 Module Initialization Order 38
6.1.7 Program Termination 40
6.1.8 Import Lists 41
6.1.8.1 Import List 43
6.1.8.2 Simple Imports 43
6.1.8.3 Unqualified Import 44
6.1.8.4 Import Consistency in Definition Modules 45
6.1.8.5 Import Consistency in Module Blocks 46
6.1.9 Export Lists 46
6.1.9.1 Unqualified Exports 46
6.1.9.2 Qualified Exports 47
6.1.10 Implicit Import and Export 48
6.1.11 Protected Modules 50
6.2 Definitions and Declarations 52
6.2.1 Identifiers 52
6.2.2 Qualified Identifiers 52
6.2.3 Definitions 54
6.2.4 Type Definitions 56
6.2.5 Procedure Headings 57
6.2.5.1 Proper Procedure Headings 58
6.2.5.2 Function Procedure Headings 60
6.2.5.3 Formal Parameters 61
6.2.6 Declarations 63

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.7 Ordering of Procedure and Module Declarations . . . 66
6.2.8 Constant Declarations . . . 67
6.2.9 Type Declarations . . . 68
6.2.10 Variable Declarations . . . 69
6.2.11 Procedure Declarations . . . 71
6.2.11.1 Proper Procedure Declarations . . . 73
6.2.11.2 Function Procedure Declarations . . . 76
6.2.12 Local Module Declarations . . . 79
6.2.13 Auxiliary Formal Model Definitions . . . 82
6.2.13.1 Identifiers Defined in Definitions . . . 82
6.2.13.2 Identifiers Declared in Declarations . . . 82
6.2.13.3 Identifiers Defined in New Types . . . 83
6.2.13.4 Qualified Identifiers Used in Declarations . . . 84
6.2.13.5 Qualified Identifiers used in Expressions . . . 84
6.2.13.6 Qualified Identifiers Used in Types . . . 87
6.2.13.7 Qualified Identifiers Used in Parameters . . . 88
6.2.13.8 Shielded Environments and Shielded Identifiers . . . 89
6.3 Data Types . . . 91
6.3.1 Type Denoters and Ordinal Type Denoters . . . 91
6.3.2 Type Identifiers . . . 92
6.3.3 New Types . . . 93
6.3.4 Enumeration Types . . . 94
6.3.5 Subrange Types . . . 95
6.3.6 Set Types . . . 97
6.3.7 Packedset Types . . . 98
6.3.8 Pointer Types . . . 99
6.3.9 Procedure Types . . . 100
6.3.9.1 Proper Procedure Types . . . 101
6.3.9.2 Function Procedure Types . . . 101
6.3.9.3 Formal Parameter Type Lists . . . 102
6.3.9.4 Variable Formal Types . . . 103
6.3.9.5 Value Formal Types . . . 104
6.3.10 Formal Types . . . 104
6.3.11 Array Types . . . 106
6.3.12 Record Types . . . 107
6.3.12.1 Fixed Fields . . . 109
6.3.12.2 Variant Fields . . . 109
6.3.13 Storage Allocation . . . 114
6.3.13.1 Allocating Storage for Constructed Types . . . 114
6.3.13.2 Allocating Storage for Elementary Types . . . 115
6.3.13.3 Allocating Storage for Structured (Composite) Types . . . 116
6.3.13.4 Allocating Storage for System Storage Types . . . 120
6.4 Expression and Assignment Compatibility . . . 121
6.4.1 Expression Compatibility . . . 121
6.4.2 Assignment Compatibility . . . 122
6.5 Blocks . . . 126
6.5.1 Proper Procedure Blocks . . . 126
6.5.2 Function Procedure Blocks . . . 129
6.5.3 Module Blocks . . . 131
6.5.4 Static Module Initialization . . . 132
6.5.5 Static Module Finalization . . . 135
6.5.6 Dynamic Module Initialization . . . 136
6.5.7 Dynamic Module Finalization . . . 138
6.5.8 Block Bodies and Exception Handling . . . 140
6.5.8.1 Non-result Bodies . . . 142
6.5.8.2 Result Bodies . . . 144
6.6 Statements . . . 146

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.6.1 Statement Sequences . . . 147
6.6.2 Empty Statements . . . 148
6.6.3 Assignment Statements . . . 148
6.6.4 Procedure Calls . . . 152
6.6.5 Return Statements . . . 153
6.6.5.1 Simple Return Statements . . . 154
6.6.5.2 Function Return Statements . . . 154
6.6.6 Retry Statements . . . 155
6.6.7 With Statements . . . 156
6.6.8 If Statements . . . 157
6.6.9 Case Statements . . . 159
6.6.9.1 Case Alternatives . . . 160
6.6.10 While Statements . . . 161
6.6.11 Repeat Statements . . . 162
6.6.12 Loop Statements . . . 163
6.6.13 Exit Statements . . . 164
6.6.14 For Statements . . . 164
6.6.15 Well-formed Control Variables . . . 168
6.6.15.1 Threatening . . . 169
6.6.16 Auxiliary Functions . . . 176
6.6.16.1 Well-formed Return Statements . . . 176
6.6.16.2 Well-formed Retry Statements . . . 177
6.6.16.3 Well-formed Exit Statements . . . 177
6.6.16.4 Statement Classes of Statement Sequences . . . 179
6.7 Variable Designators . . . 182
6.7.1 Entire Designators . . . 182
6.7.2 Indexed Designators . . . 184
6.7.3 Selected Designators . . . 185
6.7.4 Dereferenced Designators . . . 187
6.8 Expressions . . . 189
6.8.1 Ordinal Expressions . . . 190
6.8.2 Infix Expressions and Operations . . . 190
6.8.2.1 Complex Number Infix Operations . . . 193
6.8.2.2 Real Number Infix Operations . . . 196
6.8.2.3 Whole Number Infix Operations . . . 198
6.8.2.4 Boolean Infix Operations . . . 202
6.8.2.5 Set Infix Operations . . . 203
6.8.2.6 String Literal Infix Operations . . . 205
6.8.2.7 Relational Operations . . . 206
6.8.3 Prefix Expressions . . . 219
6.8.3.1 Arithmetic Prefix Operations . . . 220
6.8.3.2 Boolean Prefix Operations . . . 221
6.8.4 Value Designators . . . 222
6.8.4.1 Entire Values . . . 223
6.8.4.2 Indexed Values . . . 225
6.8.4.3 Selected Values . . . 226
6.8.4.4 Dereferenced Values . . . 227
6.8.5 Function Calls . . . 229
6.8.6 Value Constructors . . . 230
6.8.6.1 Array Constructors . . . 231
6.8.6.2 Record Constructors . . . 236
6.8.6.3 Set Constructors . . . 241
6.8.7 Constant Literals . . . 247
6.8.7.1 Whole Number Literals . . . 248
6.8.7.2 Real Literals . . . 249
6.8.7.3 String Literals . . . 250
6.8.8 Constant Expressions . . . 253

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.8.8.1 Constant Expression Evaluation . . . 257
6.9 Parameter Compatibility and Argument Binding . . . 258
6.9.1 Actual Parameters . . . 258
6.9.1.1 Variable Actual Parameters . . . 259
6.9.1.2 Expression Actual Parameters . . . 260
6.9.1.3 Type Parameters . . . 260
6.9.2 Parameter Matching . . . 261
6.9.3 Parameter Compatibility . . . 261
6.9.3.1 Value Parameter Compatibility . . . 262
6.9.3.2 Variable Parameter Compatibility . . . 263
6.9.3.3 System Storage Parameter Compatibility . . . 264
6.9.4 Argument Binding . . . 265
6.9.4.1 Argument Binding to Value Parameters . . . 266
6.9.4.2 Argument Binding to Variable Parameters . . . 267
6.10 Predefined Types, Standard Procedures and Standard Functions . . . 270
6.10.1 The Pervasive Environment . . . 270
6.10.2 Predefined Types . . . 271
6.10.2.1 Predefined Whole Number Types . . . 271
6.10.2.2 Predfined Real Number Types . . . 272
6.10.2.3 Predefined Complex Number Types . . . 273
6.10.2.4 The Boolean Type . . . 274
6.10.2.5 The Character Type . . . 274
6.10.2.6 Values of the Basic Types . . . 276
6.10.2.7 The String Literal Type . . . 278
6.10.2.8 The Nil Type . . . 279
6.10.2.9 The Bitset Type . . . 279
6.10.2.10 The Proc Type . . . 279
6.10.2.11 The Protection Type . . . 280
6.10.3 Standard Procedures . . . 280
6.10.3.1 Standard Procedure Designators . . . 280
6.10.3.2 The Procedure DEC . . . 281
6.10.3.3 The Procedure DISPOSE . . . 283
6.10.3.4 The Procedure EXCL . . . 284
6.10.3.5 The Procedure HALT . . . 285
6.10.3.6 The Procedure INC . . . 286
6.10.3.7 The Procedure INCL . . . 286
6.10.3.8 The Procedure NEW . . . 287
6.10.4 Standard Functions . . . 288
6.10.4.1 Standard Function Designators . . . 288
6.10.4.2 The Function ABS . . . 290
6.10.4.3 The Function CAP . . . 291
6.10.4.4 The Function CHR . . . 292
6.10.4.5 The Function CMPLX . . . 292
6.10.4.6 The Function FLOAT . . . 293
6.10.4.7 The Function HIGH . . . 294
6.10.4.8 The Function IM . . . 295
6.10.4.9 The Function INT . . . 296
6.10.4.10 The Function LENGTH . . . 297
6.10.4.11 The Function LFLOAT . . . 297
6.10.4.12 The Function MAX . . . 298
6.10.4.13 The Function MIN . . . 299
6.10.4.14 The Function ODD . . . 300
6.10.4.15 The Function ORD . . . 301
6.10.4.16 The Function RE . . . 301
6.10.4.17 The Function SIZE . . . 302
6.10.4.18 The Function TRUNC . . . 303
6.10.4.19 The Function VAL . . . 304

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.10.5 Auxiliary Definitions . . . 306
6.11 The Environment and Auxiliary Formal Definitions . . . 307
6.11.1 Environments . . . 307
6.11.1.1 Constant Values . . . 308
6.11.1.2 Types . . . 308
6.11.1.3 Structures . . . 309
6.11.1.4 Expression Types . . . 311
6.11.1.5 Types for Declared Procedures . . . 311
6.11.1.6 Types for Predefined Procedures . . . 311
6.11.1.7 Module Environments . . . 312
6.11.1.8 Variables . . . 312
6.11.1.9 Protection Domains . . . 313
6.11.1.10 Local Continuations . . . 313
6.11.2 Operations on the Environment . . . 313
6.11.2.1 Functions Accessing the Constants Component of the Environment . . . 313
6.11.2.2 Functions Accessing the Types Component of the Environment . . . 314
6.11.2.3 Functions Accessing the Structures Component of the Environment . . . 315
6.11.2.4 Functions Accessing the Variables Component of the Environment . . . 315
6.11.2.5 Functions Accessing the Procedures Component of the Environment . . . 316
6.11.2.6 Functions Accessing the Modules Component of the Environment . . . 317
6.11.2.7 Functions Accessing the Procedure Level Component of the Environment . . . 317
6.11.2.8 Functions Accessing the Continuations Component of the Environment . . . 318
6.11.2.9 Functions Accessing the Protection Components of the Environment . . . 318
6.11.2.10 Functions Accessing the Environment . . . 319
6.11.2.11 Operations to Update Environments . . . 319
6.11.3 Types . . . 320
6.11.3.1 New Type Names . . . 320
6.11.3.2 Structured (Composite) Types . . . 321
6.11.3.3 Elementary Types . . . 321
6.11.3.4 Scalar Types . . . 322
6.11.3.5 Ordinal Types . . . 322
6.11.3.6 Whole Number Types . . . 323
6.11.3.7 Number Types . . . 324
6.11.3.8 String Types . . . 324
6.11.3.9 System Storage Types . . . 324
6.11.4 Unit Types . . . 325
6.11.4.1 Range Type . . . 325
6.11.4.2 Host Type . . . 325
6.11.4.3 Base Type . . . 325
6.11.4.4 Component Type . . . 325
6.11.4.5 Index Type . . . 326
6.11.4.6 Bound Type . . . 326
6.11.5 Other Type Functions . . . 326
6.11.5.1 Return Types . . . 326
6.11.5.2 Formal Parameter Types . . . 326
6.11.5.3 Auxiliary Functions for Records . . . 326
6.11.5.4 Auxiliary Functions for Arrays . . . 328
6.11.5.5 Auxiliary Functions for Open Arrays . . . 328
6.11.5.6 Auxiliary Functions for Opaque Types . . . 328
6.11.6 Values Associated with a Type . . . 328
6.12 The Storage Model and Auxiliary Formal Definitions . . . 331
6.12.1 The Program State . . . 331
6.12.2 The External Program State . . . 331
6.12.2.1 Operations to Initialize and Terminate the Program State . . . 331
6.12.3 The Store . . . 332
6.12.3.1 Storage . . . 332
6.12.3.2 Values . . . 333

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.12.4 Procedure Denotations . . . 334
6.12.5 Termination . . . 334
6.12.6 Exceptions . . . 334
6.12.7 Interrupt Handlers . . . 335
6.12.8 Operations on Storage . . . 335
6.12.8.1 Operations Accessing Storage . . . 335
6.12.8.2 Storage Aliasing . . . 336
6.12.8.3 Operations to Allocate and Deallocate Storage . . . 337
6.12.8.4 Operations to Allocate and Deallocate Storage Locations . . . 338
6.12.8.5 Access Coroutine Storage . . . 339
6.12.9 Accessing the Procedure Denotation Component of the State . . . 339
6.12.10 Program Initialization and Program Termination . . . 340
6.12.10.1 Program Initialization . . . 340
6.12.10.2 Program Termination . . . 340
6.12.10.3 Program Halting . . . 341
6.12.11 Exception Handling . . . 341
6.12.11.1 Allocation of Exception Sources . . . 342
6.12.12 Interrupt Handlers . . . 343
6.12.13 Accessing the Protection Component of the State . . . 344
6.12.14 Coroutine and Continuations . . . 344
6.12.14.1 The Coroutine Environment . . . 344
6.12.14.2 The Continuations . . . 344
6.12.14.3 Operations on Continuations . . . 345

Section 7 : System Modules . . . 347
7.1 The Module SYSTEM . . . 348
7.1.1 The Interface to SYSTEM . . . 348
7.1.2 The Constants of SYSTEM . . . 351
7.1.3 The Types of SYSTEM . . . 352
7.1.3.1 System Storage Types . . . 352
7.1.3.2 The Address Type . . . 353
7.1.4 The Functions of SYSTEM . . . 353
7.1.5 Address Arithmetic Functions . . . 354
7.1.5.1 The Function ADDADR . . . 355
7.1.5.2 The Function SUBADR . . . 356
7.1.5.3 The Function DIFADR . . . 357
7.1.5.4 Properties of the address arithmetic functions . . . 358
7.1.6 Address Construction and Enquiry Functions . . . 359
7.1.6.1 The Function MAKEADR . . . 359
7.1.6.2 The Function ADR . . . 359
7.1.7 Packedset Functions . . . 360
7.1.7.1 Packedset mapping . . . 361
7.1.7.2 The Function ROTATE . . . 361
7.1.7.3 The Function SHIFT . . . 362
7.1.8 The Function CAST . . . 364
7.1.9 The Function TSIZE . . . 365
7.2 The Module COROUTINES . . . 368
7.2.1 The Interface to COROUTINES . . . 368
7.2.2 The Types of COROUTINES . . . 369
7.2.3 The Procedures of COROUTINES . . . 370
7.2.3.1 The Procedure ATTACH . . . 371
7.2.3.2 The Procedure DETACH . . . 371
7.2.3.3 The Procedure IOTRANSFER . . . 372
7.2.3.4 The Procedure LISTEN . . . 373
7.2.3.5 The Procedure NEWCOROUTINE . . . 373
7.2.3.6 The Procedure TRANSFER . . . 376
7.2.4 The Functions of COROUTINES . . . 376

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.4.1 The Function HANDLER . . . 377
7.2.4.2 The Function IsATTACHED . . . 378
7.2.4.3 The Function CURRENT . . . 379
7.2.4.4 The Function PROT . . . 379
7.3 The Module EXCEPTIONS . . . 380
7.3.1 The Interface to EXCEPTIONS . . . 380
7.3.2 The Types of EXCEPTIONS . . . 381
7.3.3 The Procedures of EXCEPTIONS . . . 381
7.3.3.1 The Procedure AllocateSource . . . 382
7.3.3.2 The Procedure RAISE . . . 383
7.3.3.3 The Procedure GetMessage . . . 383
7.3.4 The Functions of EXCEPTIONS . . . 384
7.3.4.1 The Function CurrentNumber . . . 385
7.3.4.2 The Function IsCurrentSource . . . 386
7.3.4.3 The Function IsExceptionalExecution . . . 387
7.3.4.4 The Use of EXCEPTIONS . . . 387
7.4 The Module TERMINATION . . . 389
7.4.1 The Interface to TERMINATION . . . 389
7.4.2 The Functions of TERMINATION . . . 389
7.4.2.1 The Function IsTerminating . . . 390
7.4.2.2 The Function HasHalted . . . 391
7.5 The Module M2EXCEPTION . . . 392
7.5.1 The Interface to M2EXCEPTION . . . 392
7.5.2 The Types of M2EXCEPTION . . . 393
7.5.3 The Functions of M2EXCEPTION . . . 393
7.5.3.1 The Function M2Exception . . . 394
7.5.3.2 The Function IsM2Exception . . . 395
7.5.4 Language Exceptions . . . 395
7.5.4.1 Exceptions whose detection is required . . . 396
7.5.4.2 Exceptions whose detection is not required . . . 397
7.5.5 Messages Associated with Language Exceptions . . . 397
7.5.6 Aggregation and Raising of Language Exceptions . . . 398

Section 8 : Required Library Modules . . . 399
8.1 The Module Storage . . . 399
8.1.1 The Definition Module of Storage . . . 399
8.1.2 The Procedures of Storage . . . 400
8.1.3 Exceptions Raised by Storage . . . 402
8.1.4 The State for Storage . . . 402
8.1.5 The Initialization of Storage . . . 403
8.1.6 Auxiliary Operations . . . 403
8.2 The Modules LowReal and LowLong . . . 405
8.2.1 The Definition Module of LowReal . . . 405
8.2.2 The Definition Module of LowLong . . . 406
8.2.3 The Constants of LowReal and LowLong . . . 407
8.2.4 The Procedures of LowReal and LowLong . . . 409
8.2.5 Exceptions Raised by LowReal and LowLong . . . 411
8.3 The Module CharClass . . . 412
8.3.1 The Definition Module of CharClass . . . 412
8.3.2 The Procedures of CharClass . . . 412

Section 9 : Standard Library Modules . . . 415
9.1 The Mathematical Libraries . . . 415
9.1.1 The Definition Module of RealMath . . . 415
9.1.2 The Definition Module of LongMath . . . 416
9.1.3 The Constants of RealMath and LongMath . . . 417
9.1.4 The Procedures of RealMath and LongMath . . . 417

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9.1.5 Exceptions Raised by RealMath and LongMath . . . 420
9.1.6 The State for RealMath and LongMath . . . 421
9.1.7 The Initialization of RealMath and LongMath . . . 421
9.1.8 The Definition Module of ComplexMath . . . 421
9.1.9 The Definition Module of LongComplexMath . . . 422
9.1.10 The Constants of ComplexMath and LongComplexMath . . . 423
9.1.11 The Procedures of ComplexMath and LongComplexMath . . . 423
9.1.12 Exceptions Raised by ComplexMath and LongComplexMath . . . 428
9.1.13 The State for ComplexMath and LongComplexMath . . . 429
9.1.14 The Initialization of ComplexMath and LongComplexMath . . . 429
9.1.15 Auxiliary Functions . . . 429
9.2 The Input/Output Library . . . 430
9.2.1 Standard and Default Channels . . . 431
9.2.1.1 The Module StdChans . . . 431
9.2.2 Reading and Writing of Data using Specified Channels . . . 434
9.2.2.1 The Module TextIO . . . 435
9.2.2.2 The Module WholeIO . . . 439
9.2.2.3 The Modules RealIO and LongIO . . . 442
9.2.2.4 The Module RawIO . . . 447
9.2.2.5 The Module IOConsts . . . 448
9.2.2.6 The Module IOResult . . . 449
9.2.2.7 Auxiliary Formal Model Definitions . . . 450
9.2.3 Reading and Writing of Data using Default Channels . . . 455
9.2.3.1 The Module STextIO . . . 455
9.2.3.2 The Module SWholeIO . . . 456
9.2.3.3 The Modules SRealIO and SLongIO . . . 457
9.2.3.4 The Module SRawIO . . . 459
9.2.3.5 The Module SIOResult . . . 459
9.2.4 The Interface for Device-Independent Channel Operations . . . 461
9.2.4.1 The Module IOChan . . . 461
9.2.5 Obtaining Channels from Device Modules . . . 472
9.2.5.1 The Module ChanConsts . . . 472
9.2.5.2 The Module StreamFile . . . 475
9.2.5.3 The Module SeqFile . . . 483
9.2.5.4 The Module RndFile . . . 497
9.2.5.5 The Module TermFile . . . 511
9.2.5.6 The Module ProgramArgs . . . 522
9.2.5.7 Implementation-defined Functions used in the Device Modules . . . 529
9.2.5.8 Auxiliary Formal Model Definitions . . . 529
9.2.6 The Interface to Channels for New Device Modules . . . 531
9.2.6.1 The Module IOLink . . . 531
9.2.7 The State for the Input/Output Library . . . 536
9.3 Concurrent Programming . . . 544
9.3.1 The Module Processes . . . 544
9.3.1.1 The Definition Module of Processes . . . 546
9.3.1.2 The Procedures of Processes . . . 548
9.3.1.3 Exceptions Raised by Processes . . . 554
9.3.1.4 The State for Processes . . . 555
9.3.1.5 The Initialization of Processes . . . 555
9.3.1.6 Auxiliary Formal Model Definitions . . . 556
9.3.2 The Module Semaphores . . . 559
9.3.2.1 The Definition Module of Semaphores . . . 559
9.3.2.2 The Procedures of Semaphores . . . 559
9.3.2.3 Exceptions Raised by Semaphores . . . 562
9.3.2.4 The State for Semaphores . . . 562
9.3.2.5 The Initialization of Semaphores . . . 563
9.3.2.6 Auxiliary Formal Model Definitions . . . 563

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9.4 The Module Strings . . . 564
9.4.1 The Definition Module of Strings . . . 564
9.4.2 The Procedures of Strings . . . 567
9.4.3 Auxiliary Formal Model Definitions . . . 579
9.5 The String Conversions Library . . . 581
9.5.1 Common Data Types . . . 581
9.5.1.1 The Module ConvTypes . . . 581
9.5.2 High-Level String Conversion Modules . . . 582
9.5.2.1 The Module WholeStr . . . 582
9.5.2.2 The Modules RealStr and LongStr . . . 586
9.5.3 Low-Level String Conversion Modules . . . 592
9.5.3.1 The Module WholeConv . . . 592
9.5.3.2 The Modules RealConv and LongConv . . . 598
9.5.4 Auxiliary Definitions . . . 605
9.6 The Module SysClock . . . 607
9.6.1 The Definition Module of SysClock . . . 607
9.6.2 The Types of SysClock . . . 608
9.6.3 The Procedures of SysClock . . . 608
9.6.4 The State for SysClock . . . 609

Section 10: Auxiliary Formal Model Definitions . . . 611
10.1 Extensions to VDM-SL . . . 611
10.1.1 Replacement Definitions for Combinators . . . 611
10.1.2 The McCarthy Conditional Expression . . . 612
10.2 VDM-SL Functions . . . 614
10.2.1 Set functions . . . 614
10.2.2 Sequence functions . . . 614
10.2.3 Map functions . . . 615
10.2.4 Numerical functions . . . 615

Annexes

Annex A Minimum Limit Specifications . . . 616
A.1 The Limit-specification Generators . . . 616
A.1.1 Length of Identifiers . . . 616
A.1.2 Length of String Literals . . . 617
A.1.3 Size of Single Dimension Arrays . . . 617
A.1.4 Number of Dimensions of Multidimension Arrays . . . 618
A.1.5 Depth of Nested Records . . . 619
A.1.6 Number of Sequential Local Modules . . . 620
A.1.7 Depth of Nested Local Modules . . . 620
A.1.8 Number of Enumeration Literals . . . 621
A.1.9 Number of Sequential Procedure Declarations . . . 622
A.1.10 Depth of Nested Procedure Declarations . . . 623
A.1.11 Number of Coroutines . . . 624
A.1.12 Number of Parameters of a Procedure . . . 625
A.1.13 Depth of Nested IF-THEN-ELSEs in THEN-branch . . . 626
A.1.14 Depth of Nested IF-THEN-ELSEs in Both Branches . . . 627
A.1.15 Depth of Nested LOOP Statements . . . 628
A.1.16 Depth of Nested WHILE Statements . . . 629
A.1.17 Depth of Nested REPEAT Statements . . . 630
A.1.18 Depth of Nested FOR Statements . . . 631
A.1.19 Depth of Nested WITH Statements . . . 632
A.1.20 Number of CASE Alternatives with ELSE part . . . 633
A.1.21 Depth of Nested Parentheses in Expressions . . . 634
A.1.22 Depth of Recursion . . . 635
A.1.23 Number of Imported Modules . . . 635

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Annex B The Specification of Library Modules . . . 637
B.1 Components of a Library Module Specification . . . 637
B.2 Interpretation of a Library Module Specification . . . 637

Annex C Collected Modula-2 Concrete Syntax . . . 639
C.1 Programs, Program Modules, and Separate Modules . . . 639
C.1.1 Programs and Compilation Modules . . . 639
C.1.2 Program Modules . . . 639
C.1.3 Definition Modules . . . 639
C.1.4 Implementation Modules . . . 639
C.1.4.1 Sourced Implementation Modules . . . 639
C.1.5 Import Lists . . . 639
C.1.5.1 Import List . . . 639
C.1.5.2 Simple Imports . . . 639
C.1.5.3 Unqualified Import . . . 639
C.1.6 Export Lists . . . 640
C.1.6.1 Unqualified Exports . . . 640
C.1.6.2 Qualified Exports . . . 640
C.1.7 Protected Modules . . . 640
C.2 Definitions and Declarations . . . 640
C.2.1 Qualified Identifiers . . . 640
C.2.2 Definitions . . . 640
C.2.3 Type Definitions . . . 640
C.2.4 Procedure Headings . . . 640
C.2.4.1 Proper Procedure Headings . . . 640
C.2.4.2 Function Procedure Headings . . . 641
C.2.4.3 Formal Parameters . . . 641
C.2.5 Declarations . . . 641
C.2.6 Constant Declarations . . . 641
C.2.7 Type Declarations . . . 641
C.2.8 Variable Declarations . . . 641
C.2.9 Procedure Declarations . . . 641
C.2.9.1 Proper Procedure Declarations . . . 642
C.2.9.2 Function Procedure Declarations . . . 642
C.2.10 Local Module Declarations . . . 642
C.3 Data Types . . . 642
C.3.1 Type Denoters and Ordinal Type Denoters . . . 642
C.3.2 Type Identifiers . . . 642
C.3.3 New Types . . . 642
C.3.4 Enumeration Types . . . 642
C.3.5 Subrange Types . . . 642
C.3.6 Set Types . . . 643
C.3.7 Packedset Types . . . 643
C.3.8 Pointer Types . . . 643
C.3.9 Procedure Types . . . 643
C.3.9.1 Proper Procedure Types . . . 643
C.3.9.2 Function Procedure Types . . . 643
C.3.9.3 Formal Parameter Type Lists . . . 643
C.3.9.4 Variable Formal Types . . . 643
C.3.9.5 Value Formal Types . . . 643
C.3.10 Formal Types . . . 643
C.3.11 Array Types . . . 644
C.3.12 Record Types . . . 644
C.3.12.1 Fixed Fields . . . 644
C.3.12.2 Variant Fields . . . 644
C.4 Blocks . . . 644
C.4.1 Proper Procedure Blocks . . . 644

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

C.4.2 Function Procedure Blocks . . . 644
C.4.3 Module Blocks . . . 645
C.4.4 Block Bodies and Exception Handling . . . 645
C.5 Statements . . . 645
C.5.1 Statement Sequences . . . 645
C.5.2 Empty Statements . . . 645
C.5.3 Assignment Statements . . . 645
C.5.4 Procedure Calls . . . 645
C.5.5 Return Statements . . . 645
C.5.5.1 Simple Return Statements . . . 645
C.5.5.2 Function Return Statements . . . 646
C.5.6 Retry Statements . . . 646
C.5.7 With Statements . . . 646
C.5.8 If Statements . . . 646
C.5.9 Case Statements . . . 646
C.5.9.1 Case Alternatives . . . 646
C.5.10 While Statements . . . 646
C.5.11 Repeat Statements . . . 646
C.5.12 Loop Statements . . . 647
C.5.13 Exit Statements . . . 647
C.5.14 For Statements . . . 647
C.6 Variable Designators . . . 647
C.6.1 Entire Designators . . . 647
C.6.2 Indexed Designators . . . 647
C.6.3 Selected Designators . . . 647
C.6.4 Dereferenced Designators . . . 647
C.7 Expressions . . . 648
C.7.1 Ordinal Expressions . . . 648
C.7.2 Infix Expressions and Operations . . . 648
C.7.3 Value Designators . . . 648
C.7.3.1 Entire Values . . . 648
C.7.3.2 Indexed Values . . . 648
C.7.3.3 Selected Values . . . 648
C.7.3.4 Dereferenced Values . . . 649
C.7.4 Function Calls . . . 649
C.7.5 Value Constructors . . . 649
C.7.5.1 Array Constructors . . . 649
C.7.5.2 Record Constructors . . . 649
C.7.5.3 Set Constructors . . . 649
C.7.6 Constant Literals . . . 650
C.7.7 Constant Expressions . . . 650
C.8 Parameter Compatibility and Argument Binding . . . 650
C.8.1 Actual Parameters . . . 650
C.8.1.1 Type Parameters . . . 650
C.9 Lexical components . . . 650

Annex D Collected Modula-2 Abstract Syntax . . . 652

Annex E Modula-2 Glossary . . . 660

Annex F Participating Individuals and Organisations . . . 674

Index . . . 676

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

International Standard ISO/IEC 10514-1 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces.*

ISO/IEC 10514 consists of the following parts, under the general title *Information technology — Programming languages*:

— *Part 1: Modula-2, Base Language*

— *Part 2: Object-oriented Modula-2*

Annexes A and B form an integral part of this part of ISO/IEC 10514. Annexes C to F are for information only.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Introduction

This part of ISO/IEC 10514, Modula-2 (Base language), provides a specification of the form and meaning of Modula-2 programs, and by reference to that specification lays down requirements for Modula-2 implementations.

Modula-2 is a general-purpose programming language, characterized by modules that provide data and procedure encapsulation and hiding. It provides facilities for concurrent programming based on coroutines, and includes facilities for low-level programming. Modula-2 is, therefore, suitable for systems programming. Facilities such as input/output, mathematical operations and string processing are made available through library modules.

Modula-2 is a descendent of Modula whose ancestor is Pascal. All three languages were devised by Professor Niklaus Wirth, ETH, Zürich, Switzerland. The first description of Modula-2 was published in 1980. The language became popular during the early 1980s and many implementations were developed. Because of different interpretations of various descriptions of Modula-2, significant differences in implementations appeared, and in 1984 a group of Modula-2 proponents formed a BSI panel to develop a standard. The BSI panel successfully proposed a new work item to ISO, and the ISO working group met for the first time in 1987. In 1989 the first draft of this standard was published, and in 1992 the second committee draft was published. The base documents for the formulation of this part of ISO/IEC 10514 were the various editions of *Programming in Modula-2* by Niklaus Wirth, published by Springer-Verlag.

This part of ISO/IEC 10514 contains a formal model of Modula-2; for the sake of generality, consistency, and precision, differences have been introduced between the model and the third edition of Professor Wirth's book *Programming in Modula-2* and these are identified either as 'clarifications' or 'changes'. During the standardization of Modula-2 it was recognized that users of the language were demanding extensions to the language devised by Professor Wirth and that implementors were responding by providing them. Several changes and extensions have, therefore, been made in order to prevent further divergence, and to ensure consistency and coherence.

This part of ISO/IEC 10514 uses a variety of notations to specify the requirements contained in it. A standard syntactic metalanguage is used for specifiying lexical and syntactic structures. The Vienna Development Method-Specification Language (VDM-SL), which is based upon conventional mathematics, is used to define the semantics of the language in a mathematically rigorous manner. In addition, the semantics of the Modula-2 language, system modules and standard library are also specified using natural language, albeit in a form that is possibly ambiguous. It is intended that the corresponding VDM-SL resolves any such ambiguity. If the natural language requirements and those expressed in the VDM-SL conflict, then there is a mistake in this standard. In many cases, small details of the language semantics may only be resolved by reference to the VDM-SL; by this means the natural language description has been kept relatively simple. A number of typographic and stylistic conventions have been adopted in order to make the natural language description succinct.

The definitions of key terms used in this part of ISO/IEC 10514 are expressed in natural language, with references to VDM-SL where appropriate. The requirements for conforming implementations are given in natural language, with references to VDM-SL where appropriate.

The development of large parts of the formal specification of the semantics of Modula-2 in this part of ISO/IEC 10514 was sponsored by the U.K. Alvey Research Programme. The Alvey sponsored work was carried out at Leicester University Computer Studies Department in collaboration with BSI QA Software Engineering Department.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Information technology — Programming languages — Part 1: Modula-2, Base Language

Section 1 : Scope

1.1 Goals

The goals of this part of ISO/IEC 10514 are:

— to provide a rigorous definition of the language Modula-2 and its standard library by providing a mathematical model of both;

— to provide a resolution of differences among interpretations of other descriptions of Modula-2 and its standard library, while endeavouring to preserve investment in existing practice;

— to remove features thought to be redundant, inherently flawed, or inadequate;

— to specify new language and standard library facilities where a need is perceived to exist;

— to maintain the general principles of Modula-2 laid down by its inventor, while allowing for later modernization and standardization.

1.2 Specifications included in this part of ISO/IEC 10514

This part of ISO/IEC 10514 provides specifications for:

— required symbols for Modula-2 program representation, including comments, literals, and source code directives;

— the lexical structure, the syntactic structure and the semantics of Modula-2 programs, including programs that make use of system modules.

— the interface to and the semantics of standard Modula-2 library modules.

— those separate modules of the standard library that a conforming implementation is required to supply;

— violations of the rules for use of the language, system modules and standard library modules that a conforming implementation is required to detect;

— certain criteria for the size and complexity of programs that a conforming implementation must accept;

— further compliance requirements for implementations, including documentation requirements.

1.3 Specifications not within the scope of this part of ISO/IEC 10514

This part of ISO/IEC 10514 provides no specifications for:

— the underlying representation of predefined data types (except in the case of packedset types; see 7.1.7.1);

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

— the method by which implementations are invoked (including identification of the program module and associated definition and implementation modules);

— the method by which compilation modules are stored (including the correspondence between module names and system file names where files are used);

— the method by which implementations accept input (including the encoding of source text and including the number of compilation modules accepted for each invocation);

— performance aspects of implementations, and certain quality aspects not covered by 1.2;

— the effect of executing a program that uses extensions to the language, extensions to system modules or extensions to standard library modules, or that otherwise deviates from this part of ISO/IEC 10514;

— the effect of continuing execution of a program in which an exception has occurred and execution has continued without an exception being raised;

— the meaning of a program that relies on a definition of implementation-dependent values or implementation-dependent behaviour.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Section 2 : Normative References

The following standards contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 10514. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of ISO/IEC 10514 are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 10967-1:1994, *Information technology—Language independent arithmetic—Part 1: Integer and floating point arithmetic.*

ISO/IEC 13817-1:...[1)], *Information technology — Programming languages, their environments and system software interfaces — Vienna Development Method — Specification Language—Part 1: Base language.*

BS 6154:1981, *Method of defining Syntactic Metalanguage.*

IEC 559:1989, *Binary floating-point arithmetic for microprocessor systems*

[1)] To be published.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Section 3 : Definitions, Structure, and Conventions

3.1 Definitions

For the purposes of this part of ISO/IEC 10514, the following definitions apply.

NOTE 1 — A glossary of terms used is also provided — see annex E.

3.1.1 source code: A fragment of text that is potentially part of the text representation of a component of a Modula-2 program.

3.1.2 separate module: A component of a Modula-2 program, that may be separately developed, translated, and provided, and which comprises an encapsulation of data and/or procedures that are available for use by clients of the module, together with any actions that are executed to initialize the module, and any actions that are executed to finalize the module.

NOTE 2 — The interface to a separate module is provided by a Modula-2 definition module so that other separate modules written in Modula-2 can use its facilities. The executable part of a separate module is specified in Modula-2 by an implementation module; however, an implementation of a separate module need not be derived from Modula-2 source code.

3.1.3 library: A collection of separate modules provided for general use.

3.1.4 library module: A separate module that is part of a library.

3.1.5 standard library: The library consisting of the separate modules specified in this part of ISO/IEC 10514: the required modules in section 8 and the non-required modules in section 9.

3.1.6 predefined: A term which is applied to an entity (module, procedure, type or constant), or an identifier, that is built-in to Modula-2.

3.1.7 pervasive identifier: An identifier of a predefined procedure, type, or constant that is itself predefined and whose scope covers the entirety of compilation modules, unless overridden by an identifier of the same spelling that is introduced in a definition or a declaration.

3.1.8 system module: A predefined module that is an encapsulation of predefined entities whose identifiers are only available by import from that system module. Unlike separate modules, system modules cannot be replaced.

NOTE 3 — System modules do not form part of the standard library.

3.1.9 computer system: The combination of hardware and, optionally, firmware and software (e.g. operating system) that enables the execution of software.

3.1.10 Modula-2 implementation: The means by which an executable program for a computer system is produced from the components of a Modula-2 program.

An implementation comprises at least the following:

— a translator;

— any necessary run-time support for the predefined entities such as system modules and standard procedures;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

— required separate modules;

— any software needed to support or integrate the above.

NOTES

4 The term 'translator' is intended to encompass technologies both known (e.g. compilers and interpreters) and those yet to be developed.

5 A translator need not work from source code input since an implemention can choose to use non-text representations that are deriveable from source code. — see 4.1.

6 A translator can deal with access to predefined entities in a special way (e.g. 'calls' to system module 'procedures' may be translated as inline code).

3.1.11 standard library module implementation: The means by which a module of the standard library is provided independently of a Modula-2 implementation.

3.1.12 implementor: The producer of a Modula-2 implementation.

3.1.13 processor: The combination of computer system and implementation that permits the translation and execution of Modula-2 programs. A 'processor' may include two computer systems if cross-translation is involved; in this situation an implementation's translator is operated on a **host** computer system and generates code for execution on a **target** computer system.

NOTE 7 — The term 'processor' does not preclude implementations that involve more than one hardware processor.

3.1.14 abstract implementation module: The specification of the semantics of a separate module using the formal notation VDM-SL — see annex B.

3.1.15 sourced implementation module: The implementation of a separate module for which source code is provided and whose semantics is therefore determined by the semantics of that source code as specified by this part of ISO/IEC 10514.

3.1.16 standard mode: A mode of use of an implementation in which it complies with the requirements of this part of ISO/IEC 10514.

3.1.17 implementation-defined: A term applied to a value that is not defined by this part of ISO/IEC 10514, but that requires definition before the meaning of a program that relies upon that value can be determined.

NOTE 8 — For example, the minimum and maximum values of the `INTEGER` type are implementation-defined, and the value of a call of `SYSTEM.CAST` is implementation-defined.

This part of ISO/IEC 10514 requires conforming implementations to be accompanied by documentation which includes definitions of implementation-defined values for each processor — see 4.9.

3.1.18 implementation-dependent: A term applied to a value or to behaviour that is not defined by this part of ISO/IEC 10514, and that requires no definition since the meaning of a program that relies upon such a definition is not specified by the terms of this part of ISO/IEC 10514.

NOTE 9 — For example, the order of evaluation of operands in an expression is implementation-dependent; consequently, a program that relies on the order of evaluation of operands in an expression is not given meaning by this part of ISO/IEC 10514.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

3.1.19 error: Source code is said to be in error or to contain an error if either the source code is not well-formed (i.e. it does not conform to lexical or concrete syntax rules) or if syntactically correct source code is not well-formed according to static semantic rules. For example, the syntactically correct source code fragment `a1 + a2` is in error if the entity designated by `a1` (or `a2`) is an array.

3.1.20 exception: A run-time event that is in violation of the dynamic semantic rules of this part of ISO/IEC 10514.

NOTE 10 — For example, an exception occurs if there is an attempt to dereference a pointer variable whose value is `NIL`. An exception should be detected and raised by an implementation, in which case there will be a defined control flow of the program that is different from the normal flow of control.

3.1.21 exception raising: The invocation of exception handling, or exceptional termination, following the detection of an exception occurrence.

NOTES

11 This part of ISO/IEC 10514 requires that the occurrence of certain exceptions be detected and raised by an implementation in standard mode. The function *mandatory-detection* is used in the formal definition in these cases and *non-mandatory-detection* is used otherwise — see 4.7.

12 If an implementation does not detect and raise an exception that has occurred, the effect is implementation-dependent; consequently, this part of ISO/IEC 10514 does not give meaning to a program that relies on the detection of an exception if that exception is one for which detection is not required — see 1.3.

3.1.22 declare-before-use-in-declarations: Implementations which only require that entities used in declarations be declared before they are used are described as declare-before-use-in-declarations. (They are often called 'multi-pass' implementations.) Declare-before-use-in-declarations implementations can, for example, process programs that contain local modules which import from each other. Implementations are termed declare-before-use-in-declarations if they satisfy the first predicate in 6.2.7.

3.1.23 declare-before-use: Implementations which insist that all entities except pointer types and procedure types be declared before they are used are described as declare-before-use. (They are often imprecisely called 'single-pass' implementations.) Declare-before-use implementations cannot, for example, process programs that contain local modules which import from each other. Implementations are termed declare-before-use if they satisfy the second predicate in 6.2.7.

3.2 Structure of the Formal Definition

This part of ISO/IEC 10514 uses both formal notations and natural language to state its requirements. For the most part the form and meaning of Modula-2 programs and the system modules, and library modules they use, is expressed using a formal definition accompanied by natural language normative statements and notes. The semantics of Modula-2 programs and standard libraries is expressed formally using VDM-SL (ISO/IEC 13817) with some local extensions which are described in 10.1.

Specification of language constructs

The formal definition of Modula-2 language constructs is divided into the lexis and five main parts for which headings appear under each language construct: concrete syntax, abstract syntax, declaration semantics, static semantics, and dynamic semantics. Other parts of the definition appear under the headings properties and domains.

3.2.1 lexis: The set of rules defining the lexical elements of Modula-2 source code in terms of characters or character sequences.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The Lexis is formally expressed using the *Standard Syntactic Metalanguage* (BS 6154, 1981) and is given in section 5.

3.2.2 concrete syntax: The set of grammatical rules, or 'productions', defining the syntactic structure of the language in terms of lexical tokens.

The concrete syntax is formally expressed using the *Standard Syntactic Metalanguage* (BS 6154, 1981) and is given separately for each construct in the language in section 6.

Terms from the concrete syntax are used in the natural language normative statements. The tokens of the concrete syntax have apparently meaningful names, and some productions are present solely to allow such terms to be used (for example, the token constant expression is simply defined as an expression). The actual meaning of these terms is expressed by the static semantics.

NOTE 1 — The concrete syntax rules are collected in informative annex C.

3.2.3 abstract syntax: An abstract version of the underlying structure of the language in terms of a hierarchy of VDM-SL types. Translation between the representations required by the concrete and abstract syntax is implicit. In a few cases, where the abstract syntax is not directly related to the concrete syntax, annotations are added to explain any apparent divergence.

The abstract syntax is used only in the VDM-SL specification.

NOTE 2 — The abstract syntax rules are collected in informative annex D.

3.2.4 declaration semantics: The set of rules giving the meaning of definitions, declarations, and of imports and exports, for the purpose of expressing the static semantics of the language. The rules define the scope of identifiers and their meaning within that scope — see 6.2. In this part of ISO/IEC 10514, the visibility and meaning of an identifier at the point of its use is determined by the environment value that applies at that point — see 6.11.1. The VDM-SL functions expressing the declaration semantics construct environment values from given environment values.

3.2.5 static semantics: The VDM-SL static semantics is the set of rules, expressed in the form of truth-valued functions (i.e. the 'well-formedness' predicates) on components of the abstract syntax, that specify the legality of an abstractly represented Modula-2 program.

The correponding natural language normative statements express constraints that are not formally imposed by the concrete syntax (for example, that where a component called a constant expression is expected, that component is an expression that can be evaluated without executing the program).

3.2.6 dynamic semantics: The set of rules giving the meaning of the elaboration of the declarations, and the execution of the statements, of a program that satisfies the static semantics. In VDM-SL, the dynamic semantics are expressed by operations which implicitly operate on an internal state — see 6.12.1.

3.2.7 properties: A set of equations or inequalities that specify requirements by giving relations that must apply between values of the VDM-SL types that are used in the formal specification.

3.2.8 domains: The VDM-SL types whose values are used to build the mathematical model which expresses the semantics of a Modula-2 program. The two main such types are used to model the environment and the state; the other VDM-SL types given under the heading of domains are component types of one or both of the environment or the state.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Specification of predefined entities

Identifiers of predefined types, constants, procedures, and modules may appear in the concrete and abstract representations of a program but these predefined entities do not have definitions or declarations that appear within a program. The declaration semantics for predefined entities gives the meaning of these identifiers for the purpose of expressing the static semantics governing their use.

The static and dynamic semantics that apply to the use of predefined types, constants, and modules, are given by the static and dynamic semantics for the language constructs that allow the use of such types, constants, and modules. Static and dynamic semantics for calls of predefined procedures are given explicitly for each such procedure.

In the case of the specification of system modules, a pseudo-definition module is given for informational purposes only.

Specification of separate modules

In the case of the specification of separate modules, definition modules are given to which the declaration semantics, static semantics, and dynamic sematics of definition modules are to be applied.

The dynamic semantics of calls of procedures exported by a separate module are given explicitly for each such procedure and form part of an abstract implementation module for the separate module. Rules for binding abstract implementation modules to the rest of the formal definition are given in normative annex B.

3.3 Conventions

The following conventions used in this part of ISO/IEC 10514 are to be interpreted as described below.

3.3.1 Informative text

At certain points in this part of ISO/IEC 10514, informative, rather than normative text appears. In particular, such text often appears in introductory paragraphs immediately after clause or sub-clause headings (before the specification of concrete syntax or static semantics), and within the confines of sections where it is demarcated as notes, examples, clarifications, and changes, as exemplified below.

NOTE — Notes are usually printed in a smaller font than normative text.

The pseudo-definition modules of system modules are to be regarded as informative text.

Where necessary, parts of the VDM-SL are explained by means of informative natural language comments labelled as annotations.

3.3.2 Typographical conventions

The following typographical conventions are observed.

— Lexical production rules and concrete syntax rules are printed in a sans-serif font. Meta-identifiers are printed in this same font, e.g. national character.

— VDM-SL functions, operations and values are printed in an italic font, e.g. *t-variable-designator*.

— Where a function or operation is used in a VDM-SL function or operation, the page number where the function or operation is defined is given on the right-hand side of the page.

— Fragments of Modula-2 or keywords etc. are printed in a monospaced font, e.g.

```
CONST PI = 3.14159265358979323846264338327950288419 72;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

— Some definition modules of standard library modules or pseudo-definition modules of system modules contain implementation-defined types and implementation-defined constant values. These are shown between angle brackets < and >.

3.3.3 References to pervasive identifiers

This part of ISO/IEC 10514 gives meaning to a program in which identifiers spelt in the same way as the pervasive identifiers are introduced in definitions or declarations of new entities, unrelated to the predefined entities. In such a program the meaning of a pervasive identifier is overridden in the scope of an identifier of the same spelling that is introduced in a definition or a declaration. This part of ISO/IEC 10514 refers throughout to pervasive identifiers in a monospaced font (e.g. `BOOLEAN`), *with the implication that their original, predefined, meaning is intended.* For example, a reference to the type `BOOLEAN` is a reference to the predefined type denoted by the pervasive identifier `BOOLEAN`, and not to any local redefinition that might have been made.

3.3.4 Interpretation of non-VDM-SL notations

The definition modules that are used in the specification of library modules shall be interpreted according to the concrete syntax, declaration semantics, static semantics, and dynamic semantics of definition modules as specified in section 6. Modula-2 comments that appear in these definition modules shall be interpreted as notes.

The pseudo-definition modules that are used in the specification of system modules shall be interpreted as notes which outline the declaration semantics of the facilities provided by the system modules and the static semantics that apply to their use. (The declaration semantics, static semantics, and dynamic semantics are defined explicitly in the relevant clauses of section 7.)

Natural language that appears between question marks on the right hand of syntax production rules shall be interpreted as normative requirements.

Where the terms 'implementation-defined' and 'implementation-dependent' occur (in whatever notation, with or without hyphens), the requirements of section 4 (particularly documentation requirements) shall apply. For example, given the VDM-SL expression `implementation-defined:` *Char*`-set` section 4 requires implementors to provide the set of characters and to document any elements in that set.

3.3.5 References to formal parameter names

In order to allow succint descriptions of the meaning of procedures in standard library modules, a convention has been adopted by which explanation of requirements is given in terms of a procedure call with the names of formal parameters in the place of actual parameters. Hence the description refers to formal parameter names as if they denote variables; precision is provided in the VDM-SL definition which accompanies all natural language descriptions.

NOTE — For example, the module `Strings` provides a procedure having the following heading:

```
PROCEDURE Append (source: ARRAY OF CHAR; VAR destination: ARRAY OF CHAR);
```

The corresponding informal description of `Strings.Append` is:

> The call
>
> ```
> Append(source, destination)
> ```
>
> shall create a new string value by appending the string value in `source` onto `destination`. If the length of the created string value exceeds the capacity of `destination`, the string value shall be truncated to the capacity of `destination`, and the truncated value shall be assigned to `destination`. If the length of the created string value is less than the capacity of `destination`, a string terminator shall be appended to the string value, and the resulting value shall be assigned to `destination`.

This simplifies the description of the value parameter `source` inasmuch as it can always be referred to by name, and no distinction need be made between literal values and non-constant expressions.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Section 4 : Requirements for Implementations

A conforming Modula-2 implementation meets the requirements for Modula-2 implementations that are laid down in this section.

A conforming standard library implementation meets the requirements for standard library implementations that are laid down in this section.

4.1 Translation

A conforming Modula-2 implementation shall accept compilation modules for translation from source code in the lexical form specified in section 5 (subject to the provisions of 4.2), in the syntactic form specified in section 6, and in the static semantic form specified in sections 6 and 7, as required for the type of implementation specified in 4.3, and subject to minimum implementation limits as specified in 4.11. A conforming implementation may accept for translation non-text representations that are deriveable from such source code.

A conforming Modula-2 implementation shall translate the compilation modules of a program in such a way that the execution of the program on the computer system is in accordance with the dynamic semantics in sections 6 and 7.

NOTES

1 In sections 5, 6 and 7 a formal definition of the syntax and semantics of Modula-2 programs is given. The definition does not explicitly state the requirements for Modula-2 implementaions. The present section states the requirement for an implementation to implement the semantics in sections 6 and 7, while allowing for specified minimum limits on the size and complexity of programs to be translated, allowing for implementation restrictions on the processing of preferred lexical symbols, and requiring that implementation-defined values be defined.

2 Since VDM-SL is, by its mathematical nature, very precise, it has been necessary to choose one particular evaluation order in expressing the semantics, whereas at some points the natural language states that any evaluation order is permitted. The VDM-SL definition can thus be considered to be over-specification. When this situation arises, an annotation to the VDM-SL is provided to indicate such over-specification.

4.2 Source Code Representation

A conforming Modula-2 implementation shall:

— provide all the keywords and required symbols specified in section 5, and shall recognize identifiers and literals as specified in that section;

— provide the required left and right brackets;

— provide the preferred left and right brackets if its processor supports them;

— provide the required left and right braces;

— provide the preferred left and right braces if its processor supports them;

— provide the required case separator;

— provide the preferred case separator if its processor supports it;

— provide the required dereferencing operator;

— provide the preferred dereferencing operator if its processor supports it;

— process separators (white space, comments and source code directives) in such a way that they have no effect on the meaning of the program other than delimiting tokens, or, in the case of source code directives, by determining an implementation-defined value

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

— detect (and shall issue a warning concerning) a source code directive whose body is not recognized.

NOTES

1 This part of ISO/IEC 10514 does not specify how compilation modules be stored, nor how a conforming Modula-2 implementation accepts input — see 1.3.

2 The Lexis is formally expressed as a syntax that defines tokens in terms of characters. An implementation may accept a tokenized compilation module, that has been produced according to the syntax, rather than a stream of characters — see 4.1.

4.3 Ordering of Declarations

With respect to the two alternative predicates for declarations given in 6.2.7, a conforming Modula-2 implementation shall translate source code in one of two ways:

a) If the implementation is of a type identified as declare-before-use-in-declarations (see also 4.9a), it shall accept all features of the language; i.e. the first predicate in 6.2.7 shall apply.

b) If the implementation is of a type identified as declare-before-use (see also 4.9a), it shall reject any program not satisfying the second predicate in 6.2.7 and during translation shall give a unique indication of the reason for rejection.

NOTES

1 Clause 6.2.7 contains two alternative predicates for declarations that encapsulate the different approaches that implementors may choose to support. Hence, so-called 'multi-pass' implementations (i.e. those that require programs to observe a 'declare-before-use-in-declaration' rule) will support the predicate *wf-total-declarations* that returns true and so-called 'single-pass' implementations (i.e. those that require programs to observe a more strict 'declare-before-use' rule) will support the alternative predicate.

2 The requirement to provide a unique indication when the first predicate for declarations (in 6.2.7) is not satisfied does not preclude a conforming implementation from emitting other (possibly erroneous) indications first. The exact form of the required indication is not defined by this part of ISO/IEC 10514, due to the varying translation styles and user interfaces.

3 There is a requirement for all conforming implementations to accept and correctly process the **FORWARD** directive — see 6.2.11.1 and 6.2.11.2.

4.4 Predefined Entities

A conforming Modula-2 implementation shall provide all the predefined entities as specified in sections 6 and 7.

Implementations whose target computer system supports the addressing of 8-bit locations shall provide the type BYTE and the constant LOCSPERBYTE in the module SYSTEM as specified in 7.1. The names BYTE and LOCSPERBYTE shall not be used in any implementation of SYSTEM with meanings other than those specified in 7.1.

A conforming implementation may extend the system module SYSTEM by adding to the types, constants, variables or procedures that it exports.

NOTE — Whether or not an implementation does extend the system module SYSTEM, the other requirements of this section apply. In particular, since conforming implementations are required to process programs in the static semantic form specified in section 7, and execute programs in accordance with the dynamic semantics in section 7, no change may be made to the meaning of the types, constants, variables or procedures specified in 7.1.

4.5 Library Modules

A conforming Modula-2 implementation shall provide all the separate modules as specified in section 8 and those separate modules shall conform to the requirements for standard library module implementations.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

If any or all of the separate modules provided with a conforming Modula-2 implementation have the same module names as a separate module specified in section 9, those separate modules shall conform to the requirements for standard library module implementations.

In a conforming standard library module implementation, the provided modules shall be the same as the modules of the same name in section 8 or section 9, and shall not be extended or reduced with respect to the types, constants, variables or procedures that they export.

NOTES

1 The requirement not to 'reduce' a module is meant to prevent features specified by this part of ISO/IEC 10514 from being omitted in an implementation.

2 A conforming Modula-2 implementation or standard library implementation may provide additional modules, with names that are different from the names used in section 8 and section 9; for example an implementation may provide an additional device module, for use with the input/output library, to access the specific facilities of the filesystem of an underlying operating system.

3 A conforming Modula-2 implementation or standard library implementation may not provide any module that has the same name as one of those modules specified in sections 8 and 9, but which is different from that module. For example, an alternative strings module may not be called `Strings`.

4 With the exception of the module `Semaphores`, none of the operations in the separate modules specified in sections 8 and 9 is required to be atomic in the presence of processes or coroutines. For example, suppose the input stream to a program were

```
Hello Professor, we hope you are well.
```

and two concurrent calls of `TextIO.ReadToken` (from the I/O library in 9.2.2.1.2) were made in this program for input to distinct character arrays. It is not required that, after the calls are complete, one array contain `Hello` and the other `Professor`. Such a program is incorrect (because it shares a common resource between concurrent activities without arranging for mutual exclusion between them); its behaviour is not defined.

5 In some cases (in the definition of the I/O library and elsewhere) in the interests of clarity of exposition, the formal definition makes lavish use of resources (e.g. by constructing a large set of items, only to pick one and discard the rest, or by using a buffer as large as the file that is attached to the channel). The implementation is not expected to mimic this specification exactly. Indeed, providing that the specified meaning is achieved, a more parsimonious approach is to be encouraged.

4.6 Errors

A conforming Modula-2 implementation shall determine whether or not a program contains errors and shall indicate the result of this determination to the user of the implementation. If the implementation has not attempted to discover all errors, this fact shall also be indicated to the user.

In standard mode a conforming implementation shall treat the use of extensions as errors.

NOTES

1 Whereas a conforming implementation may extend the system module `SYSTEM` (see 4.4), a program that uses such an extension, and hence relies on a definition of the extension is not given meaning by this part of ISO/IEC 10514 — see 1.3.

2 While it is expected that implementations will not issue misleading error indications, it is likely that declare-before-use implementations, when issuing error indications just prior to reporting rejection of a program not satisfying the second predicate in 6.2.7, may issue misleading indications before a declare-before-use error is properly detected and reported. (See also 4.3.)

4.7 Exceptions

A conforming Modula-2 implementation shall treat the occurrence of an exception in the following manner:

a) In standard mode it shall detect (either before or during execution) the occurrence of all language exceptions whose detection is mandatory (specified by the use of the function *mandatory-detection* in sections 6, 7, 8, and 9).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

b) It shall be free to detect, or not to detect, language exceptions whose detection is not mandatory (specified by use of the function *non-mandatory-detection* in sections 6, 7, 8, and 9). The documentation requirements of 4.9f shall apply to any modes of use that may be utilized to provide such detection.

NOTE 1 — The language exceptions that must be detected by an implementation and the functions *mandatory-detection* and *non-mandatory-detection* are given in 7.5.4.

A conforming standard library module implementation shall treat the occurrence of an exception in the following manner:

a) In standard mode it shall detect (either before or during execution) all exceptions defined by standard library modules whose detection is mandatory (as specified in sections 8, and 9).

b) It shall be free to detect and report, or not to detect, exceptions defined by required separate modules and supplied standard library modules whose detection is not mandatory (as specified in sections 8, and 9). The documentation requirements of 4.9f shall apply to any modes of use that may be utilized to provide such detection.

During exceptional termination, a Modula-2 implementation, where appropriate, shall propagate an indication of the raised exception to the environment from which the program was invoked.

NOTES

2 It is expected that implementations will allow users to choose whether or not to detect exceptions by selecting some 'mode of use'. To conform to this part of ISO/IEC 10514 an implementation must provide a standard mode in which it will detect all exceptions whose detection is mandatory.

3 The requirement to propagate an indication to the environment from which the program was invoked when appropriate is designed to avoid reference to the program user (since there may not be one, for example, in an embedded system application). It is accepted that indicating an exception from an embedded system application may have no meaning. In the formal definition, this requirement is captured by the function *exception-actions* in 6.12.11 The indication may or may not be derived from the implementation-defined text string that is supplied when an exception is raised.

4 There is no implied requirement that implementors provide a 'trace-back' facility during exceptional termination.

4.8 Implementation-dependencies

A conforming Modula-2 implementation shall be free whether to detect and report, or otherwise process, any reliance upon a definition of implementation-dependent values or implementation-dependent behaviour (specified in sections 6 to 10).

NOTE — A program that relies upon a definition of an implementation-dependent value or implementation-dependent beahviour, such as the order of evaluation of parameters, is not given meaning by this part of ISO/IEC 10514 — see 1.3. However, few production translators will be able to detect this situation reliably, and so 4.8 cannot be more specific about the processing of such a program.

4.9 Documentation

A conforming implementation shall be accompanied by documentation that shall include the following:

a) a statement as to whether the implementation supports the 'declare-before-use-in-declarations' or 'declare-before-use' semantics (see 4.3 and 6.2.7);

b) a statement on how to use the implementation in standard mode;

c) a definition of all implementation-defined values, including rules for the determination of values that are not constants;

d) rules for ascertaining the amount of workspace that is needed for a single activation of a particular procedure (without taking into account the workspace needed by any procedure that it calls);

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

e) a statement defining the layout strategy in mapping components of a variable of a structured (composite) type onto the addressable locations of the target computer system.

f) a statement defining the exception-detection capability of the implementation; this statement shall specify the influence of any relevant implementation modes (or options) that are provided;

g) a general statement about the processing of programs that rely upon a definition of an implementation-dependent value or of implementation-dependent behaviour, especially where such uses are treated as undetected errors or exceptions;

h) a statement describing any features accepted by the implementation that are extensions, and a statement on how it is possible to detect the use of such extensions, or prevent their use;

i) a statement describing any additional separate modules that are supplied (i.e. modules other than those described in sections 8, and 9);

j) a statement listing the names of any additional separate modules (i.e. modules other than those described in sections 8, and 9) that are used by the implementation of the standard library modules that are provided.

NOTE 1 — Requirement 4.9j is made so that the user of an implementation is aware of potential clashes with the names of other separate modules used in a program.

k) information about the facilities that are available to enable users to produce equivalent versions of all the standard library modules that are provided with the implementation, and the standard library modules that are not provided with the implementation but for which the information is relevant to the type of the implementation.

NOTE 2 — Requirement 4.9k is to be interpreted such that, for example, cross-translators for embedded systems applications do not need to be accompanied by information on facilities to implement, say, the standard input/output library, if that library is not itself provided with the implementation. The intention is to allow the separate modules to be replaceable, even if they are modules such as input/output modules that need to access an underlying system, without requiring the facilities for such access to be specified by this part of ISO/IEC 10514.

4.10 Statement of Compliance

A conforming implementation shall be accompanied by a compliance statement, conforming to the requirements below. In the following, the text <This implementation> shall be replaced by an unambiguous name identifying the implementation.

If the implementation complies in *all* respects with the requirements of this part of ISO/IEC 10514, the compliance statement shall be:

"<This implementation> complies with the requirements of ISO 10514-1."

If the implementation complies with some but not all of the requirements of this part of ISO/IEC 10514, the compliance statement shall be:

"<This implementation> partially complies with the requirements of ISO 10514-1.
The details of the non-conformities are as follows:"

followed by a complete list of the requirements of this part of ISO/IEC 10514 with which the implementation does not comply.

4.11 Minimum requirements

A conforming implementation shall allow a value of a type declared as SET OF CHAR to be a value that includes all values of the character type that are provided by the implementation.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

If an implementation places a restriction on the number of values of the base type of a set type such that the number of values can be at most N, then the implementation shall allow the base type to be any ordinal type that has at most N values.

If an implementation places a restriction on the number of values of the base type of a packedset type, such that the number of values can be at most N, then the implementation may restrict the base type to be a type whose values have ordinal numbers in the range 0 to $(N - 1)$.

An implementation may impose restrictions on the types that are allowed as the first parameter in calls of `SYSTEM.ROTATE` and `SYSTEM.SHIFT`, provided that at least the type `BITSET` is accepted.

A conforming implementation shall be such that the limits it imposes on the attributes specified in normative annex A shall be at least as great as the minimum limits specified in that annex (which are summarized in the table below).

NOTES

1 Annex A provides Modula-2 *limit-specifier programs* to generate programs which, when the limit-specifier programs have been parameterized by defining the constant `MinLimit`, a conforming implementation must successfully translate and execute. Thus the annex specifies each attribute using a Modula-2 program and specifies the minimum limit an implementation may impose on the attribute in the same program.

2 The table below summarizes the attributes and minimum limits specified by the limit-specifier programs in annex A. Where a number is marked *** that number is the value of the minimum limit for an attribute only for implementations in which `SYSTEM.BITSPERLOC * SYSTEM.LOCSPERWORD` (the size in bits of `SYSTEM.WORD`) is greater than or equal to 16; for implementations in which `SYSTEM.BITSPERLOC * SYSTEM.LOCSPERWORD` is less than 16, the value of the minimum limit for such an attribute is implementation-defined.

Clause giving limit-specifier program	*General description of attribute*	*Value of* `MinLimit`	
A.1.1	length of identifiers	32	
A.1.2	length of string literals	32	***
A.1.3	size of single dimension arrays	4096	***
A.1.4	number of dimensions of multidimension arrays	16	
A.1.5	depth of nested records	16	
A.1.6	number of sequential local modules	16	
A.1.7	depth of nested local modules	16	
A.1.8	number of enumeration literals	256	***
A.1.9	number of sequential procedure declarations	16	
A.1.10	depth of nested procedure declarations	16	
A.1.11	number of coroutines	16	***
A.1.12	number of parameters of a procedure	16	***
A.1.13	depth of nested IF-THEN-ELSEs in THEN-branch	16	
A.1.14	depth of nested IF-THEN-ELSEs in both branches	4	
A.1.15	depth of nested LOOP statements	16	
A.1.16	depth of nested WHILE statements	16	
A.1.17	depth of nested REPEAT statements	16	
A.1.18	depth of nested FOR statements	16	***
A.1.19	depth of nested WITH statements	16	
A.1.20	number of case alternatives with else part	255	***
A.1.21	depth of nested parentheses in expressions	16	
A.1.22	depth of recursion	256	***
A.1.23	number of imported modules	16	

3 The precise meaning of the attributes listed above is given by the limit-specifier programs in annex A; the values above are the minimum which conforming implementations are required to be able to accept. Implementors are encouraged to provide for acceptance of values that are greater than the minimal limits wherever possible.

4 Users should expect implementations to exhibit other limitations and should expect interactions when combining requirements.

5 Executing a limit-specifier program will generate Modula-2 source code which has to be accepted (that is, translated and run without any unexpected errors occurring) by a conforming implementation. Except for those cases where it is the intention to specify multi-module requirements, generator programs are deliberately formulated such that there will be one program module generated per attribute.

6 In processors involving both host and target systems, the values in the table without the *** are relevant to the capabilities of the host computer system, while those marked by *** relate to the value of `SYSTEM.BITSPERLOC * SYSTEM.LOCSPERWORD` of the target.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Section 5 : The Lexis

The Modula-2 Lexis is the specification of the lexical elements of Modula-2 source code including white space, comments, source code directives, and the tokens that are used in the concrete syntax. All the symbols and pervasive identifiers of Modula-2 are defined in this section.

5.1 Source Code Structure

modula2 source code = { delimited token }, { separator } ;

delimited token = { separator }, token ;

token = word token | symbol | constant literal ;

NOTE — A token is the largest sequence of characters that satisfies the definitions below. Consequently, a word token must be separated from a following word token, numeric literal, string literal or character number literal. Furthermore, a numeric literal must be separated from a following numeric literal or character number literal.

5.2 Word tokens

word token = identifier | keyword ;

5.2.1 Identifiers

identifier = portable identifier | full identifier | pervasive identifier ;

5.2.2 Portable Identifiers

portable identifier = (simple letter | low line), { simple alphanumeric | low line } ;

low line = "_" ;

CHANGE — The lowline symbol ('_') is permitted (and may be repeated) anywhere in an identifier.

5.2.3 Full Identifiers

full identifier = (national letter | low line), { national alphanumeric | low line } ;

5.2.4 Pervasive Identifiers

pervasive identifier =

"ABS"	"BITSET"	"BOOLEAN"	"CARDINAL"
"CAP"	"CHR"	"CHAR"	"COMPLEX"
"CMPLX"	"DEC"	"DISPOSE"	"EXCL"
"FALSE"	"FLOAT"	"HALT"	"HIGH"
"IM"	"INC"	"INCL"	"INT"
"INTERRUPTIBLE"	"INTEGER"	"LENGTH"	"LFLOAT"
"LONGCOMPLEX"	"LONGREAL"	"MAX"	"MIN"
"NEW"	"NIL"	"ODD"	"ORD"
"PROC"	"PROTECTION"	"RE"	"REAL"
"SIZE"	"TRUE"	"TRUNC"	"UNINTERRUPTIBLE"
"VAL" ;			

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

CHANGE — The following pervasive identifiers have been added to the language:

COMPLEX, CMPLX, IM, INT, INTERRUPTIBLE, LENGTH, LFLOAT, LONGCOMPLEX, LONGREAL, PROTECTION, RE and UNINTERRUPTIBLE.

NOTE — Pervasive identifiers are not reserved words; if redeclared in a program, they will no longer have their predefined meaning in the scope of the redeclared identifier; see 6.2 and see also 6.11.2.

5.2.5 Keywords

```
keyword =
    "AND"              | "ARRAY"          | "BEGIN"          | "BY"          |
    "CASE"             | "CONST"          | "DEFINITION"     | "DIV"         |
    "DO"               | "ELSE"           | "ELSIF"          | "END"         |
    "EXIT"             | "EXCEPT"         | "EXPORT"         | "FINALLY"     |
    "FOR"              | "FORWARD"        | "FROM"           | "IF"          |
    "IMPLEMENTATION"   | "IMPORT"         | "IN"             | "LOOP"        |
    "MOD"              | "MODULE"         | "NOT"            | "OF"          |
    "OR"               | "PACKEDSET"      | "POINTER"        | "PROCEDURE"   |
    "QUALIFIED"        | "RECORD"         | "REM"            | "RETRY"       |
    "REPEAT"           | "RETURN"         | "SET"            | "THEN"        |
    "TO"               | "TYPE"           | "UNTIL"          | "VAR"         |
    "WHILE"            | "WITH" ;
```

CHANGE — The keywords EXCEPT, FINALLY, FORWARD, PACKEDSET, REM and RETRY have been added to the language.

NOTE — The keywords AND, DIV, IN, MOD, NOT, OR and REM are operator keywords; the rest are punctuation keywords.

5.3 Symbols and Operators

```
symbol = required symbol | symbol with alternatives | operator ;
```

5.3.1 Required Symbols

```
required symbol =
    colon              | comma            | ellipsis         | equals        |
    period             | semicolon        | left parenthesis | right parenthesis ;

colon = ":" ;

comma = "," ;

ellipsis = ".." ;

equals = "=" ;

period = "." ;

semicolon = ";" ;

left parenthesis = "(" ;

right parenthesis = ")" ;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.3.2 Symbols with alternatives

symbol with alternatives =
    left bracket | right bracket | left brace | right brace | case separator ;

left bracket = preferred left bracket | required left bracket ;

preferred left bracket = "[" ;

required left bracket = "(!" ;

right bracket = preferred right bracket | required right bracket ;

preferred right bracket = "]" ;

required right bracket = "!)" ;

CHANGE 1 — All implementations are required to provide '(!' and '!)' as alternatives to '[' and ']'.

left brace = preferred left brace | required left brace ;

preferred left brace = "{" ;

required left brace = "(:" ;

right brace = preferred right brace | required right brace ;

preferred right brace = "}" ;

required right brace = ":)" ;

CHANGE 2 — All implementations are required to provide '(:' and ':)' as alternatives to '{' and '}'.

case separator = preferred case separator | required case separator ;

preferred case separator = "|" ;

required case separator = "!" ;

CHANGE 3 — All implementations are required to provide '!' as an alternative to '|'.

5.3.3 Operators

operator =

assignment operator	plus operator	set union operator
string catenate symbol	minus operator	set difference operator
logical disjunction operator	multiplication operator	division operator
div operator	mod operator	rem operator
set intersection operator	symmetric set difference operator	logical conjunction operator
sign	logical negation operator	equals operator
inequality operator	less than operator	greater than operator
less than or equal operator	greater than or equal operator	subset operator
superset operator	set membership operator	dereferencing operator ;

assignment operator = ":=" ;

plus operator = "+" ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

set union operator = "+" ;

string catenate symbol = "+" ;

CHANGE 1 — The string catenate symbol has been added.

minus operator = "-" ;

set difference operator = "-" ;

logical disjunction operator = "OR" ;

multiplication operator = "*" ;

division operator = "/" ;

div operator = "DIV" ;

mod operator = "MOD" ;

rem operator = "REM" ;

set intersection operator = "*" ;

symmetric set difference operator = "/" ;

logical conjunction operator = and keyword | and synonym ;

and keyword = "AND" ;

and synonym = "&" ;

sign = identity operator | arithmetic negation operator ;

identity operator = "+" ;

arithmetic negation operator = "-" ;

logical negation operator = not keyword | not synonym ;

not keyword = "NOT" ;

not synonym = "~" ;

equals operator = "=" ;

inequality operator = not equals operator | not equals synonym ;

not equals operator = "<>" ;

not equals synonym = "#" ;

less than operator = "<" ;

greater than operator = ">" ;

less than or equal operator = "<=" ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

greater than or equal operator = ">=" ;

subset operator = "<=" ;

superset operator = ">=" ;

set membership operator = "IN" ;

dereferencing operator = preferred dereferencing operator | required dereferencing operator ;

preferred dereferencing operator = "^" ;

required dereferencing operator = "@" ;

CHANGE 2 — All implementations are required to provide '@' as an alternative to '^'

5.4 Constant Literals

5.4.1 Whole Number Literals

whole number literal = decimal number | octal number | hexadecimal number ;

decimal number = digit, { digit } ;

digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;

octal number = octal digit, { octal digit }, "B" ;

octal digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" ;

hexadecimal number = digit, { hex digit }, "H" ;

hex digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" | "A" | "B" | "C" | "D" | "E" | "F" ;

5.4.2 Real Literals

real literal = digit, { digit }, ".", { digit }, [scale factor] ;

scale factor = "E", [sign], digit, { digit } ;

5.4.3 String Literals

string literal = quoted string | character number literal ;

quoted string =
    single quote, { quotable character - single quote }, single quote |
    double quote, { quotable character - double quote }, double quote ;

single quote = "'" ;

double quote = '"' ;

quotable character = national graphic character | space | white space control ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NOTES

1 A quoted string cannot contain both single and double quote characters (since one or the other will mark the end of the string).

2 A quoted string cannot cross a line boundary.

character number literal = octal digit, { octal digit }, "C" ;

5.5 Separators

separator = white space | comment | source code directive ;

5.5.1 White Space

white space = space | new line | white space control ;

space = " " ;

new line = ? possibly empty implementation-defined character sequence ? ;

white space control = ? implementation-defined character sequence that shall not include new line ? ;

5.5.2 Comments

comment = "(*", comment body, "*)" ;

comment body = { national character | separator } ;

NOTE — Since comments may be nested, the addition of comment delimiters around program text that has quoted strings containing comment delimiters may result in a different program from that which was intended (and probably in an incorrect one).

5.5.3 Source Code Directives

source code directive = "<*", directive body, "*>" ;

directive body = comment body ;

CHANGE — A syntax for source code directives has been introduced.

5.6 Letters and National Characters

CHANGE — The concept of what characters are permissible as letters (for use in identifiers) and what characters are permissible within quoted strings, comments and source code directives has been extended, so as to facilitate a widening variety of character sets.

national character = national graphic character | format effector | space ;

national graphic character =
    simple alphanumeric | national alphanumeric |
    implementation-defined graphic | punctuation character ;

simple alphanumeric = simple letter | digit ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

simple letter = "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" ;

national alphanumeric = national letter | national numeric ;

national letter = ? implementation-defined alphabetic character ? ;

national numeric = ? implementation-defined numeric character ? ;

implementation-defined graphic = ? implementation-defined character ? ;

punctuation character = ? implementation-defined punctuation character ? - national alphanumeric ;

format effector = ? implementation-defined format effector ? ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Section 6 : The Language

This section describes the structure and meaning of a Modula-2 program by specifying its components and stating the rules governing their combination. The major components are the compilation modules: program modules, definition modules, and implementation modules. The components of these modules include import and export lists, definitions and declarations of constants, types, variables, procedures and local modules, and statements and expressions which use these entities.

There are some constants, types, and procedures that are predefined. Those with pervasive identifiers are described in this section. Those that must be accessed using import lists (from system modules) are described in section 7.

The meaning of an identifier at the point of its use is specified by the environment that applies at that point — see 6.11.1. An environment associates an identifier with an entity and with the properties of that entity. An environment that applies at a point of use of an identifier is constructed from the environment that applies in a textually enclosing module or procedure, from import lists, and from defining occurrences in definitions and declarations. Environments that are constructed to express the well-formedness conditions refer only to properties that are relevant to those conditions, such as the types of variables and parameters. Environments that are constructed to express the meaning of a program refer to properties required for execution, including the storage for variables and the blocks of procedures. If an identifier is not visible in a constructed environment, an environment applies in which only the pervasive identifiers are visible.

The meaning of a program is described by the effect that its execution has on the external state (which may include the state of data files). The effect of the executable components of a program is described by the changes to an internal program state which includes the values of variables — see 6.12.1.

6.1 Programs, Program Modules, and Separate Modules

6.1.1 Programs and Compilation Modules

The major components of a Modula-2 program are the compilation modules, so-called because they can be treated separately, although not independently, by an implementation. There is, however, no requirement that the treatment of compilation modules involves actual compilation.

A program module is a compilation module that may use entities exported by separate modules, but does not itself export entities for use by other modules. Separate modules may in turn use entities from other separate modules. The entities that are available for use by clients of a separate module are defined in a compilation module called a definition module. The definition module must exist before the well-formedness of client modules can be checked. This also imposes a constraint on the compilation order — the order in which compilation modules may be elaborated. A corresponding compilation module called an implementation module is needed to complete each separate module before the program can be executed. The actions that are taken on program execution are given in initialization and finalization bodies within the program module and within the implementation modules.

A Modula-2 program consists of a program module together with the separate modules it uses (and the separate modules they in turn use).

Concrete Syntax

```
compilation module = program module | definition module | implementation module ;
```

NOTE 1 — There is no production for a Modula-2 program, or for a separate module, because of the separate treatment of compilation modules.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Abstract Syntax

types

Compilation-module = *Program-module*
| *Definition-module*
| *Implementation-module*;

Program :: *pmod* : *Program-module*
dmods : *Definition-modules*
imods : *Implementation-modules*;

Definition-modules = *Definition-module*-set;

Implementation-modules = *Implementation-module*-set

Static Semantics

For a particular program there shall be a single program module. For each separate module directly or indirectly imported by the program module, there shall be a single definition module and a single consistent corresponding implementation module — see 6.1.5.1.

NOTE 2 — This part of ISO/IEC 10514 defines the meaning of a program given the implementation modules supplied for the corresponding definition modules. This does not preclude an implementation from allowing prior selection from alternative implementation modules for a given definition module.

The identifier of the program module shall not be the same as the identifier of any separate module that forms part of the program.

NOTE 3 — The identifier of a separate module is the module identifier of its definition module and its corresponding implementation module.

The identifiers of the program module, and the identifiers of the separate modules, shall not be the same as the identifier of one of the system modules.

The use of separate modules by compilation modules shall be such that there exists at least one complete and valid order for the separate treatment of the compilation modules of the program — see 6.1.5.2.

NOTE 4 — If there are many possible orders that satisfy the constraints given in 6.1.5.2, this part of ISO/IEC 10514 does not define which order is chosen, since the choice has no effect on the meaning of the program.

The outermost environment for checking well-formedness shall be constructed from the environments defined by the definition modules that form part of the program, together with the environment defined by the system modules.

NOTES

5 The outermost environment for checking well-formedness may be understood in terms of a procedure within which all declarations are made by export from local modules which correspond to the separate modules and the system modules. The definition modules correspond to qualified export lists in those local modules.

6 The environment defined by the system modules contains the identifiers associated with the constants, types, and procedures available by import from the system modules — see section 7.

wf-program : *Program* → $\mathbb{B}$

wf-program (*mk-Program* (*pmod*, *dmods*, *imods*)) $\triangleq$
  let ρ_{out} = *outer-defs-environment* (*dmods*) in 26
  wf-modules (*pmod*, *dmods*, *imods*)ρ_{out} $\wedge$ 25
  wf-program-module (*pmod*)ρ_{out} $\wedge$ 27
  $\forall$ *dmod* $\in$ *dmods* · *wf-definition-module* (*dmod*)ρ_{out} $\wedge$ 29
  $\forall$ *imod* $\in$ *imods* · *wf-implementation-module* (*imod*)ρ_{out} $\wedge$ 30
  compilation-orders (*pmod*, *dmods*, *imods*) $\neq$ { } 36

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wf-modules : *Program-module* × *Definition-modules* × *Implementation-modules* → *Environment* → 𝔹

wf-modules (*pmod*, *dmods*, *imods*)ρ ≙
  pmod.name ∉ *SYSTEM-MODULE-NAMES* ∧
  (∀ *dmod* ∈ *dmods* · *dmod.name* ≠ *pmod.name* ∧
    ∃! *imod* ∈ *imods* · *dmod.name* = *imod.name* ∧
    dmod.name ∉ *SYSTEM-MODULE-NAMES* ∧
    wf-module-pair (*dmod*, *imod*)ρ) ∧ 34
  (∀ *imod* ∈ *imods* · ∃! *dmod* ∈ *dmods* · *imod.name* = *dmod.name*) ∧
  wf-abstract-modules (*imods*) 25

wf-abstract-modules : *Implementation-modules* → 𝔹

wf-abstract-modules (*imods*) ≙
  ∀ *mk-Abstract-implementation-module* (*name*$_1$, -, -, *dens*$_1$, -),
  mk-Abstract-implementation-module (*name*$_2$, -, -, *dens*$_2$, -) ∈ *imods* ·
    name$_1$ ≠ *name*$_2$ ⇒ dom *dens*$_1$ ∩ dom *dens*$_2$ = { }

annotations The correspondence of definition and implementation modules and the correctness of each of the compilation modules are checked with respect to the definitions of the definition modules and the system modules.

Dynamic Semantics

The program state shall be initialized from the external state as it exists before execution of the program. The initial protection from interrupts shall be implementation-defined.

NOTES

7 No variables are defined in this initial program state, and none exist upon completion of the program.

8 The external state includes the storage locations that a Modula-2 program can have access to via an address that is calculated from the machine address component of a variable declaration — see 6.12.8.2.1.

The outermost environment for program execution shall be constructed from the environments defined by the separate modules that form part of the program, together with the environment defined by the system modules.

NOTES

9 The outermost environment for program execution may be understood in terms of a procedure within which the program module, separate modules, and system modules are local modules. The definition modules correspond to qualified export lists in those local modules.

10 The environment defined by the system modules contains the identifiers associated with the constants, types, and procedures available by import from the system modules — see section 7.

The program shall be executed by first initializing the separate modules, in the order defined by their occurrence in the import lists, and then initializing the program module — see 6.1.6, 6.1.4, and 6.1.2.

NOTE 11 — In a program with coroutines, this execution is performed by the main coroutine.

If an exception is raised during initialization, and this exception is not handled, the program shall be terminated exceptionally.

On program termination, the program module and the implementation modules shall be finalized in the reverse order of their initialization — see 6.1.7.

m-program : *Program* → *External-program-state* → *External-program-state*

m-program (*mk-Program* (*pmod*, *dmods*, *imods*))(*ext-state*) ≙
  let *initial-state* = *read-external-program-state* (*ext-state*) in 331
  execute-program (*pmod*, *dmods*, *imods*) *stop cenv initial-state* 26

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations *stop*, *cenv*, and *initial-state* are supplied as explicit arguments in the application of *execute-program* as a function. Corresponding arguments to these are implied in calls of operations.

execute-program : *Program-module* × *Definition-modules* × *Implementation-modules* $\xrightarrow{o}$ ()

execute-program (*pmod*, *dmods*, *imods*) $\triangleq$
  def ρ_{out} = *merge-modules*(*pmod*, *dmods*, *imods*) in 26
  def ρ_{cont} = *c-tixe*({EXCEPTION $\mapsto$ (*termination*() ; *exception-actions*())}) ρ_{out} in 345 41 341
  (*set-initializing*() ; 340
   let *ex-order* = *execution-order* (*pmod*, *dmods*, *imods*) in 39
   initialize-modules(*ex-order*) ρ_{cont}; 26
   initialize-program-module(*pmod*) ρ_{cont}; 28
   termination() ; 41
   termination-actions()) 341

Auxiliaries

outer-defs-environment : *Definition-modules* → *Environment*

outer-defs-environment (*defs*) $\triangleq$
  let ρ = let ρ_{defs} = *merge-environments* ({*d-definition-module* (*def*)ρ | *def* ∈ *defs*}) in 319 29
        merge-environments ({ρ_{defs}, $\rho_{sys\text{-}modules}$}) in 319
  ρ

annotations The definition modules of the program are taken to construct an environment corresponding to the definitions of the definition modules and the system modules.

merge-modules : *Program-module* × *Definition-modules* × *Implementation-modules* $\xrightarrow{o}$ *Environment*

merge-modules (*pmod*, *dmods*, *imods*) $\triangleq$
  let *order* ∈ *compilation-orders* (*pmod*, *dmods*, *imods*) in 36
  merge-compilation-modules(*order*) $\rho_{sys\text{-}modules}$ 26

annotations The set of valid orders of compilation is produced and one of these orders is chosen arbitrarily. The operation is modelling the link-edit process in constructing the outer environment for execution and allocating storage.

merge-compilation-modules : *Module-order* → *Environment* $\xrightarrow{o}$ *Environment*

merge-compilation-modules (*order*)ρ $\triangleq$
  if *order* = []
  then return ρ
  else def ρ_{new} = *m-compilation-module*(hd *order*) ρ in 26
     merge-compilation-modules(tl *order*) ρ_{new} 26

annotations The declarations of each module are elaborated in order. The order is such that all the definitions of all modules used by a client module have been added to the environment before that client module is processed. Elaboration of the declarations allocates storage and builds an environment, and the resulting outer environment for the execution of the program is constructed.

m-compilation-module : *Compilation-module* → *Environment* $\xrightarrow{o}$ *Environment*

m-compilation-module (*cm*)ρ $\triangleq$
  (*is-Program-module*(*cm*) → *m-program-module*(*cm*) ρ, 28
   is-Definition-module(*cm*) → *m-definition-module*(*cm*) ρ, 30
   is-Implementation-module(*cm*) → *m-implementation-module*(*cm*) ρ) 31

initialize-modules : *Execution-order* → *Environment* $\xrightarrow{o}$ ()

initialize-modules (*order*)ρ $\triangleq$
  if *order* = []
  then skip
  else (*initialize-an-implementation-module*(hd *order*) ρ; 31
     initialize-modules(tl *order*) ρ) 26

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations The implementation modules are initialized in the execution order defined by the imports.

6.1.2 Program Modules

A program module consists of a sequence of import lists followed by a module block, which comprises a sequence of declarations and an optional module body. Both of these sequences may be empty. The declarations may include the declaration of local modules.

The program module block is executed after the initialization of all implementation modules of the program.

A program module may have an interrupt protection specified in its heading. The statements and procedures of the module are executed under a protection from interrupts as specified by the protection expression — see 6.1.11.

Concrete Syntax

```
program module  =
    "MODULE", module identifier, [ interrupt protection ], semicolon,
    import lists,
    module block, module identifier, period  ;

module identifier  =  identifier  ;
```

The two module identifier components of a program module shall be identical.

Abstract Syntax

types

Program-module :: *name* : *Identifier*
imports : *Import-lists*
block : *Module-block*
protection : [*Interrupt-protection*]

Static Semantics

The program module shall not import itself. If present, the import lists of the program module shall be valid given the outermost environment — see 6.1.8. If present, the import lists and the declarations of the program module block shall be consistent with one another — see 6.1.8.5.

Given the identifiers of the system modules and the separate modules that are imported into the program module, and given the identifiers that are imported from those modules:

— the declarations of the module block of the program module shall be well-formed — see 6.2.6,

— the bodies of module and procedure declarations of the module block shall be well-formed — see 6.2.7,

— if present, the body of the module block shall be a well-formed module body — see 6.5.3.

If present in the program module heading, the interrupt protection shall be a valid interrupt protection in the context of the pervasive identifier environment only — see 6.1.11.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wf-program-module : *Program-module* → *Environment* → 𝔹

wf-program-module (*mk-Program-module* (*name*, *imports*, *block*, *protection*))ρ_{out} ≜
  name ∉ elems *directly-imports* (*imports*) ∧
  wf-import-lists (*imports*)ρ_{out} ∧ 42
  let *mk-Module-block* (*decls*, -, -) = *block* in
  consistent-imports (*imports*, *decls*)ρ_{out} ∧ 46
  let ρ_{import} = *d-import-lists* (*imports*)ρ_{out} in 42
  (*wf-declarations* (*decls*)ρ_{import} ∧ 65
  wf-declaration-bodies (*decls*)ρ_{import} ∧ 67
  wf-module-body (*block*)ρ_{import}) ∧ 132
  (*protection* = nil ∨ *wf-interrupt-protection* (*protection*)ρ_{empty}) 51

Dynamic Semantics

The elaboration of the program module shall elaborate the declarations of the module block, given the meaning of the imported identifiers.

If the program module is a protected module, the protection specified for the module shall be used to protect the protection domain of the program module. The protection shall be evaluated in the context of the pervasive identifier environment only.

m-program-module : *Program-module* → *Environment* $\xrightarrow{o}$ *Environment*

m-program-module (*mk-Program-module* (*name*, *imports*, *block*, *protection*))ρ ≜
  let ρ_{import} = *d-import-lists* (*imports*)ρ in 42
  def *pval* = if *protection* = nil then nil else *evaluate-interrupt-protection*(*protection*) ρ_{empty} in 51
  let $\rho_{internal}$ = *add-protection* (*pval*)ρ_{import} in 318
  def ρ_{block} = *m-module-block*(*block*) $\rho_{internal}$ in 132
  return *overwrite-mod-environment* ({*name* ↦ ρ_{block}})ρ 317

annotations The compilation order is such that, when this operation is called, the result of elaboration of all definition modules used by the definition module have been merged in the given environment. The protection expression (if present) is evaluated in an environment consisting of the pervasive identifiers only, and is added to the returned environment for use by the protection mechanism.

The program module shall be initialized by initializing the program module block given the environment resulting from the elaboration of the program module — see 6.5.4.

initialize-program-module : *Program-module* → *Environment* $\xrightarrow{o}$ *Environment*

initialize-program-module (*mk-Program-module* (*name*, -, *block*, -))ρ ≜
  let ρ_{block} = *associated-module* (*name*)ρ in 317
  execute-static-module-initialization(*block*) ρ_{block} 132

annotations The environment for initializing the module block of the program module is retrieved from the given environment.

6.1.3 Definition Modules

A definition module consists of a sequence of import import lists and a sequence of definitions only. Both of these sequences may be empty. The identifiers introduced by the definitions denote entities that are to be available for use in other modules by import from the corresponding separate module — see 6.2.3.

In the case of procedure definitions, only the procedure headings are shown. This is sufficient to allow calls of these procedures in other modules to be checked for consistency, and for those modules to be prepared for execution.

In the case of type definitions, a complete declaration may be given, or the definition may specify only the type identifier, resulting in what is called an opaque type. If the type identifier is defined as the identifier of an enumeration type, the enumeration constant identifiers are also available for import from the corresponding separate module.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

CLARIFICATION — This part of ISO/IEC 10514 adopts the position taken in later editions of *Programming in Modula-2* of not allowing export lists in definition modules.

Concrete Syntax

definition module =
"DEFINITION", "MODULE", module identifier, semicolon,
import lists, definitions,
"END", module identifier, period ;

The two module identifier components of a definition module shall be identical.

Abstract Syntax

types

Definition-module :: *name* : *Identifier*
imports : *Import-lists*
defs : *Definitions*

Declaration Semantics

The export closure of the identifiers introduced by the definitions of a definition module, and the entities they denote, shall be associated with the identifier of the definition module in the outermost environment.

NOTE — The export closure includes the identifiers implicitly exported when an identifier is explicitly exported — see 6.1.10.

d-definition-module : *Definition-module* → *Environment* → *Environment*
d-definition-module (*mk-Definition-module* (*name*, *imports*, *defs*))ρ_{out} ≜
let ρ_{import} = *d-import-lists* (*imports*)ρ_{out} in 42
let ρ_{def} = *overwrite-environment* (*d-definitions* (*defs*)ρ_{def})ρ_{import} in 319 54
let *ids* = *identifiers-defined-in* (*defs*) in 82
let ρ_{local} = *restrict-environment* (*ids*)ρ_{def} in 320
let *xids* = *x-closure* (*ids*)ρ_{def} in 49
let ρ_{mod} = *restrict-environment* (*xids*)ρ_{def} in 320
merge-environments ({*construct-module-environment* (*name*, ρ_{mod}), 319 317
construct-local-environment (*name*, ρ_{local})}) 317

annotations The given environment is restricted by the import lists, and then used in the construction of the environment from the definitions of the definition module. The constructed module environment associates the name of the module with the export closure of the identifiers and entities defined in the module and is used for import from the module. The constructed local environment associates the name of the module with the identifiers and entities defined in the module, and is used within the corresponding implementation module.

Static Semantics

A definition module shall not import from its own separate module. If present, the import lists of a definition module shall be valid given the outermost environment — see 6.1.8. If present, the import lists and the definitions of a definition module shall be consistent with one another — see 6.1.8.4.

The definitions of the definition module shall be well-formed given the identifiers of the system modules and the separate modules that are imported into the definition module, and given the identifiers that are imported from those modules.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wf-definition-module : *Definition-module* → *Environment* → $\mathbb{B}$

wf-definition-module (*mk-Definition-module* (*name*, *imports*, *defs*))ρ_{out} $\triangleq$
  name $\notin$ elems *directly-imports* (*imports*) $\wedge$
  wf-import-lists (*imports*)ρ_{out} $\wedge$ 42
  consistent-imports-in-definitions (*imports*, *defs*)ρ_{out} $\wedge$ 45
  let ρ_{import} = *d-import-lists* (*imports*)ρ_{out} in 42
  wf-definitions (*defs*)ρ_{import} 55

Dynamic Semantics

The elaboration of a definition module shall elaborate the definitions of the definition module, given the meaning of the imported identifiers.

m-definition-module : *Definition-module* → *Environment* $\xrightarrow{o}$ *Environment*

m-definition-module (*mk-Definition-module* (*name*, *imports*, *defs*))ρ $\triangleq$
  let ρ_{import} = *d-import-lists* (*imports*)ρ in 42
  def ρ_{def} = *m-definitions*(*defs*) ρ_{import} in 56
  let *ids* = *identifiers-defined-in* (*defs*) in 82
  let ρ_{local} = *restrict-environment* (*ids*)ρ_{def} in 320
  let *xids* = *x-closure* (*ids*)ρ_{def} in 49
  let ρ_{mod} = *restrict-environment* (*xids*)ρ_{def} in 320
  return *merge-environments* ({ρ, ρ_{mod}, ρ_{local}}) 319

annotations The compilation order is such that, when this operation is called, the result of elaboration of all definition modules used by the definition module have been merged in the given environment.

6.1.4 Implementation Modules

The formal model in this part of ISO/IEC 10514 provides two ways by which the dynamic semantics of an implementation of a separate module may be determined:

— from the source text of a Modula-2 implementation module for the separate module,

— from a formal specification of the separate module, as given for the standard library modules in sections 8 and 9.

The meaning of a program that uses a standard library module is obtained by producing an abstract implementation module directly from the specification. The formal model of an implementation module for which Modula-2 source is submitted is called a sourced implementation module in this part of ISO/IEC 10514.

Abstract Syntax

types

Implementation-module = *Sourced-implementation-module* | *Abstract-implementation-module*

Static Semantics

wf-implementation-module : *Implementation-module* → *Environment* → $\mathbb{B}$

wf-implementation-module (*imod*)ρ_{out} $\triangleq$
  (*is-Sourced-implementation-module*(*imod*) → *wf-sourced-implementation-module* (*imod*)ρ_{out}, 32
   is-Abstract-implementation-module(*imod*) → *wf-abstract-implementation-module* (*imod*)ρ_{out}) 33

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Dynamic Semantics

m-implementation-module : *Implementation-module* → *Environment* $\xrightarrow{o}$ *Environment*

m-implementation-module (*im*)ρ $\triangleq$
(*is-Sourced-implementation-module*(*imod*) → *m-sourced-implementation-module*(*imod*) ρ, 32
is-Abstract-implementation-module(*imod*) → *m-abstract-implementation-module*(*imod*) ρ) 33

initialize-an-implementation-module : *Implementation-module* → *Environment* $\xrightarrow{o}$ ()

initialize-an-implementation-module (*im*)ρ $\triangleq$
(*is-Sourced-implementation-module*(*im*) → *initialize-a-sourced-implementation-module*(*im*) ρ, 33
is-Abstract-implementation-module(*im*) → *initialize-an-abstract-implementation-module*(*im*) ρ) 33

6.1.4.1 Sourced Implementation Modules

An implementation module in source form consists of a sequence of import lists followed by a module block, which comprises a sequence of declarations and an optional module body. Both of these sequences may be empty. The source code for an implementation module is only distinguishable from the source code for a program module by the appearance of the keyword `IMPLEMENTATION` as the first token.

The meaning of procedures and opaque types defined in the definition module is expressed by full declarations in the implementation module. The declarations may include the declaration of local modules and other entities hidden from clients of the separate module.

After the initialization of the implementation modules of imported separate modules, any local modules declared in the implementation module are initialized and the initialization body of the implementation module (if any) is executed. This order of initialization allows actions to be taken before exported procedures are called from client modules and is guaranteed, provided that no separate module indirectly imports itself — see 6.1.6.

An implementation module may have an interrupt protection specified in its heading. The statements and procedures of the module are executed under a protection from interrupts as specified by the protection expression — see 6.1.11.

Concrete Syntax

implementation module =
"IMPLEMENTATION", "MODULE", module identifier, [interrupt protection], semicolon,
import lists,
module block, module identifier, period ;

The two module identifier components of an implementation module shall be identical.

Abstract Syntax

types

Sourced-implementation-module :: *name* : *Identifier*
imports : *Import-lists*
block : *Module-block*
protection : [*Interrupt-protection*]

Static Semantics

An implementation module shall not import from its own separate module. If present, the import lists of an implementation module shall be valid given the outermost environment — see 6.1.8. If present, the import lists and the declarations of an implementation module block shall be consistent with one another — see 6.1.8.5.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NOTE — An implementation module and it corresponding definition module must also be consistent (for example the exported procedures must be declared) — see 6.1.1 and 6.1.5.1.

Given the identifiers of the system modules and the separate modules that are imported into an implementation module, given the identifiers that are imported from those modules, and given the identifiers introduced by the definitions of the corresponding definition module:

— the declarations of the module block of the implementation module shall be well-formed — see 6.2.6,

— the bodies of module and procedure declarations of the module block shall be well-formed — see 6.2.7,

— if present, the body of the module block shall be a well-formed module body — see 6.5.3.

If present in an implementation module heading, the interrupt protection shall be a valid interrupt protection in the context of the pervasive identifier environment only — see 6.1.11.

wf-sourced-implementation-module : *Sourced-implementation-module* → *Environment* → $\mathbb{B}$

wf-sourced-implementation-module (*mk-Sourced-implementation-module* (*name*, *imports*, *block*, *protection*))ρ_{out} ≙
  name ∉ elems *directly-imports* (*imports*) ∧
  wf-import-lists (*imports*)ρ_{out} ∧ 42
  let *mk-Module-block* (*decls*, -, -) = *block* in
  consistent-imports (*imports*, *decls*)ρ_{out} ∧ 46
  let ρ_{def} = *associated-local-environment* (*name*)ρ_{out} in 317
  let ρ_{import} = *d-import-lists* (*imports*)ρ_{out} in 42
  let ρ_{block} = *merge-environments* ({ρ_{import}, ρ_{def}}) in 319
  (*wf-declarations* (*decls*)ρ_{block} ∧ 65
  wf-declaration-bodies (*decls*)ρ_{block} ∧ 67
  wf-module-body (*block*)ρ_{block}) ∧ 132
  (*protection* = nil ∨ *wf-interrupt-protection* (*protection*)ρ_{empty}) 51

Dynamic Semantics

The elaboration of an implementation module shall elaborate the declarations of the module block, given the meaning of the imported identifiers and the meaning of the identifiers that are introduced by the definitions of the definition module.

If an implmentation module is a protected module, the protection specified for the module shall be used to protect the protection domain of the implementation module. The protection shall be evaluated in the context of the pervasive identifier environment only.

m-sourced-implementation-module : *Sourced-implementation-module* → *Environment* $\xrightarrow{o}$ *Environment*

m-sourced-implementation-module (*mk-Sourced-implementation-module* (*name*, *imports*, *block*, *protection*))ρ_{out} ≙
  let ρ_{def} = *associated-local-environment* (*name*)ρ_{out} in 317
  let ρ_{import} = *d-import-lists* (*imports*)ρ_{out} in 42
  let ρ_{com} = *combine-definition-and-imported-environments* (ρ_{def}, ρ_{import}) in 320
  let ρ_{mod} = if *protection* ≠ nil
    then *mark-exported-from-protected* (*visible-identifiers* (ρ_{def}))ρ_{com} 318 319
    else *mark-exported-from-protected* ({ })ρ_{com} in 318
  def *pval* = if *protection* = nil then nil else *evaluate-interrupt-protection*(*protection*) ρ_{empty} in 51
  let $\rho_{mod'}$ = *add-protection* (*pval*)ρ_{mod} in 318
  def ρ_{block} = *m-module-block*(*block*) $\rho_{mod'}$ in 132
  return *overwrite-mod-environment* ({*name* ↦ ρ_{block}})ρ_{out} 317

annotations The compilation order is such that, when this operation is called, the result of elaboration of all definition modules used by the implementation module have been merged in the given environment. The protection expression (if present) is evaluated in an environment consisting of the pervasive identifiers only, and is added to the returned environment for use by the protection procedures.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The implementation module shall be initialized by initializing the implementation module block given the environment resulting from the elaboration of the implementation module — see 6.5.4.

initialize-a-sourced-implementation-module : *Sourced-implementation-module* → *Environment* $\xrightarrow{o}$ ()

initialize-a-sourced-implementation-module (*mk-Sourced-implementation-module* (*name*, -, *block*, -))ρ $\triangleq$
  let ρ_{block} = *associated-module* (*name*)ρ in 317
  execute-static-module-initialization(*block*) ρ_{block} 132

annotations The environment for initializing the module block of the implementation module is retrieved from the given environment.

6.1.4.2 Abstract Implementation Modules

Abstract implementation modules are derived from the formal specification of a separate module and not from Modula-2 source text. The method of producing an abstract implementation module is given in normative annex B.

Abstract Syntax

types

Abstract-implementation-module :: *name* : *Identifier*
  init : [*Intialization-actions*]
  procs : *Procedures*
  dens : *Procdens*
  final : [*Termination-actions*]

inv *mk-Abstract-implementation-module* (-, -, *procs*, *dens*, -) $\triangleq$
  rng *procs* $\subseteq$ dom *dens*

Static Semantics

wf-abstract-implementation-module : *Abstract-implementation-module* → *Environment* → $\mathbb{B}$

wf-abstract-implementation-module (*aim*)ρ $\triangleq$
  true

Dynamic Semantics

m-abstract-implementation-module : *Abstract-implementation-module* → *Environment* $\xrightarrow{o}$ *Environment*

m-abstract-implementation-module (*mk-Abstract-implementation-module* (*name*, -, *pids*, *pdens*, -))ρ_{out} $\triangleq$
  (*store-procedure-dens*(*dens*) ; 340
  let ρ_{def} = *associated-module* (*name*)ρ_{out} in 317
  let $\rho_{abstract} = \mu\,(\rho_{empty}, procs \mapsto pids,)$ in
  let ρ_{block} = *merge-environment* ($\rho_{def}, \rho_{abstract}$) in 319
  return *overwrite-mod-environment* ({*name* $\mapsto$ ρ_{block}})ρ_{out}) 317

initialize-an-abstract-implementation-module : *Sourced-implementation-module* → *Environment* $\xrightarrow{o}$ ()

initialize-an-abstract-implementation-module (*mk-Abstract-implementation-module* (-, *init*, -, -, *final*))ρ $\triangleq$
  (if *final* $\neq$ nil
  then *queue-termination*(*mk-Abstract-module-finalization* (*final*, ρ)) 341
  else skip;
  let *handler* = *current-exception-handler* (ρ) in 318
  init() *handler*) 33

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.1.5 Program Module and Separate Module Consistency

Consistency between the compilation modules of a program is achieved if there is consistency between the correspondingly named definition and implementation modules, and if there is at least one valid compilation order.

6.1.5.1 Definition Module and Implementation Module Consistency

The definition module and implementation module of a separate module are consistent if their definitions and declarations are consistent.

Static Semantics

The identifiers imported into the implementation module by the import lists of the implementation module shall be distinct from the the identifiers defined in the definitions of the corresponding definition module.

Except for opaque types and procedures, no identifier defined in the definition module of a separate module shall be declared in the corresponding implementation module.

Each opaque type defined in the definition module of a separate module shall be declared in its corresponding implementation module as a pointer type or as identical to another opaque type.

NOTES

1 The defining instance of the type corresponding to an opaque type identifier defined in a definition module need not be within the declarations of the corresponding implementation module — the opaque type identifier may be exported from a local module of the implementation module or the opaque type may be declared as identical to that denoted by an imported identifier.

2 An opaque type may be redeclared as the address type, since the address type is also a pointer type.

CHANGE — In the third edition of *Programming in Modula-2*, opaque types must be declared as pointer types. This part of ISO/IEC 10514 allows an opaque type to be declared as another opaque type, since that other opaque type must directly or indirectly be implemented as a pointer type.

CLARIFICATION 1 — An opaque type cannot be declared as a scalar type.

Each procedure heading in the definition module of a separate module shall have a matching procedure declaration in the corresponding implementation module.

NOTE 3 — A procedure declaration matching a procedure heading need not be within the declarations of the corresponding implementation module — the procedure identifier may be exported from a local module of the implementation module.

The formal parameter list of the procedure heading in the definition module shall match the formal parameter list of the procedure heading of the procedure declaration in the implementation module. The formal parameter lists of procedure headings shall match if the formal types of each of the value parameter specifications or variable parameter specifications of corresponding formal parameters are identical.

CLARIFICATION 2 — The definition of a procedure in a definition module and the corresponding declaration of that procedure in the implementation module must match in the sense that corresponding parameters must have the identical type. The identifiers used to denote the parameters need not be the same, and the identifiers used to denote the types of these parameters need not be the same.

NOTE 4 — Even if a definition module does not contain any procedure definitions or opaque type definitions there must still be a corresponding implementation module. Likewise, even if there are no exports from a module and the implementation module is used solely for initialization, there must still be a corresponding definition module.

$wf\text{-}module\text{-}pair : Definition\text{-}module \times Implementation\text{-}module \rightarrow Environment \rightarrow \mathbb{B}$

$$wf\text{-}module\text{-}pair\ (dmod, imod)\rho \triangleq$$
$$(is\text{-}Sourced\text{-}implementation\text{-}module(imod) \rightarrow wf\text{-}sourced\text{-}module\text{-}pair\ (dmod, imod)\rho, \quad 35$$
$$is\text{-}Abstract\text{-}implementation\text{-}module(imod) \rightarrow wf\text{-}abstract\text{-}module\text{-}pair\ (dmod, imod)\rho) \quad 36$$

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wf-sourced-module-pair : *Definition-module* × *Sourced-Implementation-module* → *Environment* → 𝔹

wf-sourced-module-pair (*dmod*, *imod*)ρ ≙
  let ρ_{def} = *definition-environment* (*dmod*)ρ in 35
  let ρ_{res} = *residual-environment* (*dmod*)ρ_{def} in 35
  let ρ_{imp} = *imported-environment* (*imod*)ρ in 35
  let ρ_{mod} = *module-environment* (*imod*)ρ in 35
  no-imports-from-definition-module (ρ_{def}, ρ_{imp}) ∧ 35
  no-illegal-redeclarations (ρ_{res}, ρ_{mod}) ∧ 35
  opaque-types-declared (ρ_{def}, ρ_{mod}) ∧ 36
  procedures-declared (ρ_{def}, ρ_{mod}) 36

definition-environment : *Definition-module* → *Environment* → *Environment*

definition-environment (*dmod*)ρ ≙
  let *mk-Definition-module* (*name*, -, -) = *dmod* in
  associated-module (*name*)ρ 317

annotations Retrieve an environment for a definition module.

residual-environment : *Definition-module* → *Environment* → *Environment*

residual-environment (*mk-Definition-module* (*name*, -, -))ρ ≙
  let *dprocs* = *procedures-of* (ρ) in 316
  let *dopaques* = *opaque-types-of* (ρ) in 328
  restrict-environment (dom (*dprocs* ∪ *dopaques*))ρ 320

annotations Remove any components of the given environment that are derived from declarations of procedure headings or opaque types.

imported-environment : *Sourced-implementation-module* → *Environment* → *Environment*

imported-environment (*imod*)ρ ≙
  let *mk-Sourced-implementation-module* (-, *imports*, -, -) = *imod* in
  d-import-lists (*imports*)ρ 42

annotations Construct an environment for any imported entities.

module-environment : *Sourced-implementation-module* → *Environment* → *Environment*

module-environment (*imod*)ρ ≙
  let *mk-Sourced-implementation-module* (*name*, *imports*, *block*, -) = *imod* in
  let ρ_{def} = *associated-module* (*name*)ρ in 317
  let ρ_{import} = *d-import-lists* (*imports*)ρ in 42
  let ρ_{module} = *merge-environments* ({ρ_{import}, ρ_{def}}) in 319
  let *mk-Module-block* (*decls*, -) = *block* in
  d-declarations (*decls*)ρ_{module} 64

annotations Construct an environment for an implementation module.

no-imports-from-definition-module : *Environment* × *Environment* → 𝔹

no-imports-from-definition-module (ρ_{def}, ρ_{imp}) ≙
  let *defed* = *visible-identifiers* (ρ_{def}) in 319
  let *imported* = *visible-identifiers* (ρ_{imp}) in 319
  imported ∩ *defed* = { }

annotations Construct a set of identifiers that contains the identifiers defined in the definitions component of a definition module and construct a set containing the identifiers contained in the import lists component of an implementation module. The result is *true* if and only if the intersection of these two sets is empty.

no-illegal-redeclarations : *Environment* × *Environment* → 𝔹

no-illegal-redeclarations (ρ_{res}, ρ_{mod}) ≙
  visible-identifiers (ρ_{res}) ∩ *visible-identifiers* (ρ_{mod}) = { } 319 319

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

opaque-types-declared : *Environment* × *Environment* → $\mathbb{B}$

opaque-types-declared $(\rho_{def}, \rho_{mod}) \triangleq$
  let *dopaques* = *opaque-types-of* (ρ_{def}) in 328
  $\forall\, id \in dopaques \cdot$ *is-pointer-type* $(id)\rho_{mod} \vee$ *is-opaque-type* $(id)\rho_{mod}$ 322 322

annotations Check that each of the opaque types defined in a definition module is declared as a pointer type (which may be the address type) or as an opaque type in the corresponding implementation module.

procedures-declared : *Environment* × *Environment* → $\mathbb{B}$

procedures-declared $(\rho_1, \rho_2) \triangleq$
  let *procs*1 = *procedures-of* (ρ_1) in 316
  let *procs*2 = *procedures-of* (ρ_2) in 316
  $\forall\, id \in$ dom *procs*1 $\cdot$ *procs*1(id) = *procs*2(id)

annotations Extract the procedure headings from the two environments and check that they are identical. This function is used to check definition modules against implementation modules and forward procedure declarations against full procedure declarations.

wf-abstract-module-pair : *Definition-module* × *Abstract-Implementation-module* → *Environment* → $\mathbb{B}$

wf-abstract-module-pair $(dmod, imod)\rho \triangleq$
  true

annotations An abstract implementation module is consistent with its corresponding definition module by virtue of the method of its construction from the specification of an abstract implementation module.

6.1.5.2 Module Compilation Order

A compilation module may only be elaborated if the definition modules that define the entities imported by that compilation module have been elaborated. This restriction may always be satisfied provided no definition module indirectly imports itself. (A module S directly imports a module T if T occurs in the import list of S. A module S indirectly imports another module U if module S directly imports U, or if module S directly imports a module T and T indirectly imports module U.)

Static Semantics

A valid compilation order shall be such that

— all compilation modules of the program are included exactly once,

— a definition module is included before the corresponding implementation module,

— a definition module is included before any other compilation module that imports from the corresponding separate module.

types

Module-order = *Compilation-unit**;

Compilation-unit = *Program-module*
  | *Definition-module*
  | *Implementation-module*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

compilation-orders : *Program-module* × *Definition-modules* × *Implementation-modules* → *Module-order*-set

compilation-orders (*pmod*, *dmods*, *imods*) ≙
  {*order* | *order* : *Module-order* ·
    is-complete-order (*order*, *pmod*, *dmods*, *imods*) ∧ 37
    is-correct-order (*order*, *pmod*, *dmods*, *imods*)} 37

Auxiliaries

is-complete-order : *Module-order* × *Program-module* × *Definition-modules* × *Implementation-modules* → 𝔹

is-complete-order (*order*, *pmod*, *dmods*, *imods*) ≙
  rng *order* = {*pmod*} ∪ *dmods* ∪ *imods* ∧
  ∀ *i*, *j* ∈ inds *order* · *i* = *j* ∨ *order*(*i*) ≠ *order*(*j*)

is-correct-order : *Module-order* × *Program-module* × *Definition-modules* × *Implementation-modules* → 𝔹

is-correct-order (*order*, *pmod*, *dmods*, *imods*) ≙
  ∀ *i* ∈ inds *order* ·
    let *cdefs* = *compiled-definition-modules* (*order* (1, . . . , *i* − 1)) ∪ *SYSTEM-MODULE-NAMES* in 37
    cases *order*(*i*) :
      mk-Program-module (-, *imports*, -, -) →
        let *names* = *directly-imports* (*imports*) in 42
        elems *names* ⊆ *cdefs*,
      mk-Definition-module (-, *imports*, -) →
        let *names* = *directly-imports* (*imports*) in 42
        elems *names* ⊆ *cdefs*,
      mk-Sourced-implementation-module (*name*, *imports*, -, -) →
        name ∈ *cdefs* ∧
        let *names* = *directly-imports* (*imports*) in 42
        elems *names* ⊆ *cdefs*,
      mk-Abstract-implementation-module (*name*, -, -, -, -) →
        name ∈ *cdefs*
    end

compiled-definition-modules : *Module-order* → *Identifier*-set

compiled-definition-modules (*order*) ≙
  {*id* | *mk-Definition-module* (*id*, -, -) ∈ elems *order*}

annotations Construct the set of the identifiers of the definition modules that appear in a sequence of program, definition, and implementation modules.

6.1.5.3 Module Dependencies

The compilation modules that depend upon a definition module of a program are those compilation modules that must be elaborated after the definition module is itself elaborated in a valid compilation order — see 6.1.5.2.

If a definition module of a program is modified, any record of a previous elaboration (or partial elaboration) of that definition module, and of a previous elaboration (or partial elaboration) of a dependent module, is potentially inconsistent with the modification. In this case, all affected modules must be elaborated again, in the same order as their elaboration in a valid compilation order.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

$dependent\text{-}modules : Compilation\text{-}module \times Program\text{-}module \times Definition\text{-}modules \times Implementation\text{-}modules \rightarrow Module\text{-}order\text{-}\mathsf{set}$

$dependent\text{-}modules\,(module, pmod, dmods, imods) \triangleq$
  let $orders = compilation\text{-}orders\,(pmod, dmods, imods)$ in 36
  let $depend = \{x \mid x : Module\text{-}order \cdot \exists\, order \in orders \cdot is\text{-}suffix\,([module] \frown x, order)\}$ in 614
  let $size = mins\,(\{\mathsf{len}\ x \mid x \in depend\})$ in 614
  $\{dep \mid dep \in depend \cdot \mathsf{len}\ dep = size\}$

annotations The result of this operation is a set of compilation orders for the compilation modules that must be recompiled if the compilation module that is the first parameter is recompiled.

6.1.6 Module Initialization Order

The order in which the implementation modules of the program are initialized is determined by the order in which the corresponding separate module identifiers appear in the import lists of the program module and the import lists of the other compilation modules — see 6.1.8.

The order is such that, unless a separate module indirectly imports itself, an implementation module is initialized before the initialization of any client modules.

A separate module C is a 'client' of a separate module M if the definition module of C, or the implementation module of C, imports from the separate module M and thus is a compilation module that depends upon the definition module of M.

A program module is a client of a separate module M if it imports from the separate module M and thus is a compilation module that depends upon the definition module of M.

A separate module indirectly imports from itself if it is part of a circle of references in the import lists. For example, if an implementation module A imports from a separate module B whose definition or implementation module imports from separate module A, then modules A and B both indirectly import from themselves. The relative order of initialization of such modules is related to the order of their occurrence in the import lists of the program.

Dynamic Semantics

The initialization order shall be determined by first processing the import lists of the program module. The import lists of the program module shall be processed by processing the import lists of each separate module whose identifier appears in the import lists of the program module, in order of occurrence in those import lists.

If an import list is an unqualified import, the import list shall be processed by processing the import lists of the separate module whose module identifier appears in the import list. If an import list is a simple import, the import list shall be processed by processing in turn the import lists of each separate module whose identifier appears in the import list identifiers.

If the import lists of a separate module have finished being processed, or have started to be processed, the import lists of that module shall not be processed again. Otherwise, the import lists of the definition module shall be processed, and then the import lists of the implementation module shall be processed.

The initialization order shall be the order in which the corresponding separate modules finish the processing of their import lists.

CHANGE — The second edition of *Programming in Modula-2* does not deal with the effects of import lists on the order of the initialization of modules. The third edition states how import lists appearing in the implementation modules are used in the definition of the initialization order. The rule for import lists in implementation modules has been extended to cover the program module and definition modules.

This part of ISO/IEC 10514 defines an order for the initialization of modules when there is a circular reference in the import lists. The third edition of *Programming in Modula-2* states that the order of initialization is undefined in such a case.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

types

Execution-order = *Implementation-module**

execution-order :
Program-module × *Definition-modules* × *Implementation-modules* → *Execution-order*

execution-order (*pmod*, *dmods*, *imods*) ≜
let *mk-Program-module* (-, *imports*, -, -) = *pmod* in
let *names* = *directly-imports* (*imports*) in 42
import-orders (*names*, *dmods*, *imods*, *SYSTEM-MODULE-NAMES*) 39

import-orders :
Identifier* × *Definition-modules* × *Implementation-modules* × *Identifier*-set → *Execution-order*

import-orders (*names*, *dmods*, *imods*, *done*) ≜
if *names* = []
then []
else let *first* = *import-order* (hd *names*, *dmods*, *imods*, *done*) in 39
let *rest* = *import-orders* (tl *names*, *dmods*, *imods*, *done* ∪ elems *first*) in 39
.*first* ↷ *rest*

import-order :
Identifier × *Definition-modules* × *Implementation-modules* × *Identifier*-set → *Execution-order*

import-order (*name*, *dmods*, *imods*, *done*) ≜
if *name* ∈ *done*
then []
else let *imported* = *imported-by-definition-module* (*name*, *dmods*) ↷ 39
imported-by-implementation-module (*name*, *imods*) in 39
let *imod* ∈ *imods* be st *imod*.*name* = *name* in
import-orders (*imported*, *dmods*, *imods*, *done* ∪ {*name*}) ↷ *imod* 39

Auxiliaries

imported-by-definition-module : *Identifier* × *Definition-modules* → *Identifier**

imported-by-definition-module (*nm*, *dmods*) ≜
let *mk-Definition-module* (*name*, *imports*, -) ∈ *dmods* be st *name* = *nm* in
directly-imports (*imports*) 42

annotations Construct a sequence that contains the identifiers of those modules that are directly imported by the definition module.

imported-by-implementation-module : *Identifier* × *Implementation-module* → *Identifier**

imported-by-implementation-module (*nm*, *imod*) ≜
(*is-Sourced-implementation-module*(*imod*) → *imported-by-sourced-implementation-module* (*nm*, *imod*), 39
is-Abstract-implementation-module(*imod*) → [])

imported-by-sourced-implementation-module : *Identifier* × *Sourced-implementation-modules* → *Identifier**

imported-by-sourced-implementation-module (*nm*, *imods*) ≜
let *mk-Sourced-implementation-module* (*name*, *imports*, -, -) ∈ *mod* be st *name* = *nm* in
directly-imports (*imports*) 42

annotations Construct a sequence of identifiers that contains the identifiers of those modules that are directly imported by the implementation module.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.1.7 Program Termination

During program termination, the finalization bodies of those static modules that have started initialization are executed in the reverse order of their initialization. The static modules are the program module, the implementation modules, and any local modules declared in the module blocks of these compilation modules.

Program termination starts on the first occurrence of one of the following termination events:

a) the end of the normal part of the program module initialization body is reached,

b) a `RETURN` statement is executed in the program module initialization body,

c) the standard procedure `HALT` is called,

d) an exception is raised and this exception is not handled.

Cases a) and b) constitute normal termination. Case c) does not change the execution state, but no exit is made from protection domains — see 6.1.11 and 6.1.1. Case d) constitutes exceptional termination. (During exceptional termination an implementation, where appropriate, is required to propagate a report of the exception to the environment from which the program was invoked — see 4.7.)

In a program with coroutines, cases a) and b) can only occur during execution of the main coroutine. It is possible for cases c) and d) to occur during the execution of any coroutine.

Further termination events might occur during finalization of the static modules. Firstly, the finalization body of a static module block may itself call `HALT` or may raise an exception that is not handled. Secondly, since coroutine activity continues during the finalization of the static modules, there might be a further termination event during execution of a coroutine.

In order to guarantee that the finalization bodies of static modules are always executed in the reverse order of their initialization, and that all static modules that have started their initialization are finalized, the first occurrence of a termination event in a coroutine causes static module finalization to be executed by that coroutine. This is true whether or not a termination event has already occurred during execution of another coroutine.

On the second and subsequent occurrences of a termination event in a coroutine, during static module finalization by that coroutine, the execution of the finalization body ceases and execution in that coroutine continues with the next finalization body that is to be executed by that coroutine.

As soon as the execution of the last static module to be finalized is completed, or is ceased, execution of the entire program is completed.

Since a coroutine may be started during the initialization of static modules, a coroutine may start finalizing before further static modules have started their initialization. In that case, no further static modules are initialized — see 6.5.4.

The system module `TERMINATION` exports enquiry procedures that allow the program to determine if program termination is in progress and if there has been a call of `HALT` — see 7.4.

Dynamic Semantics

On the first occurrence of a termination event during the execution of a coroutine, if there is at least one static module of the program that has a finalization body and that has completed or started its initialization, static module finalization shall be started by that coroutine. The finalization bodies shall be executed for all static modules that have finalization bodies, and that have completed or started initialization. The finalization order shall be the reverse of the order in which these modules were initialized.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

If there is a subsequent occurrence of a termination event, while the coroutine is executing the finalization body of a static module, the coroutine shall execute the next finalization body in the finalization order.

If there are no static modules to be finalized, or no further static modules to be finalized, the execution of the program shall be complete.

NOTES

1 In a program with coroutines other than the main coroutine, the finalization body of a static module should be written so that it will behave correctly if called more than once or if reentered.

2 Modules that are dynamically initialized (on calls to procedures in which they are declared as local modules) are not finalized on program termination.

CHANGE — *Programming in Modula-2* makes no provision for the finalization of static modules on program termination.

```
termination : () →o ()
termination () ≙
  (def terminating = is-terminating() in                    340
   if terminating
   then skip
   else set-terminating() ;                                 340
   finalize-modules() )                                     41
```

annotations The formal model keeps a stack of static modules to be finalized in the global state. This stack is built during static module initialization. The operations *is-terminating* and *set-terminating* are per-coroutine operations. *set-terminating* makes a copy of the stack in the per-coroutine state.

```
finalize-modules : () →o ()
finalize-modules () ≙
  def module = next-module() in                             341
  if module = nil
  then skip
  else (finalize-a-static-module(module) ;                  135
        finalize-modules() )                                41
```

annotations The operation *next-module* is a per-coroutine operation. It pops the next module off the per-coroutine copy of the termination stack.

6.1.8 Import Lists

Following the module heading, a module may have a sequence of import lists. An import list includes a list of the identifiers that are to be explicitly imported into the module. Explicit import of an enumeration type identifier implicitly imports the enumeration constant identifiers of the enumeration type.

Imported identifiers are introduced into the module, thus extending their scope, but they have a defining occurrence that appears elsewhere.

Every kind of module may include a sequence of import lists, whether it is a program module, a definition module, an implementation module, or a local module. In the case of a definition module, the imported identifiers may be used in the definitions of that module. In the case of any other kind of module, the imported identifiers may be used in the block of the module.

Concrete Syntax

```
import lists = { import list } ;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Abstract Syntax

types

Import-lists = *Import-list**

Declaration Semantics

The identifiers imported by a sequence of import lists of a module shall be all of the identifiers imported separately by the individual import lists. The import of an identifier shall also import the entity denoted by that identifier.

d-import-lists : *Import-lists* → *Environment* → *Environment*

d-import-lists (*imports*)ρ ≜
  merge-environments ({*d-import-list* (*import*)ρ | *import* ∈ elems *imports*}) 319 43

Static Semantics

Each import list in a sequence of import lists shall be valid in the given environment.

NOTE — In the case of an import list appearing in a local module, the given environment is the environment that applies at the point of the local module declaration — see 6.2.12. In the case of an import list appearing in a program module, definition module, or implementation module, the given environment is the outermost environment which includes the identifiers of the system modules and the separate modules of the program — see 6.1.1.

An identifier shall not be imported (explicitly or implicitly) by more than one import list in the import lists of a module.

wf-import-lists : *Import-lists* → *Environment* → $\mathbb{B}$

wf-import-lists (*imports*)ρ ≜
  ∀ *import* ∈ elems *imports* · *wf-import-list* (*import*)ρ ∧ 43
  ∀ i, j ∈ inds *imports* · $i = j$ ∨ (*imported-names* (*import*(i))ρ) ∩ *imported-names* (*import*(j))ρ) = { }) 43 43

Auxiliaries

directly-imports : *Import-lists* → *Identifier**

directly-imports (*imports*) ≜
  [*imported-module-name* (*imports*(i)) | i ∈ inds *imports*] 42

annotations Construct a sequence of identifiers containing the identifiers of all the modules whose identifiers occur in the import lists of a module.

imported-module-name : *Import-list* → *Identifier**

imported-module-name (*import*) ≜
  cases *import* :
    mk-Simple-import (*ids*) → *ids*,
    mk-Unqualified-import (*from*, -) → [*from*]
  end

annotations Construct a sequence of identifiers containing the identifiers of all the modules that occur in an import list of a module.

imports-of : *Import-lists* → *Environment* → *Identifier*-set

imports-of (*imports*)ρ ≜
  ⋃{*imported-names* (*import*)ρ | *import* ∈ elems *imports*} 43

annotations Construct a set of identifiers from the import lists of a module.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

imported-names : *Import-list* → *Environment* → *Identifier*-set

imported-names (*import*)ρ ≜
  cases *imports* :
    mk-Simple-import (*ids*) → $\bigcup\{x\text{-}closure\,(id)\rho \mid id \in \text{elems } ids\}$, 49
    mk-Unqualified-import (*from*, *ids*) → let ρ_{mod} = *associated-module* (*from*)ρ in 317
      $\bigcup\{x\text{-}closure\,(id)\rho_{mod} \mid id \in \text{elems } ids\}$ 49
  end

annotations Construct a set of identifiers from the import list of a module. The identifiers in the set represent entities that are imported into a module. Note that the identifier in the FROM component (i.e. the name of the module) of an unqualified import is not in the set.

6.1.8.1 Import List

An import list may either be a simple import or an unqualified import.

Concrete Syntax

import list = simple import | unqualified import ;

Abstract Syntax

types

Import-list = *Simple-import* | *Unqualified-import*

Declaration Semantics

d-import-list : *Import-list* → *Environment* → *Environment*

d-import-list (*import*)ρ ≜
  (*is-Simple-import*(*import*) → *d-simple-import* (*import*)ρ, 44
   is-Unqualified-import(*import*) → *d-unqualified-import* (*import*)ρ) 45

Static Semantics

wf-import-list : *Import-lists* → *Environment* → $\mathbb{B}$

wf-import-list (*import*)ρ ≜
  (*is-Simple-import*(*import*) → *wf-simple-import* (*import*)ρ, 44
   is-Unqualified-import(*import*) → *wf-unqualified-import* (*import*)ρ) 45

6.1.8.2 Simple Imports

A simple import includes a list of the identifiers that are to be explicitly imported. Explicit import of an enumeration type identifier implicitly imports the enumeration constant identifiers of the enumeration type.

The identifiers imported by a simple import are introduced into the module in which the simple import appears. In the case of a simple import appearing in a local module, the scope of the imported identifiers is extended from the block in which the local module is declared. In the case of a simple import in a program module, definition module, or implementation module, the scope is extended from the outermost environment which includes the identifiers of the system modules and the separate modules of the program.

Concrete Syntax

simple import = "IMPORT", identifier list, semicolon ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Abstract Syntax

types

Simple-import :: *ids* : *Identifier**

Declaration Semantics

The identifiers that are imported by a simple import shall be the identifiers in the identifier list of the simple import, together with any identifiers that are implicitly imported with those identifiers — see 6.1.10.

d-simple-import : *Simple-import* → *Environment* → *Environment*

d-simple-import (*mk-Simple-import* (*ids*))ρ ≙
  let *closure* = $\bigcup \{$*x-closure* (*id*)ρ | *id* ∈ elems *ids*$\}$ in [49]
  restrict-environment (*closure*)ρ [320]

Static Semantics

The identifiers imported (explicitly or implicitly) by a simple import shall be distinct from each other. Each of the identifiers in the identifier list shall be visible in the given environment.

NOTE — The given environment does not include the pervasive identifiers, and so the pervasive identifiers may not be included in a simple import.

wf-simple-import : *Simple-import* → *Environment* → $\mathbb{B}$

wf-simple-import (*mk-Simple-import* (*ids*))ρ ≙
  ($\forall i, j \in$ inds *ids* · $i = j \vee$ *x-closure* (*ids*(i)) ∩ *x-closure* (*ids*(j)) = { }) ∧ [49] [49]
  elems *ids* ⊆ *visible-identifiers* (ρ) [319]

6.1.8.3 Unqualified Import

An unqualified import specifies a module identifier and a list of the identifiers that are to be explicitly imported from that module. Explicit import of an enumeration type identifier implicitly imports the enumeration constant identifiers of the enumeration type.

The identifiers imported by an unqualified import are introduced into the module in which the unqualified import appears. The scope of the imported identifiers is extended from the module specified by the unqualified import.

In the case of an unqualified import appearing in a local module, the module identifier refers to the module it denotes in the block in which the local module is declared. In the case of an unqualified import in a program module, definition module, or implementation module, the module identifier refers to one of the system modules or one of the separate modules of the program.

Concrete Syntax

unqualified import = "FROM", module identifier, "IMPORT", identifier list, semicolon ;

NOTE — The identifier list of the unqualified import contains identifiers that are distinct from each other.

Abstract Syntax

types

Unqualified-import :: *from* : *Identifier*
  ids : *Identifier*-set

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

annotations The number of elements in the set of identifiers of the abstract syntax of an unqualified import is equal to the number of identifiers in the identifier list of the concrete syntax.

Declaration Semantics

The identifiers that are imported by an unqualified import shall be the identifiers in the identifier list of the unqualified import that are exported by the module denoted by the module identifier, together with any identifiers that are implicitly imported with those identifiers — see 6.1.10.

d-unqualified-import : *Unqualified-import* → *Environment* → *Environment*

d-unqualified-import (*mk-Unqualified-import* (*from*, *ids*))ρ ≜
  let ρ_{mod} = *associated-module* (*from*)ρ in 317
  let *closure* = $\bigcup \{$*x-closure* (*id*)ρ_{mod} | *id* ∈ *ids*$\}$ in 49
  restrict-environment (*closure*)ρ_{mod} 320

Static Semantics

The identifiers imported by an unqualified import shall be distinct from each other. The module identifier shall be the identifier of a module that is visible in the given environment, and each of the identifiers in the identifier list shall be exported from the identified module.

wf-unqualified-import : *Unqualified-import* → *Environment* → $\mathbb{B}$

wf-unqualified-import (*mk-Unqualified-import* (*from*, *ids*))ρ ≜
  from ∈ *visible-identifiers* (ρ) ∧ 319
  is-module (*from*)ρ ∧ 317
  is-exported (*from*, *ids*)ρ ∧ 45
  ($\forall$ *i*, *j* ∈ *ids* · *i* = *j* ∨ *x-closure* (*i*) ∩ *x-closure* (*j*) = { }) 49 49

Auxiliaries

is-exported : *Identifier* × *Identifier*-set → *Environment* → $\mathbb{B}$

is-exported (*from*, *ids*)ρ ≜
  let ρ_{def} = *associated-module* (*from*)ρ in 317
  ids ⊆ *visible-identifiers* (ρ_{def}) 319

annotations The identifiers that are imported from a module must be exported by that module; thus they must be in the domain of the environment associated with the module from which they were exported.

6.1.8.4 Import Consistency in Definition Modules

Static Semantics

In the case of a definition module containing an import list, none of the implicitly or explicitly imported identifiers may be the same as an identifier defined in the definition module.

consistent-imports-in-definitions : *Import-lists* × *Definitions* → *Environment* → $\mathbb{B}$

consistent-imports-in-definitions (*imports*, *defs*)ρ ≜
  let *defined* = *identifiers-defined-in* (*defs*) in 82
  let *imported* = *imports-of* (*imports*)ρ in 42
  imported ∩ *defined* = { }

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.1.8.5 Import Consistency in Module Blocks

Static Semantics

In the case of a module other than a definition module containing an import list, none of the explicitly or implicitly imported identifiers may be the same as an identifier declared in the module block of the importing module.

consistent-imports : *Import-lists* × *Declarations* → *Environment* → $\mathbb{B}$

consistent-imports (*imports*, *decls*)ρ $\triangleq$
  let *decled* = *identifiers-declared-in* (*decls*) in 82
  let *imported* = *imports-of* (*imports*)ρ in 42
  imported $\cap$ *decled* = { }

6.1.9 Export Lists

Following the import lists, if any, a local module may have a single export list — see 6.2.12. An export list includes a list of the identifiers that are explicitly exported from a local module. Explicit export of an enumeration type identifier implicitly exports the enumeration constant identifiers of the enumeration type.

An export list may either be an unqualified export or a qualified export.

Concrete Syntax

export list = unqualified export | qualified export ;

Abstract Syntax

types

Export-list = *Unqualified-export* | *Qualified-export*

Declaration Semantics

The export of an identifier shall also export the entity denoted by that identifier.

d-export-list : *Export-list* → *Environment* → *Environment*

d-export-list (*export*)ρ $\triangleq$
  (*is-Unqualified-export*(*export*) → *d-unqualified-export* (*export*)ρ, 47
   is-Qualified-export(*export*) → *d-qualified-export* (*export*)ρ) 48

Static Semantics

wf-export-list : *Export-list* → *Environment* → $\mathbb{B}$

wf-export-list (*export*)ρ $\triangleq$
  (*is-Unqualified-export*(*export*) → *wf-unqualified-export* (*export*)ρ, 47
   is-Qualified-export(*export*) → *wf-qualified-export* (*export*)ρ) 48

6.1.9.1 Unqualified Exports

The identifiers exported by an unqualified export of a local module are introduced into the procedure or module in which the local module declaration appears. In the block of that procedure or module, the exported identifiers can be used with or without being qualified by the identifier of the local module.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Concrete Syntax

unqualified export = "EXPORT", identifier list, semicolon ;

NOTE 1 — The identifier list of the unqualified export contains identifiers that are distinct from each other.

Abstract Syntax

types

Unqualified-export :: *ids* : *Identifier*-set

annotations The number of elements in the set of identifiers of the abstract syntax of an unqualified export is equal to the number of identifiers in the identifier list of the concrete syntax.

Declaration Semantics

The identifiers that are exported by an unqualified export shall be the identifiers in the identifier list of that unqualified export, together with any identifiers that are implicitly exported with those identifiers — see 6.1.10.

d-unqualified-export : *Unqualified-export* → *Environment* → *Unqualified-environment*

d-unqualified-export (*mk-Unqualified-export* (*ids*))ρ $\triangleq$
  let *closure* = $\bigcup \{x\text{-}closure\,(id)\rho \mid id \in ids\}$ in 49
  restrict-environment (*closure*)ρ 320

Static Semantics

The identifiers in the identifier list of an unqualified export shall be distinct from each other.

The identifiers in the identifier list of an unqualified export shall be declared in the declarations of the local module that contains the unqualified export.

NOTES

2 The identifiers in the identifier list of an unqualified export are explicitly exported. Identifiers implicitly exported from a local module need not be declared in declarations of the local module since export of an enumeration type identifier implicitly exports the enumeration constant identifiers of the enumeration type whether or not the type identifier and the enumeration constant identifiers are introduced by the same declaration — see 6.1.10.

3 The identifiers in the identifier list of an unqualified export may not be otherwise introduced into the procedure block or module block in which the local module is declared. This follows from the rules that declarations of a block may include only one defining occurrence of a particular identifier and, in the case of a module block, that none of the imported identifiers may be the same as an identifier declared in that block — see 6.2.6 and 6.1.8.5.

wf-unqualified-export : *Unqualified-export* → *Environment* → $\mathbb{B}$

wf-unqualified-export (*mk-Unqualified-export* (*ids*))ρ $\triangleq$
  ids $\subseteq$ *visible-identifiers* (ρ) 319

6.1.9.2 Qualified Exports

The identifiers exported by a qualified export of a local module are introduced for use in the procedure or module in which the local module declaration appears. In the block of that procedure or module, the exported identifiers can be used only if qualified by the identifier of the local module.

Concrete Syntax

qualified export = "EXPORT", "QUALIFIED", identifier list, semicolon ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NOTE 1 — The identifier list of the qualified export contains identifiers that are distinct from each other.

Abstract Syntax

types

Qualified-export :: *ids* : *Identifier*-set

annotations The number of elements in the set of identifiers of the abstract syntax of a qualified export is equal to the number of identifiers in the identifier list of the concrete syntax.

Declaration Semantics

The identifiers that are exported by a qualified export shall be the identifiers in the identifier list of that qualified export, together with any identifiers that are implicitly exported with those identifiers — see 6.1.10.

d-qualified-export : *Qualified-export* → *Environment* → *Module-environment*

d-qualified-export (*mk-Qualified-export* (*ids*))ρ ≜
  let *closure* = $\bigcup$ {*x-closure* (*id*)ρ | *id* ∈ *ids*} in 49
  restrict-environment (*closure*)ρ 320

Static Semantics

The identifiers in the identifier list of a qualified export shall be distinct from each other.

The identifiers in the identifier list of a qualified export shall be declared in the declarations of the local module that contains the qualified export.

NOTE 2 — The identifiers in the identifier list of a qualified export are explicitly exported. Identifiers implicitly exported from a local module need not be declared in declarations of the local module since export of an enumeration type identifier implicitly exports the enumeration constant identifiers of the enumeration type whether or not the type identifier and the enumeration constant identifiers are introduced by the same declaration — see 6.1.10.

wf-qualified-export : *Qualified-export* → *Environment* → $\mathbb{B}$

wf-qualified-export (*mk-Qualified-export* (*ids*))ρ ≜
  ids ⊆ *visible-identifiers* (ρ) 319

Auxiliaries

exported-identifiers : [*Export-list*] → *Identifier*-set

exported-identifiers (*export*) ≜
  cases *export* :
    (nil) → { },
    mk-Unqualified-export (*ids*) → *ids*,
    mk-Qualified-export (*ids*) → *ids*
  end

6.1.10 Implicit Import and Export

The set of identifiers that is imported or exported if an identifier is explicitly imported or exported is called the (import and export) closure of that identifier. Normally, the closure includes only the explicitly imported or exported identifier. However, in the case of the explicit import or export of an identifier of an enumeration type, the closure also includes the identifiers of the values of that type.

Implicit export applies to the identifiers that are exported (qualified) from separate modules, by virtue of their being the subject of a definition in a definition module, as well as to export from a local module that uses an export list.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Declaration Semantics

The closure of an identifier shall be a set consisting of that identifier, together with the closure of the type if the identifier is a type identifier.

The closure of a type shall be the empty set, except in the case that the type is an enumeration type, when the closure shall be the set of identifiers associated with the values of the enumeration type.

NOTE 1 — Thus if an identifier denoting an enumeration type is imported (exported), then the occurrence of that identifier in the import list (export list) is an abbreviation for the occurrence of that identifier, together with all of the identifiers associated with the constant values of the enumeration type.

CLARIFICATIONS

1 If an identifier associated with a module that contains an unqualified export list is imported (exported), then the identifiers occurring in the export list of the module shall not be automatically imported (exported).

2 If an entity of a record type is imported or exported, the field identifiers of that record type are automatically available to be used as selectors. This does not constitute implicit import or export since, unlike the identifiers of enumeration values, field identifiers cannot be the subject of explicit import or export.

x-closure : *Identifier* → *Environment* → *Identifier*-set

x-closure (*id*)ρ $\triangleq$
  if *is-type* (*id*)ρ 314
  then $\{id\} \cup$ *x-closure-typed* (*entity*)ρ 49
  else $\{id\}$

x-closure-typed : *Typed* → *Environment* → *Identifier*-set

x-closure-typed (*typed*)ρ $\triangleq$
  let *struc* = *structure-of* (*typed*)ρ in 315
  if *is-Enumeration-structure*(*struc*)
  then let *mk-Enumeration-structure* (*values*) = *struc* in
    elems *values*
  else { }

annotations The closure of a type is the empty set, except in the case that the type is an enumeration type, when it is the set of identifiers associated with the values of the enumeration type.

NOTE 2 — Identifiers implicitly exported from a module need not be declared in definitions or declarations of the module since export of an enumeration type identifier implicitly exports the enumeration constant identifiers of the enumeration type. This is true whether or not the type identifier and the enumeration constant identifiers are introduced by the same definition or declaration — see 6.1.9 and 6.1.3. This situation is illustrated by the following example which uses unqualified export from a local module although similar comments apply to export from a separate module (through definitions), and to qualified export from a local module. There is only a single defining occurrence of the enumeration constant identifiers in the declarations of M, as is required for well-formed declarations — see 6.2.6.

EXAMPLE — Implicit export of enumeration constant identifiers.

```
MODULE M;

MODULE M1;
EXPORT T1;  (* implicitly exports red, green, and blue *)
TYPE T1 = (red, green, blue);
END M1;

MODULE M2;
IMPORT T1;
EXPORT T2;  (* implicitly exports red, green, and blue *)
TYPE T2 = T1;
END M2;

CONST shade = M2.red;

END M;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.1.11 Protected Modules

An interrupt request may be generated by the system underlying a Modula-2 implementation to indicate that an external condition has arisen that requires the attention of the program. Depending on the run-time environment of an implementation, interrupt requests may originate from hardware device controllers, or may be generated by an underlying operating system.

The acceptance of such a request involves the interruption of the normal sequence of execution (see 7.2) and may need to be postponed if the program is not prepared to handle the request, if it is a condition of relatively low priority, or if acceptance would lead to interference with the section of program that is under execution.

A program module, implementation module, or local module may specify, by including an interrupt protection in its heading, the extent to which the execution of enclosed statement sequences is to be protected from interrupts.

Protection from interrupts is provided by setting the current protection to the value specified in the module heading during initialization and finalization of the module and on activation of procedures exported from the module.

The pervasive identifiers `UNINTERRUPTIBLE` and `INTERRUPTIBLE` are provided to denote protection values. All maskable interrupts are disabled if the current protection is specified as `UNINTERRUPTIBLE`. All interrupts are enabled if the current protection is specified as `INTERRUPTIBLE` — see 6.10.2.11. The current protection is initialized to an implementation-defined value — see 6.12.2.1.

A module with a protection in its heading is called a protected module. Procedures exported from a protected module are called protected procedures.

Procedures that are not exported, and therefore not directly protected, are indirectly protected if called from a protected block body. However, such procedures may be made available, unprotected, for use outside a protected module if their values are returned as a result of a procedure call.

The declaration of a protected local module might occur within the declarations of another protected module. In all cases, the protection that is applied to the protection domain of a protected module is the protection specified for that module. Only a single protection is applied on entry to a protection domain.

Concrete Syntax

interrupt protection = left bracket, protection expression, right bracket ;

protection expression = constant expression ;

Abstract Syntax

types

Interrupt-protection :: *expr* : *Expression*

Static Semantics

The protection expression of a program module or implementation module shall be a constant expression of the protection type in the context of the pervasive identifier environment — see 6.10.2.11.

The protection expression of a local module that is protected from interrupts shall be a constant expression of the protection type given the environment that applies at the point of the local module declaration.

CHANGE — In *Programming in Modula-2*, the identifiers that have meaning in a protection expression of a separate module or a local module are not explicitly discussed, and the only examples given of protection expressions are single literal values. It is necessary to liberalize this, at least to the pervasive identifiers, in order to handle the definition of protection (for which, in general, literal values may vary between implementations).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wf-interrupt-protection : *Interrupt-protection* → *Environment* → 𝔹

wf-interrupt-protection (*mk-Interrupt-protection* (*expr*))ρ ≙
  is-constant-expression (*expr*)ρ ∧ 254
  t-expression (*expr*)ρ = PROTECTION-TYPE 190

Dynamic Semantics

For a module with interrupt protection, the value of the protection expression shall be used as the protection value during execution within the protection domain of the module.

evaluate-interrupt-protection : *Interrupt-protection* → *Environment* $\xrightarrow{o}$ *Protection-domain*

evaluate-interrupt-protection (*mk-Interrupt-protection* (*expr*))ρ ≙
  def *val* = *m-expression*(*expr*) ρ in 190
  return *mk-Protection-domain* (*val*)

Provided the protection of the new protection domain does not allow interrupts from any source that is not allowed by the current protection, entering an interrupt protection domain shall cause the current interrupt protection to be saved and then set to the value specified by the protection expression of the protected module; otherwise an exception shall be raised.

NOTE — In cases where a local static module with interrupt protection is declared within the declarations of another static module with interrupt protection, the raising of this exception is inevitable if the protection of the inner module allows interrupts from a source not allowed by the protection of the outer module. Implementations are free to detect such situations prior to execution — see 4.7.

activate-entry-protection : *Protection-domain* → *Environment* $\xrightarrow{o}$ *Protection-domain*

activate-entry-protection (*mk-Protection-domain* (*val*))ρ ≙
  if *is-less-restrictive-protection*(*val*) 344
  then let *handler* = *current-exception-handler* (ρ) in 318
    mandatory-detection(LESSPROTECTION) *handler* 396
  else def *oval* = *switch-protection*(*val*) in 344
    return *mk-Protection-domain* (*oval*)

Leaving an interrupt protection domain shall cause the current interrupt protection to be restored to the value saved on the corresponding entry to the protection domain.

activate-exit-protection : *Protection-domain* → *Environment* $\xrightarrow{o}$ ()

activate-exit-protection (*mk-Protection-domain* (*val*))ρ ≙
  set-protection(*val*) 344

Auxiliaries

is-protected-procedure : *Proper-procedure-declaration* → *Environment* → 𝔹

is-protected-procedure (*name*)ρ ≙
  name ∈ *exported-from-protected* (ρ) 318

is-protected-module : *Environment* → 𝔹

is-protected-module (ρ) ≙
  let *pdom* = *module-protection-domain* (ρ) in *pdom* ≠ nil 318

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.2 Definitions and Declarations

Definitions and declarations serve to introduce the identifiers of a module or procedure into their scope. Definitions appear in definition modules; declarations appear in program modules, implementation modules, local modules and procedures.

With the exception of the predefined pervasive identifiers, each identifier that is used in a module must either be introduced by a definition or a declaration in that module or must be introduced by import from another module.

With the exception of the predefined pervasive identifiers, each identifier that is used in a procedure must either be introduced by a declaration in that procedure or must have a scope that includes the block in which the procedure is declared.

The scope of an identifier comprises the entirety of the definitions, blocks, or statement sequences within which the identifier may be used. Within its scope, an identifier has a unique meaning. However, it is not always the case that the meaning of an identifier is known throughout its scope. For example, the meaning of an identifier that is used in a declaration is not known if that identifier is introduced into its scope by a subsequent declaration in the same sequence of declarations. The static semantics require that, where the meaning of an identifier needs to be known for a declaration to be elaborated, uses of that identifier in a declaration must not precede any declaration of that identifier.

If a point where identifiers may be used is included in the scope of a particular identifier, that identifier is said to be visible at that point of use, whether or not its meaning is known. In this part of ISO/IEC 10514, the visibility and meaning of an identifier at the point of its use is determined by the environment that applies at that point — see 6.11.1.

The declarations of procedures and of local modules may include statements within the blocks of those procedures and local modules. This part of ISO/IEC 10514 defines a meaning for programs in which the statements of such declarations use identifiers whose scope includes those statements but whose introduction to that scope is given in a subsequent declaration in the same sequence of declarations. However, a conforming implementation may reject programs that use identifiers in statements that precede the declaration of those identifiers — see 4.3. The alternative static semantic rules that may be applied are given in 6.2.7.

6.2.1 Identifiers

An identifier denotes a constant, type, variable, procedure, or module.

NOTE — The syntactic class identifier is an element of the lexis — see section 5.

Abstract Syntax

types

$$Identifier = token_{ident}$$

annotations A set of tokens that are in a 1–1 correspondence with the set of identifiers in the syntactic class identifier.

The elements of *Identifier* are denoted by identifiers in the type style used for Modula-2 source text in this part of ISO/IEC 10514.

6.2.2 Qualified Identifiers

The term qualified identifier is used in situations where the Modula-2 concrete syntax permits a single identifier to be qualified by the identifier of the module that exports it. Since module identifiers may be exported from local modules, identifiers may be qualified by more than one module identifier.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Concrete Syntax

qualified identifier = { qualifying identifier, period }, identifier ;

qualifying identifier = module identifier ;

Abstract Syntax

types

$Qualident = Identifier \mid Qualified;$

$Qualified :: id : Identifier$
$\quad qualificand : Qualident$

annotations It is the responsibility of the translation process from concrete to abstract syntax to distinguish between the use of the period for module qualification and the use of the period for field selection.

Static Semantics

An identifier that is not qualified by a module identifier shall be visible in the environment that applies at the point of use of that identifier or else shall be a pervasive identifier — see 6.10.

Each qualifying identifier of a qualified identifier shall be the name of a module from which the succeeding identifier is exported.

$wf\text{-}qualident : Qualident \to Environment \to \mathbb{B}$

$wf\text{-}qualident\,(qid)\rho \triangleq$
$\quad (is\text{-}Identifier(qid) \to \text{let } \rho_{env} = overwrite\text{-}environment\,(\rho)\rho_{PERV} \text{ in}$ 319
$\qquad\qquad qid \in visible\text{-}identifiers\,(\rho_{env}),$ 319
$\quad\; is\text{-}Qualified(qid) \to wf\text{-}qualified\,(qid)\rho)$ 53

$wf\text{-}qualified : Qualified \to Environment \to \mathbb{B}$

$wf\text{-}qualified\,(qid)\rho \triangleq$
$\quad \text{let } mk\text{-}Qualified\,(id, qual) = qid \text{ in}$
$\quad \text{let } \rho_{mods} = \rho.mods \text{ in}$
$\quad \text{if } id \in \text{dom } \rho_{mods}$
$\quad \text{then let } \rho_{mod} = \rho_{mods}\,(id).env \text{ in}$
$\qquad (is\text{-}Identifier(qual) \to qual \in visible\text{-}identifiers\,(\rho_{mod}),$ 319
$\qquad\; is\text{-}Qualified(qual) \to wf\text{-}qualified\,(qual)\rho_{mod})$ 53
$\quad \text{else false}$

Auxiliaries

$unqualify : Qualident \to Environment \to (Identifier \times Environment)$

$unqualify\,(qid)\rho \triangleq$
$\quad (is\text{-}Identifier(qid) \to \text{let } \rho_{env} = overwrite\text{-}environment\,(\rho)\rho_{PERV} \text{ in}$ 319
$\qquad\qquad mk\text{-}(qid, \rho_{env}),$
$\quad\; is\text{-}Qualified(qid) \to unqualified\,(qual)\rho)$ 53

$unqualified : Qualified \to Environment \to (Identifier \times Environment)$

$unqualified\,(qid)\rho \triangleq$
$\quad \text{let } mk\text{-}Qualified\,(id, qual) = qid \text{ in}$
$\quad \text{let } \rho_{mods} = \rho.mods \text{ in}$
$\quad \text{let } \rho_{mod} = \rho_{mods}\,(id).env \text{ in}$
$\quad (is\text{-}Identifier(qual) \to mk\text{-}(qid, \rho_{mod}),$
$\quad\; is\text{-}Qualified(qid) \to unqualified\,(qual)\rho_{mod})$ 53

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations The result is the environment of the module containing the declaration of the identifier component of the qualified identifier, together with that identifier.

6.2.3 Definitions

Definitions declare the identifiers that are to be exported from a separate module and introduce those identifiers into their scope. Definitions also specify certain permanent properties of the associated entities denoted by the identifiers, such as:

— whether the entity is a constant and, if so, its associated type and value;

— whether the entity is a type and, if so, possibly its structure and in the case of a new ordinal type, its set of permitted values;

— whether the entity is a variable and, if so, its type and possibly its address;

— whether the entity is a procedure and, if so, the structure of its heading.

The identifier is used to denote the associated entity, but only in those parts of the program that are within the scope of the identifier.

NOTE 1 — The scope of an identifier introduced in a definition includes the definition module in which the definition appears, the module block of the corresponding implementation module, the definitions of definition modules that import the identifier, and the module blocks of other modules that import the identifier.

Concrete Syntax

```
definitions = { definition } ;

definition =
    "CONST", { constant declaration, semicolon }      |
    "TYPE", { type definition, semicolon }            |
    "VAR", { variable declaration, semicolon }        |
    procedure heading, semicolon ;
```

Abstract Syntax

types

Definitions = *Definition**;

Definition = *Constant-declaration* | *Type-definition* | *Variable-declaration* | *Procedure-heading*

Declaration Semantics

The identifiers and entities defined by the definitions of a definition module shall be the identifiers and entities defined by each definition in the context of any preceding definition of the definition module.

d-definitions : *Definitions* → *Environment* → *Environment*

d-definitions (*defs*)ρ ≙
  if *defs* = [ ]
  then ρ_{empty}
  else let ρ_{def} = *d-definition* (hd *defs*)ρ, 55
       ρ_{rest} = *d-definitions* (tl *defs*)ρ in 54
    merge-environment (ρ_{def}, ρ_{rest}) 319

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations The environment constructed from each definition is used in the construction of the environment from any following definitions

d-definition : *Definition* → *Environment* → *Environment*

d-definition (*def*)ρ ≙
  (*is-Constant-declaration*(*def*) → *d-constant-declaration* (*def*)ρ, 67
  is-Type-definition(*def*) → *d-type-definition* (*def*)ρ, 57
  is-Variable-declaration(*def*) → *d-variable-declaration* (*def*)ρ, 70
  is-Procedure-heading(*def*) → *d-procedure-heading-declaration* (*def*)ρ) 58

Static Semantics

The identifiers that are used in the definitions of a definition module shall be identifiers whose scope includes those definitions.

An identifier that is used in a definition may not be one whose defining occurrence is given in a following definition, unless its use is within a new pointer type or a new procedure type and it is subsequently defined as a type identifier.

NOTE 2 — The use of an identifier within a new pointer type or a new procedure type follows the keywords `POINTER TO` or `PROCEDURE` and is 'shielded' from the declare-before-use-in-declarations rule that otherwise applies.

The definitions of a definition module may include only one defining occurrence of a particular identifier.

wf-definitions : *Definitions* → *Environment* → $\mathbb{B}$

wf-definitions (*defs*)ρ ≙
  let ρ_{def} = *overwrite-environment* (*d-definitions* (*defs*)ρ_{def})ρ in 319 54
  is-disjoint-definitions (*defs*)ρ_{def} ∧ 55
  is-complete-definitions (*defs*)ρ_{def} ∧ 55
  ∀ i ∈ inds *defs* ·
    let *predefs* = *front* (*defs*, $i - 1$) in 614
    let ρ_{pre} = *overwrite-environment* (*d-definitions* (*predefs*)ρ_{pre})ρ in 319 54
    let ρ_s = *build-shielded-environment* (*defs*(i)) in 89
    let ρ_f = *merge-environment* (ρ_{pre}, ρ_s) in 319
    wf-definition (*defs*(i))ρ_f 55

annotations ρ_{def} is only defined for a consistent set of definitions, thus it is necessary to check that each identifier is defined only once, that shielded identifiers are visible, and that the define before use in definitions rule is satisfied.

is-disjoint-definitions : *Definitions* → *Environment* → $\mathbb{B}$

is-disjoint-definitions (*defs*)ρ ≙
  ∀ i, j ∈ inds *defs* ·
    $i \neq j$ ⇒ let ρ_i = *d-definition* (*defs*(i))ρ, 55
      ρ_j = *d-definition* (*defs*(j))ρ in 55
    visible-identifiers (ρ_i) ∩ *visible-identifiers* (ρ_j) = { } 319 319

annotations Check that no identifier is defined twice in a sequence of definitions.

is-complete-definitions : *Definitions* → *Environment* → $\mathbb{B}$

is-complete-definitions (*defs*)ρ ≙
  ∀ *def* ∈ elems *defs* ·
    ∀ *qid* ∈ *shielded* (*def*) · 89
      is-type (*qid*)ρ 314

annotations Check that all identifiers shielded by POINTER TO or PROCEDURE in new types in a sequence of definitions are visible to those definitions as type identifiers.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wf-definition : *Definition* → *Environment* → $\mathbb{B}$

wf-definition (*def*)ρ $\triangleq$
  (*is-Constant-declaration*(*def*) → *wf-constant-declaration* (*def*)ρ, 68
  is-Type-definition(*def*) → *wf-type-definition* (*def*)ρ, 57
  is-Variable-declaration(*def*) → *wf-variable-declaration* (*def*)ρ, 70
  is-Procedure-heading(*def*) → *wf-procedure-heading* (*def*)ρ) 58

Dynamic Semantics

The elaboration of the definitions of a definition module shall associate, in the resulting execution environment, the identifiers and entities defined by each definition.

m-definitions : *Definitions* → *Environment* $\xrightarrow{o}$ *Environment*

m-definitions (*defs*)ρ $\triangleq$
  let ρ_{defs} = *overwrite-environment* (*d-definitions* (*defs*)ρ_{defs})ρ in 319 54
  construct-def-storage(*defs*) ρ_{defs} 56

annotations Resolve all definitions and then allocate storage.

construct-def-storage : *Definitions* → *Environment* $\xrightarrow{o}$ *Environment*

construct-def-storage (*defs*)ρ $\triangleq$
  if *defs* = [ ]
  then return ρ
  else def ρ_{def} = *m-definition*(hd *defs*) ρ in 56
    let ρ_{new} = *overwrite-environment* (ρ_{def})ρ in 319
    construct-def-storage(tl *defs*) ρ_{new} 56

m-definition : *Definition* → *Environment* $\xrightarrow{o}$ *Environment*

m-definition (*def*)ρ $\triangleq$
  (*is-Constant-declaration*(*def*) → *m-constant-declaration*(*def*) ρ, 68
  is-Type-definition(*def*) → *m-type-definition*(*def*) ρ, 57
  is-Variable-declaration(*def*) → *m-variable-declaration*(*def*) ρ, 71
  is-Procedure-heading(*def*) → skip)

6.2.4 Type Definitions

Type definitions serve to introduce an identifier that denotes a type and to specify that type and possibly its structure and set of permitted values. If the structure is omitted, the type definition defines an opaque type.

Concrete Syntax

type definition = type declaration | opaque type definition ;

opaque type definition = identifier ;

Abstract Syntax

types

Type-definition = *Type-declaration* | *Opaque-type-definition*;

Opaque-type-definition :: *id* : *Identifier*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Declaration Semantics

The identifier of an opaque type definition shall be defined as the identifier of a unique opaque type; otherwise the identifiers and entities defined by a type definition shall be the identifiers and entities defined by a type declaration — see 6.2.9.

d-type-definition : *Type-definition* → *Environment* → *Environment*

d-type-definition (*def*)ρ ≙
(*is-Type-declaration*(*def*) → *d-type-declaration* (*def*)ρ, 69
is-Opaque-type-definition(*def*) → *d-opaque-type-definition* (*def*)ρ) 57

d-opaque-type-definition : *Opaque-type-definition* → *Environment* → *Environment*

d-opaque-type-definition (*mk-Opaque-type-definition* (*id*))ρ ≙
let *typenm* = *generate-type-name* (ρ) in 320
μ (ρ_{empty}, *types* ↦ {*id* ↦ *typenm*}, *strucs* ↦ {*typenm* ↦ OPAQUE})

annotations A unique type name is generated for the opaque type, and the type identifier is associated with that type name; the environment contains a record of the new type and the structure of the type as being opaque.

Static Semantics

If a type definition is a type declaration it shall have the static semantics of a type declaration — see 6.2.9.

wf-type-definition : *Type-definition* → *Environment* → $\mathbb{B}$

wf-type-definition (*def*)ρ ≙
(*is-Type-declaration*(*def*) → *wf-type-declaration* (*def*)ρ, 69
is-Opaque-type-definition(*def*) → *wf-opaque-type-definition* (*def*)ρ) 57

wf-opaque-type-definition : *Opaque-type-definition* → *Environment* → $\mathbb{B}$

wf-opaque-type-definition (*mk-Opaque-type-definition* (-))ρ ≙
true

Dynamic Semantics

The elaboration of a type definition that is a type declaration shall associate the identifiers and entities of the type declaration in the resulting execution environment.

NOTE — No dynamic semantics need to be given for the elaboration of opaque type definitions since, during program execution, a corresponding pointer type declaration is elaborated.

m-type-definition : *Type-definition* → *Environment* $\xrightarrow{o}$ *Environment*

m-type-definition (*def*)ρ ≙
(*is-Type-declaration*(*def*) → *m-type-declaration*(*def*) ρ, 69
is-Opaque-type-definition(*def*) → skip)

annotations Opaque type definitions have no dynamic semantics.

6.2.5 Procedure Headings

There are two kinds of procedure heading, namely proper procedure headings and function procedure headings. A proper procedure heading is the heading of a proper procedure. A proper procedure is activated by a procedure call statement. A function procedure heading is the heading of a function procedure. A function procedure is activated by a function call as a factor of an expression, and yields a result that is used as an operand in the expression. A function procedure heading is distinguished from a proper procedure heading by the specification of the function result type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Procedure headings occur in definitions, within a definition module, and also occur in declarations, within a module or procedure block. Within declarations, a procedure heading has its own associated procedure block — see 6.2.11. One of the properties of a procedure heading, and its associated procedure block, is its 'declaration level'. This is taken from the declaration level of the definitions or declarations in which the procedure heading occurs. Definitions, and the blocks of static modules, have a declaration level of 0. The declaration level of a procedure block is one more than the declaration level of the block in which the procedure is declared. The declaration level is required for static semantic rules governing the assignment of procedure values to procedure variables — see 6.4.2.

Concrete Syntax

procedure heading = proper procedure heading | function procedure heading ;

Abstract Syntax

types

Procedure-heading = *Proper-procedure-heading* | *Function-procedure-heading*

Declaration Semantics

d-procedure-heading-declaration : *Procedure-heading* → *Environment* → *Environment*

d-procedure-heading-declaration (*def*)ρ $\triangleq$
(*is-Proper-procedure-heading*(*def*) → *d-proper-procedure-heading-declaration* (*def*)ρ, 59
is-Function-procedure-heading(*def*) → *d-function-procedure-heading-declaration* (*def*)ρ) 60

Static Semantics

wf-procedure-heading : *Procedure-heading* → *Environment* → $\mathbb{B}$

wf-procedure-heading (*def*)ρ $\triangleq$
(*is-Proper-procedure-heading*(*def*) → *wf-proper-procedure-heading* (*def*)ρ, 59
is-Function-procedure-heading(*def*) → *wf-function-procedure-heading* (*def*)ρ) 60

6.2.5.1 Proper Procedure Headings

A proper procedure heading declares the name (and, if present, the formal parameters) of a proper procedure.

Concrete Syntax

proper procedure heading = "PROCEDURE", procedure identifier, [formal parameters] ;

formal parameters = left parenthesis, [formal parameter list], right parenthesis ;

formal parameter list = formal parameter, { semicolon, formal parameter } ;

Abstract Syntax

types

Proper-procedure-heading :: *name* : *Identifier*
parms : *Formal-parameter-list*;

Formal-parameter-list = *Formal-parameter**

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Declaration Semantics

A proper procedure heading shall define the procedure identifier as a procedure identifier of a proper procedure, with a structure given by the declared number, positions, and formal types of parameters, and with the declaration level of the definitions or declarations in which it appears.

d-proper-procedure-heading-declaration : *Proper-procedure-heading* → *Environment* → *Environment*

d-proper-procedure-heading-declaration (*head*)ρ ≙
  let *ptyped* = *to-proc-type* (*head*)ρ in
  μ (ρ_{empty}, *procs* ↦ {*head.name* ↦ *mk-Procedure-value* (*ptyped*, nil)})

59

Static Semantics

The identifiers declared as formal parameters in the formal parameter list of a proper procedure heading shall be distinct from each other, and distinct from the formal type identifiers.

wf-proper-procedure-heading : *Proper-procedure-heading* → *Environment* → $\mathbb{B}$

wf-proper-procedure-heading (*mk-Proper-procedure-heading* (-, *parms*))ρ ≙
  is-formal-parameters-distinct (*parms*) ∧
  ∀ *parm* ∈ elems *parms* ·
    wf-formal-parameter (*parm*)ρ

60
62

Auxiliaries

to-proc-type :
  Proper-procedure-heading → *Environment* → *Proper-procedure-typed*

to-proc-type (*head*)ρ ≙
  let *nparms* = *construct-proper-procedure-struc* (*head*) in
  let *type* = *mk-Proper-procedure-structure* (*nparms*) in
  let *level* = *current-level* (ρ) in
  mk-Proper-procedure-typed (*type*, *level*)

59
317

construct-proper-procedure-struc : *Proper-procedure-heading* → *Proper-procedure-structure*

construct-proper-procedure-struc (*pph*) ≙
  let *mk-Proper-procedure-heading* (-, *parms*) = *pph* in
  let *fpl* = *construct-parameter-typed-list* (*parms*) in
  mk-Proper-procedure-structure (*fpl*)

59

annotations Convert a proper procedure heading into a proper procedure structure.

construct-parameter-typed-list : *Formal-parameter-list* → *Formal-parameters-typed*

construct-parameter-typed-list (*fpl*) ≙
  [*construct-parameter-typed* (*fpl*(*i*)) | *i* ∈ inds *fpl*]

59

annotations Convert a formal parameter list into a formal parameter type list.

construct-parameter-typed : *Formal-parameter* → *Formal-parameter-typed*

construct-parameter-typed (*fp*) ≙
  cases *fp* :
    mk-Value-parameter-specification (-, *ftype*) → *mk-Value-formal-typed* (*ftype*),
    mk-Variable-parameter-specification (-, *ftype*) → *mk-Variable-formal-typed* (*ftype*)
  end

annotations Convert a value or variable parameter specification into a value or variable formal type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

is-formal-parameters-distinct : *Formal-parameter-list* → 𝔹

is-formal-parameters-distinct (*parms*) ≜
  let *ids* = *identifiers-of-formal-parameter-list* (*parms*) in 76
  let *pds* = *identifiers-used-in-parameters* (*parms*) in 88
  let *tds* = {*id* | *id* : *Identifier* · *id* ∈ *pds* ∨ *mk-Qualified* (*id*, -) ∈ *pds*} in
  card *ids* = len *parms* ∧
  ids ∩ *tds* = { }

annotations Check that the identifiers declared in a formal parameter list are all distinct.

6.2.5.2 Function Procedure Headings

A function procedure heading declares the name (and, if present, the formal parameters) of a function procedure and declares the type of the result to be associated with the procedure identifier.

Concrete Syntax

function procedure heading =
  "PROCEDURE", procedure identifier, formal parameters,
  colon, function result type ;

function result type = type identifier ;

Abstract Syntax

types

Function-procedure-heading :: *name* : *Identifier*
  parms : *Formal-parameter-list*
  return : *Type-identifier*

Declaration Semantics

A function procedure heading shall define the procedure identifier as a procedure identifier of a function procedure, with a structure given by the declared number, positions, formal types of parameters, and result type, and with the declaration level of the definitions or declarations in which it appears.

d-function-procedure-heading-declaration : *Function-procedure-heading* → *Environment* → *Environment*

d-function-procedure-heading-declaration (*head*)ρ ≜
  let *ptyped* = *to-func-type* (*head*)ρ in 61
  μ (ρ$_{empty}$, *procs* ↦ {*head.name* ↦ *mk-Procedure-value* (*ptyped*, nil)})

Static Semantics

The identifiers declared as formal parameters in the formal parameter list of a function procedure heading shall be distinct from each other and distinct from the formal type identifiers. The qualified identifier that defines the return type of the function procedure shall denote a type.

wf-function-procedure-heading : *Function-procedure-heading* → *Environment* → 𝔹

wf-function-procedure-heading (*mk-Function-procedure-heading* (-, *parms*, *return*))ρ ≜
  is-formal-parameters-distinct (*parms*) ∧ 60
  wf-type-identifier (*return*)ρ ∧ 93
  ∀ *parm* ∈ elems *parms* ·
    wf-formal-parameter (*parm*)ρ 62

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Auxiliaries

to-func-type :
Function-procedure-heading → *Environment* → *Function-procedure-typed*

to-func-type (*head*)ρ ≙
let *nparms* = *construct-function-procedure-struc* (*head*) in 61
let *type* = *mk-Function-procedure-structure* (*nparms*) in
let *level* = *current-level* (ρ) in 317
mk-Function-procedure-typed (*type*, *level*)

construct-function-procedure-struc : *Function-procedure-heading* → *Function-procedure-type*

construct-function-procedure-struc (*fph*) ≙
let *mk-Function-procedure-heading* (-, *parms*, *return*) = *fph* in
let *fpl* = *construct-parameter-typed-list* (*parms*) in 59
mk-Function-procedure-structure (*fpl*, *return*)

annotations Convert a function procedure heading into a function procedure structure.

6:2.5.3 Formal Parameters

As an optional part of a procedure heading, a formal parameter provides an explicit interface between a block body that calls the procedure and the block body of the procedure. Calls of a procedure that has formal parameters must supply corresponding actual parameters. These are evaluated, and the resulting arguments are accessed in the procedure through the identifiers of the formal parameters.

NOTE — There may also be an implicit interface between the block bodies through variable identifiers whose scope includes both block bodies.

There are two kinds of formal parameter: value parameters and variable parameters. In each case, the type of a formal parameter is specified by a formal type — see 6.3.10.

The declaration semantics and static semantics of formal parameters are given in this clause. The dynamic semantics of formal parameters are given by procedure activation — see 6.9.4.

Concrete Syntax

formal parameter = value parameter specification | variable parameter specification ;

Abstract Syntax

types

Formal-parameter = *Value-parameter-specification* | *Variable-parameter-specification*

Declaration Semantics

d-formal-parameter : *Formal-parameter* → *Environment* → *Environment*

d-formal-parameter (*parm*)ρ ≙
cases *parm* :
mk-Value-parameter-specification (-, -) → *d-value-parameter-specification* (*parm*)ρ, 62
mk-Variable-parameter-specification (-, -) → *d-variable-parameter-specification* (*parm*)ρ 63
end

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Static Semantics

wf-formal-parameter : *Formal-parameter* → *Environment* → $\mathbb{B}$

wf-formal-parameter (*parm*)ρ $\triangleq$
  cases *parm* :
    mk-Value-parameter-specification (-, -) → *wf-value-parameter-specification* (*parm*)ρ, 62
    mk-Variable-parameter-specification (-, -) → *wf-variable-parameter-specification* (*parm*)ρ 63
  end

6.2.5.3.1 Value Parameters

Value parameters provide a means of explicitly passing a value into a procedure block body. Corresponding actual parameters must be expressions. In the procedure block body, a value parameter acts like a local variable to which the result of the evaluation of the actual parameter is assigned as an initial value.

Concrete Syntax

value parameter specification = identifier list, colon, formal type ;

Abstract Syntax

types

Value-parameter-specification :: *id* : *Identifier*
ftype : *Formal-type*

annotations In the translation from concrete to abstract syntax, each identifier in the identifier list is associated with its formal type.

Declaration Semantics

A value parameter specification shall define each identifier in the identifier list to be a value formal parameter having the declared formal type — see 6.3.10.

d-value-parameter-specification : *Value-parameter-specification* → *Environment* → *Value-formal-typed*

d-value-parameter-specification (*mk-Value-parameter-specification* (*id*, *ftype*))ρ $\triangleq$
  let *mk-Value-formal-typed* (*type*) = *d-formal-type* (*ftype*)ρ in 105
  $\{id \mapsto type\}$

Static Semantics

wf-value-parameter-specification : *Value-parameter-specification* → *Environment* → $\mathbb{B}$

wf-value-parameter-specification (*mk-Value-parameter-specification* (*id*, *ftype*))ρ $\triangleq$
  wf-formal-type (*ftype*)ρ 105

6.2.5.3.2 Variable Parameters

Variable parameters provide a means of explicitly passing a variable into a procedure block body. Corresponding actual parameters must designate variables. In the procedure block body, the variable parameter identifier stands as an alias for the designated variable.

Concrete Syntax

variable parameter specification = "VAR", identifier list, colon, formal type ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Abstract Syntax

types

Variable-parameter-specification :: *id* : *Identifier*
ftype : *Formal-type*

annotations In the translation from concrete to abstract syntax, each identifier in the identifier list is associated with its formal type.

Declaration Semantics

A variable parameter specification shall define each identifier in the identifier list to be a variable formal parameter having the declared formal type — see 6.3.10.

d-variable-parameter-specification : *Variable-parameter-specification* → *Environment* → *Variable-formal-typed*

d-variable-parameter-specification (*Variable-parameter-specification*(*id*, *ftype*))ρ ≙
let *mk-Variable-formal-typed* (*type*) = *d-formal-type* (*ftype*)ρ in 105
{*id* ↦ *type*}

Static Semantics

wf-variable-parameter-specification : *Variable-parameter-specification* → *Environment* → $\mathbb{B}$

wf-variable-parameter-specification (*mk-Variable-parameter-specification* (*id*, *ftype*))ρ ≙
wf-formal-type (*ftype*)ρ 105

6.2.6 Declarations

Declarations introduce identifiers into their scope in a procedure or module (other than a definition module). Declarations also specify certain permanent properties of the associated entities denoted by the identifiers, such as:

— whether the entity is a constant and, if so, its associated type and value;

— whether the entity is a type and, if so, its structure and, in the case of a new ordinal type, its set of permitted values;

— whether the entity is a variable and, if so, its type and possibly its address;

— whether the entity is a procedure and, if so, the structure of its heading and the associated block;

— whether the entity is a local module and, if so, its exported identifiers and its associated block.

The identifier is used to denote the associated entity, but only in those parts of the program that are within the scope of the identifier.

NOTE 1 — The scope of an identifier declared in a procedure is the procedure block, but this can be restricted by redeclaring it within a nested procedure or by not importing it into a (nested) local module. The scope of an identifier declared in a local module is the module block plus enclosing blocks to which it is exported, but this can be restricted by redeclaring it within a nested procedure or by not importing it into a (nested) local module.

Concrete Syntax

declarations = { declaration } ;

declaration =
"CONST", { constant declaration, semicolon }

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

"TYPE", { type declaration, semicolon }
"VAR", { variable declaration, semicolon }
procedure declaration, semicolon
local module declaration, semicolon ;

Abstract Syntax

types

Declarations = *Declaration**;

Declaration = *Constant-declaration* | *Type-declaration* | *Variable-declaration*
| *Procedure-declaration* | *Module-declaration*

Declaration Semantics

The identifiers and entities defined by a sequence of declarations shall be the identifiers and entities defined by each declaration in the context of any preceding declaration in the sequence.

d-declarations : *Declarations* → *Environment* → *Environment*

d-declarations (*decls*)ρ ≙
if *decls* = []
then ρ_{empty}
else let ρ_{decl} = *d-declaration* (hd *decls*)ρ, 64
ρ_{rest} = *d-declarations* (tl *decls*)ρ in 64
merge-environment (ρ_{decl}, ρ_{rest}) 319

annotations The environment constructed from each declaration is used in the construction of the environment from any following declarations

d-declaration : *Declaration* → *Environment* → *Environment*

d-declaration (*decl*)ρ ≙
(*is-Constant-declaration*(*decl*) → *d-constant-declaration* (*decl*)ρ, 67
is-Type-declaration(*decl*) → *d-type-declaration* (*decl*)ρ, 69
is-Variable-declaration(*decl*) → *d-variable-declaration* (*decl*)ρ, 70
is-Procedure-declaration(*decl*) → *d-procedure-declaration* (*decl*)ρ, 72
is-Module-declaration(*decl*) → *d-module-declaration* (*decl*)ρ) 80

Static Semantics

The identifiers that are used in the declarations of a module block or procedure block shall be identifiers whose scope includes those declarations.

An identifier that is used in a declaration may not be one whose defining occurrence is given in a following declaration, unless its use is within a new pointer type or a new procedure type and it is subsequently declared as a type identifier.

NOTE 2 — The use of an identifier within a new pointer type or a new procedure type follows the keywords `POINTER TO` or `PROCEDURE` and is 'shielded' from the declare-before-use-in-declarations rule that otherwise applies.

The declarations of a block may include only one defining occurrence of a particular identifier.

NOTES

3 Identifiers declared in a local module declaration, and exported from that module by an unqualified export, are effectively declared in the block that contains the module declaration — their scope includes that block and they are not imported into that block.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

4 Identifiers declared in a local module declaration, and exported from that module by a qualified export, may have the same spelling as identifiers whose scope includes the block that contains the module declaration. Uses of such identifiers must be qualified by the identifier of the local module.

wf-declarations : *Declarations* → *Environment* → 𝔹

wf-declarations (*decls*)ρ ≙
  let ρ_{dcl} = *overwrite-environment* (*d-declarations* (*decls*)ρ_{dcl})ρ in 319 64
  is-disjoint-declarations (*decls*)ρ_{dcl} ∧ 65
  wf-forward-declarations (*decls*)ρ_{dcl} ∧ 72
  is-complete-declarations (*decls*)ρ_{dcl} ∧ 65
  let *ids* = *visible-identifiers* (*d-declarations* (*decls*)ρ_{dcl}) in 319 64
  let ρ_{taken} = *remove-from-environment* (*ids*)ρ in 320
  ∀ *i* ∈ inds *decls* ·
    let *predecls* = *front* (*decls*, *i* − 1) in 614
    let ρ_{pre} = *overwrite-environment* (*d-declarations* (*predecls*)ρ_{pre})ρ_{taken} in 319 64
    let ρ_s = *build-shielded-environment* (*decls*(*i*)) in 89
    let ρ_f = *merge-environment* (ρ_{pre}, ρ_s) in 319
    wf-declaration (*decls*(*i*))ρ_f 65

annotations ρ_{dcl} is only defined for a consistent set of declarations, thus it is necessary to check that each identifier is declared only once, that shielded identifiers are visible, that the declare before use in declarations rule is satisfied, that all procedures that have been declared as **FORWARD** are fully declared, and that the bodies of modules and procedures are well-formed.

is-disjoint-declarations : *Declarations* → *Environment* → 𝔹

is-disjoint-declarations (*decls*)ρ ≙
  ∀ *i*, *j* ∈ inds *decls* ·
    i < *j* ∧ ¬ *is-forward-procedure-declaration* (*decls*(*i*)) ⇒ 72
    let ρ_i = *d-declaration* (*decls*(*i*))ρ, 64
      ρ_j = *d-declaration* (*decls*(*j*))ρ in 64
    let *common* = *visible-identifiers* (ρ_i) ∩ *visible-identifiers* (ρ_j) in 319 319
    common = { }
    ∨
    ∀ *id* ∈ *common* · *access-environment* (*id*)ρ_i = *access-environment* (*id*)ρ_j ∧ 319 319
      is-enumeration-constant (*id*)ρ_i 323

annotations Check that no identifier is declared twice in a sequence of declarations. Identifiers exported from local modules will be included in the check. Identical enumeration constants may be exported from more than one local module because of implicit export.

is-complete-declarations : *Declarations* → *Environment* → 𝔹

is-complete-declarations (*decls*)ρ ≙
  ∀ *decl* ∈ elems *decls* ·
    ∀ *qid* ∈ *shielded* (*decl*) · 89
      is-type (*qid*)ρ 314

annotations Check that all identifiers shielded by POINTER TO or PROCEDURE in new types in a sequence of declarations are visible to those definitions as type identifiers.

wf-declaration : *Declaration* → *Environment* → 𝔹

wf-declaration (*decl*)ρ ≙
  (*is-Constant-declaration*(*decl*) → *wf-constant-declaration* (*decl*)ρ, 68
   is-Type-declaration(*decl*) → *wf-type-declaration* (*decl*)ρ, 69
   is-Variable-declaration(*decl*) → *wf-variable-declaration* (*decl*)ρ, 70
   is-Procedure-declaration(*decl*) → *wf-procedure-declaration* (*decl*)ρ, 72
   is-Module-declaration(*decl*) → *wf-module-declaration* (*decl*)ρ) 80

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Dynamic Semantics

The elaboration of the declarations of a block shall associate, in the resulting execution environment, the identifiers and entities defined by each declaration.

m-declarations : *Declarations* → *Environment* $\xrightarrow{o}$ *Environment*

m-declarations (*decls*)ρ $\triangleq$
  let ρ_{dcls} = *overwrite-environment* (*d-declarations* (*decls*)ρ_{dcls})ρ in 319 64
  construct-dcl-storage(*decls*) ρ_{dcls} 66

annotations Resolve all declarations and then allocate storage.

construct-dcl-storage : *Declarations* → *Environment* $\xrightarrow{o}$ *Environment*

construct-dcl-storage (*decls*)ρ $\triangleq$
  if *decls* = [ ]
  then return ρ
  else def ρ_{dcl} = *m-declaration*(hd *decls*) ρ in 66
    let ρ_{new} = *overwrite-environment* (ρ_{dcl})ρ in 319
    construct-dcl-storage(tl *decls*) ρ_{new} 66

m-declaration : *Declaration* → *Environment* $\xrightarrow{o}$ *Environment*

m-declaration (*decl*)ρ $\triangleq$
  (*is-Constant-declaration*(*decl*) → *m-constant-declaration*(*decl*) ρ, 68
   is-Type-declaration(*decl*) → *m-type-declaration*(*decl*) ρ, 69
   is-Variable-declaration(*decl*) → *m-variable-declaration*(*decl*) ρ, 71
   is-Procedure-declaration(*decl*) → *m-procedure-declaration*(*decl*) ρ, 73
   is-Module-declaration(*decl*) → *m-module-declaration*(*decl*) ρ) 81

6.2.7 Ordering of Procedure and Module Declarations

This part of ISO/IEC 10514 gives meaning to programs that conform to either or both of the alternative static semantic rules for the ordering of procedure and module declarations given below.

NOTE — Modula-2 implementations are required to choose one of the alternative rules for the ordering of procedure and module declarations according to the criteria of 4.3.

Static Semantics

Declare Before Use in Declarations Model

The bodies of procedures and local modules declared within a sequence of declarations shall be well-formed given an environment in which all identifiers declared in that sequence are visible and in which the meaning of identifiers introduced in all declarations of that sequence is known. Thus, the identifiers used in the statements of a procedure declaration or the statements of a local module declaration of a block may be identifiers whose defining occurrence is given in a subsequent declaration of that block.

wf-declaration-bodies : *Declarations* → *Environment* → $\mathbb{B}$

wf-declaration-bodies (*decls*)ρ $\triangleq$
  let ρ_{body} = *overwrite-environment* (*d-declarations* (*decls*)ρ_{body})ρ in 319 64
  $\forall\, i \in$ inds *decls* · *wf-declaration-body* (*decls*(*i*))ρ_{body} 67

Declare Before Use Model

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The bodies of procedures and local modules declared within a sequence of declarations shall be well-formed given an environment in which all identifiers declared in that sequence are visible but in which the meaning of identifiers introduced in subsequent declarations of that sequence is unknown. Thus, the identifiers used in the statements of a procedure declaration or the statements of a local module declaration of a block may not be identifiers whose defining occurrence is given in a subsequent declaration of that block.

wf-declaration-bodies : *Declarations* → *Environment* → $\mathbb{B}$

wf-declaration-bodies (*decls*)ρ $\triangleq$
  let ρ_{body} = *overwrite-environment* (*d-declarations* (*decls*)ρ_{body})ρ in 319 64
  $\forall\, i \in$ inds *decls* ·
    let *predecls* = *front* (*decls*, *i*), 614
      postdecls = *rest* (*decls*, *i* + 1) in 614
    let ρ_i = *overwrite-environment* (*d-declarations* (*predecls*)ρ_i)ρ_{body}, 319 64
      ρ_r = *overwrite-environment* (*d-declarations* (*postdecls*)ρ_r)ρ_{body} in 319 64
    let ρ_s = *overwrite-environment* (*to-be-declared* (ρ_r))ρ_i in 319 320
    wf-declaration-body (*decl*(*i*))ρ_s 67

Auxiliaries

wf-declaration-body : *Declaration* → *Environment* → $\mathbb{B}$

wf-declaration-body (*decl*)ρ $\triangleq$
  (*is-Constant-declaration*(*decl*) → true,
   is-Type-declaration(*decl*) → true,
   is-Variable-declaration(*decl*) → true,
   is-Procedure-declaration(*decl*) → *wf-procedure* (*decl*)ρ, 73
   is-Module-declaration(*decl*) → *wf-module* (*decl*)ρ) 81

6.2.8 Constant Declarations

A constant declaration introduces an identifier as a synonym for a value. The value is specified by a constant expression. Each occurrence of the identifier is equivalent to an explicit occurrence of the value itself.

Concrete Syntax

constant declaration = identifier, equals, constant expression ;

Abstract Syntax

types

Constant-declaration :: *id* : *Identifier*
  expr : *Expression*

Declaration Semantics

The identifier of a constant declaration shall be defined as a constant identifier that is associated with the type and value given by the evaluation of the constant expression at the point of the constant declaration.

d-constant-declaration : *Constant-declaration* → *Environment* → *Environment*

d-constant-declaration (*mk-Constant-declaration* (*id*, *expr*))ρ $\triangleq$
  let *type* = *t-expression* (*expr*)ρ in 190
  let *value* = *evaluate-constant-expression* (*expr*)ρ in 257
  μ (ρ_{empty}, *consts* $\mapsto$ {*id* $\mapsto$ *mk-Constant-value* (*type*, *value*)})

annotations The identifier is associated with the constant value, and its type, in the consts component of the constructed environment.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Static Semantics

The expression component of a constant declaration shall be a constant expression. The identifier of a constant declaration shall not be used in the constant expression.

NOTE — The value of a constant expression can be determined statically since a constant expression can be completely evaluated by an analysis of the source code without executing the program — see 6.8.8.

wf-constant-declaration : *Constant-declaration* → *Environment* → $\mathbb{B}$

wf-constant-declaration (*mk-Constant-declaration* (*id*, *expr*))ρ $\triangleq$
  id $\notin$ *identifiers-used-in-expression* (*expr*) $\wedge$ 84
  is-constant-expression (*expr*)ρ $\wedge$ 254
  wf-expression (*expr*)ρ 190

Dynamic Semantics

The elaboration of a constant declaration shall associate, in the resulting execution environment, the identifier of the declaration, the value obtained from the evaluation of the constant expression, and the type of that value.

m-constant-declaration : *Constant-declaration* → *Environment* $\xrightarrow{o}$ *Environment*

m-constant-declaration (*mk-Constant-declaration* (*id*, *expr*))ρ $\triangleq$
  let *type* = *t-expression* (*expr*)ρ in 190
  def *value* = *m-expression*(*expr*) ρ in 190
  return μ (ρ_{empty}, *consts* $\mapsto$ {*id* $\mapsto$ *mk-Constant-value* (*type*, *value*)})

6.2.9 Type Declarations

Every value and every variable has a type — see 6.3 and 6.10.2. The type of a value determines the operations that may be applied to that value — see 6.4.1 and 6.9.3.1. The type of a variable determines the set of values that may be assigned to that variable and determines the other operations that may be applied to that variable — see 6.4.2 and 6.9.3.2.

Type declarations are used to introduce new identifiers for existing identified types and to introduce identifiers for new types. Type identifiers may be used in variable declarations to specify the type of declared variables and are needed in procedure headings to specify the types of formal parameters and the return type of function procedures.

NOTES

1 Type identifiers are not always needed in variable declarations since variables can be declared to be of a new type. In that case, the new type is an anonymous type since no type identifier is associated with that type.

2 Each occurrence of a new type defines a type that is different from any other type.

Concrete Syntax

type declaration = identifier, equals, type denoter ;

Abstract Syntax

types

Type-declaration :: *id* : *Identifier*
  type : *Type*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Declaration Semantics

The identifier of a type declaration shall be defined as a type identifier that is associated with the unique type denoted by the type denoter.

NOTE 3 — If the type denoter of a type declaration denotes a new enumeration type, the identifiers of the identifier list of the enumeration type are declared as constant identifiers of values of the enumeration type — see 6.3.4.

d-type-declaration : *Type-declaration* → *Environment* → *Environment*

d-type-declaration (*mk-Type-declaration* (*id*, *type*))ρ $\triangleq$
  let *mk-* (*typen*, ρ_{type}) = *d-type* (*type*)ρ in 92
  μ (ρ_{empty}, *types* $\mapsto$ {*id* $\mapsto$ *typen*}, *strucs* $\mapsto$ ρ_{type})

annotations The identifier is associated with a unique name for the type in the types component of the constructed environment, and the unique name for the type is associated with its structure in the strucs component of that environment.

Static Semantics

The identifier of a type declaration shall not be used in the type denoter unless its use is within a new pointer type or a new procedure type.

EXAMPLE — Use of the identifier of a type declaration in the type denoter.

```
ScanState =  (* The type of lexical scanning control procedures *)
  PROCEDURE (CHAR, VAR ScanClass, VAR ScanState);
```

This type declaration is taken from the standard strings conversion library — see 9.5.

wf-type-declaration : *Type-declaration* → *Environment* → $\mathbb{B}$

wf-type-declaration (*mk-Type-declaration* (*id*, *type*))ρ $\triangleq$
  id $\notin$ *identifiers-used-in-type* (*type*) $\wedge$ 87
  wf-type (*type*)ρ 92

Dynamic Semantics

The elaboration of a type declaration shall associate, in the resulting execution environment, the identifier of the declaration and the unique type that is the result of the elaboration of the type denoter

m-type-declaration : *Type-declaration* → *Environment* $\xrightarrow{o}$ *Environment*

m-type-declaration (*mk-Type-declaration* (*id*, *type*))ρ $\triangleq$
  return *d-type-declaration* (*mk-Type-declaration* (*id*, *type*))ρ 69

6.2.10 Variable Declarations

A variable declaration serves to declare one or more variables of a specified type and to introduce an identifier for each variable. Each variable declared in a variable declaration can be associated with a specified address.

Concrete Syntax

variable declaration = variable identifier list, colon, type denoter ;

variable identifier list = identifier, [machine address], { comma, identifier, [machine address] } ;

machine address = left bracket, value of address type, right bracket ;

value of address type = constant expression ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Abstract Syntax

types

Variable-declaration :: *ids* : *Variable-identifier*-set
type : *Type*;

Variable-identifier :: *id* : *Identifier*
addr : [*Expression*]

annotations Identifiers may only occur once in a variable identifier list, and thus the cardinality of the set of identifiers of a *Variable-declaration* is the same as the number of identifiers in the variable identifier list.

Declaration Semantics

The identifiers of a variable declaration shall be defined as variable identifiers of variables that are of the unique type denoted by the type denoter.

NOTES

1 If the type denoter of a type declaration denotes a new type, the new type is an anonymous type since no type identifier is associated with that type.

2 If the type denoter of a type declaration denotes a new enumeration type, the identifiers of the identifier list of the enumeration type are declared as constant identifiers of values of the enumeration type — see 6.3.4.

$d\text{-}variable\text{-}declaration : Variable\text{-}declaration \to Environment \to Environment$

$d\text{-}variable\text{-}declaration\ (mk\text{-}Variable\text{-}declaration\ (vids, type))\rho \triangleq$
let $mk\text{-}(typen, \rho_{type}) = d\text{-}type\ (type)\rho$ in 92
$\mu\ (\rho_{empty}, vars \mapsto \{id \mapsto typen \mid mk\text{-}Variable\text{-}identifier\ (id, \text{-}) \in vids\}, strucs \mapsto \rho_{type})$

annotations The identifiers are associated with a unique name for the type in the vars component of the constructed environment, and the unique name for the type is associated with its structure in the strucs component of that environment.

Static Semantics

The identifiers occurring in the variable identifier list of a variable declaration shall not be used in the type denoter.

Each machine address appearing in a variable declaration shall be a constant expression of the address type.

NOTES

3 Since the declarations of a block may include only one defining occurrence of a particular identifier, the identifiers occurring in the variable identifier list of a variable declaration must be distinct from each other — see 6.2.6.

4 Constant values of the address type may only be formed by application of the predefined function procedures **MAKEADR** or **CAST** from the system module **SYSTEM**. The use of a machine address to fix the address of a variable thus requires a direct or indirect import from **SYSTEM**.

$wf\text{-}variable\text{-}declaration : Variable\text{-}declaration \to Environment \to \mathbb{B}$

$wf\text{-}variable\text{-}declaration\ (mk\text{-}Variable\text{-}declaration\ (vars, type))\rho \triangleq$
let $ids = \{id \mid mk\text{-}Variable\text{-}identifier\ (id, \text{-}) \in vars\}$ in
$ids \cap identifiers\text{-}used\text{-}in\text{-}type\ (type) = \{\}\ \wedge$ 87
$wf\text{-}type\ (type)\rho\ \wedge$ 92
$\forall\ mk\text{-}Variable\text{-}identifier\ (\text{-}, expr) \in vars \cdot$
$expr = \text{nil}\ \vee$
$(wf\text{-}expression\ (expr)\rho\ \wedge$ 190
$is\text{-}constant\text{-}expression\ (expr)\rho\ \wedge$ 254
let $etype = t\text{-}expression\ (expr)\rho$ in 190
$is\text{-}address\text{-}type\ (etype)\rho)$ 324

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Dynamic Semantics

The elaboration of a variable declaration shall associate, in the resulting execution environment, the identifiers of the declaration with variables of the unique type denoted by the type denoter.

Storage shall be allocated for each variable as appropriate for the type of the variables. If a machine address is used in declaring a variable, the storage shall be allocated at the given address.

m-variable-declaration : *Variable-declaration* → *Environment* $\xrightarrow{o}$ *Environment*

m-variable-declaration (*mk-Variable-declaration* (*ids*, *type*))ρ $\triangleq$
  let ρ_{var} = *d-variable-declaration* (*mk-Variable-declaration* (*ids*, *type*))ρ in 70
  allocate-variables(*ids*) ρ_{var} 71

annotations The elaboration of a variable declaration involves the elaboration of the type information to give the structure of the storage associated with the type, followed by the allocation of that storage. The variable identifier is associated with the storage by using a mapping in the environment.

Auxiliaries

allocate-variables : *Variable-identifier*-set → *Environment* $\xrightarrow{o}$ *Environment*

allocate-variables (*vars*)ρ $\triangleq$
  if *vars* = { }
  then return ρ
  else let *var* ∈ *vars* in
    def ρ_{var} = *allocate-a-variable*(*var*) ρ in 71
    allocate-variables(*vars* − {*var*}) ρ_{var} 71

annotations Storage is allocated for each of the variables declared in the variable declaration.

allocate-a-variable : *Variable-identifier* → *Environment* $\xrightarrow{o}$ *Environment*

allocate-a-variable (*var*)ρ $\triangleq$
  let *mk-Variable-identifier* (*id*, *expr*) = *var* in
  let *type* = *access-environment* (*id*)ρ in 319
  def *var* = *allocate-program-storage*(*type*) ρ in 337
  (if *expr* ≠ nil
   then def *val* = *m-expression*(*expr*) ρ in 190
     alias-machine-address(*var*, *val*) 336
   else skip;
   return μ (ρ_{empty}, *vars* ↦ {*id* ↦ *var*}))

annotations If the variable is of an elementary type, storage is allocated for it and information about the storage is added to the environment. If the variable is structured, then storage is allocated for each of its components, and the storage information is added to the environment. The result of this operation is the updated environment.

6.2.11 Procedure Declarations

There are two kinds of procedures, namely proper procedures and function procedures. A proper procedure is activated by a procedure call statement. A function procedure is activated by a function call as a factor of an expression, and yields a result that is used as an operand in the expression. A function procedure declaration is distinguished from a proper procedure declaration by its heading — see 6.2.5.

A full procedure declaration consists of a procedure heading followed by a procedure block. The procedure block gives the declarations that are to be elaborated when the procedure is activated and the procedure body that is to be executed on that activation.

A forward procedure declaration consists of a procedure heading followed by the keyword FORWARD. A forward declaration may be used to avoid the forward references that would otherwise occur when two or more full procedure

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

declarations contain mutually recursive references, and which will not be accepted by an implementation choosing the 'declare before use model' — see 6.2.7.

Concrete Syntax

procedure declaration = proper procedure declaration | function procedure declaration ;

Abstract Syntax

types

Procedure-declaration = *Proper-procedure-declaration* | *Function-procedure-declaration*

Declaration Semantics

d-procedure-declaration : *Procedure-declaration* → *Environment* → *Environment*

d-procedure-declaration (*decl*)ρ $\triangleq$
(*is-Proper-procedure-declaration*(*decl*) → *d-proper-procedure-declaration* (*decl*)ρ, 73
is-Function-procedure-declaration(*decl*) → *d-function-procedure-declaration* (*decl*)ρ) 77

Static Semantics

If a forward declaration of a procedure is made, it shall precede a corresponding full declaration of that procedure. The full declaration shall be made in the same block or in a nested local module from which it is exported unqualified.

A procedure that is exported from a separate module, having a corresponding procedure heading in the definition module, shall not be given a forward procedure declaration in the implementation module.

The parameter lists of the headings of the forward declaration of the procedure and full declaration of that procedure shall match. The formal parameter lists of procedure headings shall match if the formal types of each of the value parameter specifications or variable parameter specifications of corresponding formal parameters are the same.

NOTE — The rule for matching the procedure headings of forward and full procedure declarations is the same as the rule for matching the definition of a procedure in a definition module and the corresponding declaration of that procedure in the implementation module — see 6.1.5. The identifiers used to denote the parameters need not be the same, and the identifiers used to denote the types of these parameters need not be the same.

wf-forward-declarations : *Declarations* → *Environment* → $\mathbb{B}$

wf-forward-declarations (*decls*)ρ $\triangleq$
$\forall\, i \in$ inds *decls* ·
is-forward-procedure-declaration (*decls*(*i*)) ⇒ 72
$\exists!\, j \in$ inds *decls* · ¬ *is-forward-procedure-declaration* (*decls*(*j*)) ∧ 72
let ρ_i = *d-declaration* (*decls*(*i*))ρ, 64
ρ_j = *d-declaration* (*decls*(*j*))ρ in 64
procedures-declared (ρ_i, ρ_j) 36

is-forward-procedure-declaration : *Declaration* → $\mathbb{B}$

is-forward-procedure-declaration (*decl*) $\triangleq$
(*is-Proper-procedure-declaration*(*decl*) → *decl.block* = FORWARD,
is-Function-procedure-declaration(*decl*) → *decl.block* = FORWARD,
others → false)

wf-procedure-declaration : *Procedure-eclaration* → *Environment* → $\mathbb{B}$

wf-procedure-declaration (*decl*)ρ $\triangleq$
(*is-Proper-procedure-declaration*(*decl*) → *wf-proper-procedure-declaration* (*decl*)ρ, 74
is-Function-procedure-declaration(*decl*) → *wf-function-procedure-declaration* (*decl*)ρ) 77

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wf-procedure : *Declaration* → *Environment* → $\mathbb{B}$

wf-procedure (*decl*)ρ $\triangleq$
  (*is-Proper-procedure-declaration*(*decl*) → *wf-proper-procedure* (*decl*)ρ, 74
   is-Function-procedure-declaration(*decl*) → *wf-function-procedure* (*decl*)ρ) 78

Dynamic Semantics

m-procedure-declaration : *Declaration* → *Environment* $\xrightarrow{o}$ *Environment*

m-procedure-declaration (*decl*)ρ $\triangleq$
  (*is-Proper-procedure-declaration*(*decl*) → if *decl.block* = FORWARD
      then return ρ
      else *m-proper-procedure-declaration*(*decl*) ρ, 75
   is-Function-procedure-declaration(*decl*) → if *decl.block* = FORWARD
      then return ρ
      else *m-function-procedure-declaration*(*decl*) ρ) 78

6.2.11.1 Proper Procedure Declarations

A proper procedure declaration consists of a proper procedure heading, which declares the procedure identifier and the formal parameters, followed by the keyword **FORWARD** in the case of a forward procedure declaration, or followed by a proper procedure block in the case of a full procedure declaration. The proper procedure block contains the local declarations and the procedure body that is executed when the procedure is called.

Concrete Syntax

proper procedure declaration =
    proper procedure heading, semicolon,
    (proper procedure block, procedure identifier | "FORWARD") ;

procedure identifier = identifier ;

In a full proper procedure declaration the procedure identifier of the proper procedure heading shall be identical to the procedure identifier following the proper procedure block of that declaration.

CHANGE — The keyword **FORWARD** is not in early editions of *Programming in Modula-2*.

Abstract Syntax

types

  Proper-procedure-declaration :: *head* : *Proper-procedure-heading*
                                  block : (*Proper-procedure-block* | FORWARD)

Declaration Semantics

A proper procedure declaration shall, through its heading, declare its identifier as a procedure identifier of a proper procedure structure, with the declared number, positions, and types of parameters, and with the declaration level of the block in which it is declared.

NOTE 1 — The blocks of static modules are at the declaration level of 0. The declaration level of a procedure block is one more than the declaration level of the block in which the procedure is declared.

d-proper-procedure-declaration : *Proper-procedure-declaration* → *Environment* → *Environment*

d-proper-procedure-declaration (*mk-Proper-procedure-declaration* (*head*, *block*))ρ $\triangleq$
  d-proper-procedure-heading-declaration (*head*)ρ

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Within the block of a full proper procedure declaration, the proper procedure heading shall declare the identifiers of any formal parameters to be variable identifiers of variables that are of the type denoted by the corresponding formal type.

d-proper-procedure-heading : *Proper-procedure-heading* → *Environment* → *Environment*

d-proper-procedure-heading (*mk-Proper-procedure-heading* (*name*, *parms*))ρ ≙
  let *lenv* = $\bigcup$ {*d-formal-parameter* (*parm*)ρ | *parm* ∈ elems *parms*} in 61
  μ (ρ_{empty}, *vars* ↦ *lenv*)

Static Semantics

The identifiers declared as formal parameters in the formal parameter list of the proper procedure heading of a proper procedure declaration shall be distinct from each other, and distinct from the formal type identifiers.

The identifiers declared in the declarations of a full proper procedure declaration shall be distinct from the identifiers declared as formal parameters in the formal parameter list of the proper procedure heading, and the declarations shall be well-formed given the environment that applies at the point of the procedure declaration, as overwritten by the parameters declared in the procedure heading.

wf-proper-procedure-declaration : *Proper-procedure-declaration* → *Environment* → $\mathbb{B}$

wf-proper-procedure-declaration (*mk-Proper-procedure-declaration* (*head*, *block*))ρ ≙
  wf-proper-procedure-heading (*head*)ρ ∧ 59
  (*block* = FORWARD
    ∨
   is-procedure-declarations-distinct (*head*, *block*) ∧ 76
   let ρ_{block} = *add-proc-parameters* (*head*)ρ in 75
   let *mk-Proper-procedure-block* (*decls*, -) = *block* in
   wf-declarations (*decls*)ρ_{block}) 65

Given a declaration level greater by one than the declaration level of the block in which the procedure is declared, and given the environment that applies at the point of the procedure declaration, as overwritten by the parameters declared in the procedure heading:

— the bodies of module and procedure declarations of a full proper procedure declaration shall be well-formed — see 6.2.7,

— if present, the body of the procedure block shall be a well-formed proper procedure body — see 6.5.1.

NOTE 2 — The environment that applies at the point of the procedure declaration depends upon the 'declare before use' rule that is applied — see 6.2.7.

wf-proper-procedure : *Proper-procedure-declaration* → *Environment* → $\mathbb{B}$

wf-proper-procedure (*decl*)ρ ≙
  block = FORWARD ∨
  let *mk-Proper-procedure-declaration* (*head*, *block*) = *decl* in
  let *mk-Proper-procedure-block* (*decls*, -) = *block* in
  let ρ_{block} = *add-proc-parameters* (*head*)ρ in 75
  let ρ_{inner} = *new-level* (ρ_{block}) in 318
  wf-declaration-bodies (*decls*)ρ_{inner} ∧ 67
  wf-proper-procedure-body (*block*)ρ_{inner} 126

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Auxiliaries

add-proc-parameters : *Proper-procedure-heading* → *Environment* → *Environment*

add-proc-parameters (*head*)ρ ≙
  let *parmsenv* = *d-proper-procedure-heading* (*head*)ρ in 74
  overwrite-environment (*parmsenv*)ρ 319

Dynamic Semantics

The elaboration of a full proper procedure declaration shall associate, in the resulting execution environment, the procedure identifier with a procedure value corresponding to the proper procedure block. The associated procedure value shall be such that, on activation, the actual parameters will be bound to the formal parameters and the proper procedure block will be executed.

NOTES

3 The binding of the actual parameters to the formal parameters includes the allocation of storage for the parameters — see 6.9.4.

4 The elaboration of the declarations of the procedure block, including the allocation of storage for the local variables of the procedure block, occurs on the execution of the block.

5 Conforming implementations are required to document rules for the calculation of the amount of workspace that will be needed by a single activation of a particular proper procedure (without taking into account the workspace needed by any procedure that it calls) — see 4.9d.

The associated procedure value shall also be such that, on normal or exceptional completion of the execution of the proper procedure block, the storage associated with the parameters will be freed.

m-proper-procedure-declaration : *Proper-procedure-declaration* → *Environment* $\xrightarrow{o}$ *Environment*

m-proper-procedure-declaration (*mk-Proper-procedure-declaration* (*head*, *block*))ρ ≙
  let *mk-Proper-procedure-heading* (*name*, *parms*) = *head* in
  let *f* = *generate-proper-procedure* (*name*, *parms*, *block*)ρ in 75
  def *pid* = *store-denotation*(*f*) in 339
  let *ptyped* = *to-proc-type* (*head*)ρ in 59
  return μ (ρ_{empty}, *procs* ↦ {*name* ↦ *mk-Procedure-value* (*ptyped*, *pid*)})

generate-proper-procedure :
  Identifier × *Formal-parameter-list* × *Proper-procedure-block* → *Environment* → *Proper-procedure-value*

generate-proper-procedure (*name*, *parms*, *block*)ρ ≙
  let *f* = λ *args* : *Arguments* ·
    λ *handler* : *Exception-handler* ·
      def ρ_{args} = *bind-arguments*(*parms*, *args*) ρ in 265
      def ρ_{cont} = *c-tixe*({EXCEPTION ↦ (*deallocate-parameters*(*parms*) ρ_{args}; 345 76
          execute(*handler*))}) ρ in 345
      ((¬ *is-handled* (*block*) ∧ ¬ *is-protected-procedure* (*name*)ρ → 76 51
          m-proper-procedure-block(*block*) ρ_{cont}, 127
        ¬ *is-handled* (*block*) ∧ *is-protected-procedure* (*name*)ρ → 76 51
          m-proper-procedure-block-p(*block*) ρ_{cont}, 127
        is-handled (*block*) ∧ ¬ *is-protected-procedure* (*name*)ρ → 76 51
          m-proper-procedure-block-h(*block*) ρ_{cont}, 128
        is-handled (*block*) ∧ *is-protected-procedure* (*name*)ρ → 76 51
          m-proper-procedure-block-hp(*block*) ρ_{cont}); 128
       deallocate-parameters(*parms*) ρ_{args}) in 76
  f

annotations The elaboration of a proper procedure declaration is a proper procedure denotation that is the meaning of the parameters followed by the meaning of the block of the procedure.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Auxiliaries

is-handled : *Block-body* → 𝔹

is-handled (*mk-Block-body* (-, *except*)) ≜
  except ≠ nil

deallocate-parameters : *Formal-parameter-list* → *Environment* $\xrightarrow{o}$ ()

deallocate-parameters (*parms*)ρ ≜
  let *pids* = *identifiers-of-formal-parameter-list* (*parms*) in 76
  deallocate-storage(*pids*) ρ 337

is-procedure-declarations-distinct : *Proper-procedure-heading* × *Proper-procedure-block* → 𝔹

is-procedure-declarations-distinct (*head*, *block*) ≜
  let *fids* = *identifiers-of-formal-parameter-list* (*head.parms*) in 76
  let *vars* = *identifiers-declared-in* (*block.decls*) in 82
  fids ∩ *vars* = { }

annotations Check that the identifiers declared in a proper procedure heading are not re-declared in the associated proper procedure block.

identifiers-of-formal-parameter-list : *Formal-parameter-list* → *Identifier*-set

identifiers-of-formal-parameter-list (*parms*) ≜
  {*identifier-of-a-formal-parameter*(*parm*) | *parm* ∈ elems *parms*}

annotations The result is a set of identifiers that name the formal parameters of a formal parameter list.

identifier-of-a-formal-parameter : *Formal-parameter* → *Identifier*

identifier-of-a-formal-parameter (*parm*) ≜
  cases *parm* :
    mk-Value-parameter-specification (*id*, -) → *id*,
    mk-Variable-parameter-specification (*id*, -) → *id*
  end

annotations The result is an identifier that is the name of a formal parameter.

6.2.11.2 Function Procedure Declarations

A function procedure declaration consists of a function procedure heading, which declares the procedure identifier, the formal parameters and the result type of the function procedure, followed by the keyword FORWARD in the case of a forward procedure declaration, or followed by a function procedure block in the case of a full procedure declaration. The function procedure block contains the local declarations and the procedure body that evaluates the result of the procedure.

Concrete Syntax

function procedure declaration =
  function procedure heading, semicolon,
  (function procedure block, procedure identifier | "FORWARD") ;

In a full function procedure declaration the procedure identifier of the function procedure heading shall be identical to the procedure identifier following the function procedure block of that declaration.

CHANGE — The keyword FORWARD is not in early editions of *Programming in Modula-2*.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Abstract Syntax

types

Function-procedure-declaration :: *head* : *Function-procedure-heading*
block : (*Function-procedure-block* | FORWARD)

Declaration Semantics

A function procedure declaration shall, through its heading, declare its identifier as a procedure identifier of a function procedure structure, with the declared number, positions, and types of parameters, with the declared result type, and with the declaration level of the block in which it is declared.

NOTE 1 — The blocks of static modules are at the declaration level of 0. The declaration level of a procedure block is one more than the declaration level of the block in which the procedure is declared.

d-function-procedure-declaration : *Function-procedure-declaration* → *Environment* → *Environment*

d-function-procedure-declaration (*mk-Function-procedure-declaration* (*head*, *block*))ρ ≜
d-function-procedure-heading-declaration (*head*)ρ 60

Within the block of a full function procedure declaration, the function procedure heading shall declare the identifiers of any formal parameters to be variable identifiers of variables that are of the type denoted by the corresponding formal type.

d-function-procedure-heading : *Function-procedure-heading* → *Environment* → *Environment*

d-function-procedure-heading (*mk-Function-procedure-heading* (*name*, *parms*, *return*))ρ ≜
let *lenv* = $\bigcup$ {*d-formal-parameter* (*parms*(*i*))ρ | *i* ∈ inds *parms*} in 61
μ (ρ_{empty}, *vars* ↦ *lenv*)

Static Semantics

The identifiers declared as formal parameters in the formal parameter list of the proper procedure heading of a proper procedure declaration shall be distinct from each other, and distinct from the formal type identifiers.

The identifiers declared in the declarations of a full function procedure declaration shall be distinct from the identifiers declared as formal parameters in the formal parameter list of the function procedure heading, and the declarations shall be well-formed given the environment that applies at the point of the procedure declaration, as overwritten by the parameters declared in the procedure heading.

wf-function-procedure-declaration : *Function-procedure-declaration* → *Environment* → $\mathbb{B}$

wf-function-procedure-declaration (*mk-Function-procedure-declaration* (*head*, *block*))ρ ≜
wf-function-procedure-heading (*head*)ρ ∧ 60
(*block* = FORWARD
∨
is-function-declarations-distinct (*head.parms*, *block*) ∧ 79
let ρ_{block} = *add-func-parameters* (*head*)ρ in 78
let *mk-Function-procedure-block* (*decls*, -) = *block* in
wf-declarations (*decls*)ρ_{block}) 65

Given a declaration level greater by one than the declaration level of the block in which the procedure is declared, and given the environment that applies at the point of the procedure declaration, as overwritten by the parameters declared in the procedure heading:

— the bodies of module and procedure declarations of a full function procedure declaration shall be well-formed — see 6.2.7,

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

— given the type denoted by the function result type, the body of the procedure block shall be a well-formed function procedure body — see 6.5.2.

NOTE 2 — The environment that applies at the point of the procedure declaration depends upon the 'declare before use' rule that is applied — see 6.2.7.

wf-function-procedure : *Function-procedure-declaration* → *Environment* → 𝔹

wf-function-procedure (*decl*)ρ ≙
  block = FORWARD ∨
  let *mk-Function-procedure-declaration* (*head*, *block*) = *decl* in
  let *mk-Function-procedure-block* (*decls*, -) = *block* in
  let ρ_{block} = *add-func-parameters* (*head*)ρ in 78
  let ρ_{inner} = *new-level* (ρ_{block}) in 318
  wf-declaration-bodies (*decls*)ρ_{inner} ∧ 67
  let *mk-* (*rtype*, -) = *d-type-identifier* (*head.return*)ρ in 93
  wf-function-procedure-body (*rtype*, *block*)ρ_{inner} 129

Auxiliaries

add-func-parameters : *Proper-procedure-heading* → *Environment* → *Environment*

add-func-parameters (*head*)ρ ≙
  let *parmsenv* = *d-function-procedure-heading* (*head*)ρ in 77
  overwrite-environment (*parmsenv*)ρ 319

Dynamic Semantics

The elaboration of a full function procedure declaration shall associate, in the resulting execution environment, the procedure identifier with a procedure value corresponding to the function procedure block. The associated procedure value shall be such that, on activation, the actual parameters will be bound to the formal parameters, and the function procedure block will be executed.

NOTES

3 The binding of the actual parameters to the formal parameters includes the allocation of storage for the parameters — see 6.9.4.

4 The elaboration of the declarations of the procedure block, including the allocation of storage for the local variables of the procedure, occurs on the execution of the block.

5 Conforming implementations are required to document rules for the calculation of the amount of workspace that will be needed by a single activation of a particular function procedure (without taking into account the workspace needed by any procedure that it calls) — see 4.9d.

The associated procedure value shall also be such that, on normal or exceptional completion of the execution of the function procedure block, the storage associated with the parameters will be freed, and, on normal completion, the result value will be returned to the caller of the function procedure.

NOTE 6 — Normal completion of a function procedure block occurs on the execution of a return statement of that block.

m-function-procedure-declaration : *Function-procedure-declaration* → *Environment* $\xrightarrow{o}$ *Environment*

m-function-procedure-declaration (*mk-Function-procedure-declaration* (*head*, *block*))ρ ≙
  let *mk-Function-procedure-heading* (*name*, *parms*, -) = *head* in
  let *f* = *generate-function-procedure* (*name*, *parms*, *block*)ρ in 78
  def *pid* = *store-denotation*(*f*) in 339
  let *ptyped* = *to-func-type* (*head*)ρ in 61
  return μ (ρ_{empty}, *procs* ↦ {*name* ↦ *mk-Procedure-value* (*ptyped*, *pid*)})

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

generate-function-procedure :
Identifier × *Formal-parameter-list* × *Function-procedure-block* → *Environment* → *Function-procedure-value*

generate-function-procedure (*name*, *parms*, *block*)ρ ≙
let $f = \lambda$ *args* : *Arguments* ·
λ *handler* : *Exception-handler* ·
def ρ_{args} = *bind-arguments*(*parms*, *args*) ρ in 265
def ρ_{cont} = *c-tixe*({EXCEPTION ↦ (*deallocate-parameters*(*parms*) ρ_{args}; 345 76
execute(*handler*))}) ρ in 345
def *result* = (¬ *is-handled* (*block*) ∧ ¬ *is-protected-procedure* (*name*)ρ → 76 51
m-function-procedure-block(*block*) ρ_{cont}, 130
¬ *is-handled* (*block*) ∧ *is-protected-procedure* (*name*)ρ → 76 51
m-function-procedure-block-p(*block*) ρ_{cont}, 130
is-handled (*block*) ∧ ¬ *is-protected-procedure* (*name*)ρ → 76 51
m-function-procedure-block-h(*block*) ρ_{cont}, 130
is-handled (*block*) ∧ *is-protected-procedure* (*name*)ρ → 76 51
m-function-procedure-block-hp(*block*) ρ_{cont}) in 131
(*deallocate-parameters*(*parms*) ρ_{args}; 76
return *result*) in

annotations The elaboration of a function procedure declaration is a function procedure denotation that is the meaning of the parameters followed by the meaning of the block of the procedure.

Auxiliaries

is-function-declarations-distinct : *Function-procedure-heading* × *Proper-procedure-block* → $\mathbb{B}$

is-function-declarations-distinct (*head*, *block*) ≙
let *fids* = *identifiers-of-formal-parameter-list* (*head.parms*) in 76
let *vars* = *identifiers-declared-in* (*block.decls*) in 82
fids ∩ *vars* = { }

annotations Check that the identifiers declared in a function procedure heading are not redeclared in the associated function procedure block.

6.2.12 Local Module Declarations

A local module consists of a sequence of import lists, an optional export list, a sequence of declarations, and an optional module body. Both of these sequences may be empty. The import lists, if present, control the visibility of identifiers declared outside the module but usable within it. The export list, if present, controls the visibility of identifiers declared inside the module and usable outside it.

A local module may have a protection specified in its heading. The statements and procedures in the domain of protection are then executed under a protection as specified by the protection expression and the optional access-control procedures — see 6.1.11.

Concrete Syntax

local module declaration =
"MODULE", module identifier, [interrupt protection], semicolon,
import lists, [export list],
module block, module identifier ;

The two module identifier components of a module declaration shall be identical.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Abstract Syntax

types

Module-declaration :: *name* : *Identifier*
imports : *Import-lists*
export : [*Export-list*]
block : *Module-block*
protection : [*Interrupt-protection*]

Declaration Semantics

A local module declaration shall declare its identifier as a module identifier that may be used to qualify the identifiers exported from that module. In the case of unqualified export, the scope of the exported identifiers shall include the block in which the module is declared.

If the module is a protected module, or if the module is exported from an enclosing protected module, any exported procedures shall be protected procedures; otherwise, any exported procedures of the module that are also exported from an enclosing protected module shall be protected procedures.

d-module-declaration : *Module-declaration* → *Environment* → *Environment*

d-module-declaration (*mk-Module-declaration* (*name*, *imports*, *export*, *block*, *protection*))ρ ≙
  let $\rho_{protected}$ = let *ex-ids* = *exported-identifiers* (*export*) in 48
    let *pr-ids* = *exported-from-protected* (ρ) in 318
    if *protection* ≠ nil ∨ *name* ∈ *pr-ids*
    then *mark-exported-from-protected* (*ex-ids*)ρ 318
    else *mark-exported-from-protected* (*ex-ids* ∩ *pr-ids*)ρ in 318
  let $\rho_{imported}$ = *d-import-lists* (*imports*)$\rho_{protected}$ in 42
  let ρ_{block} = *d-module-block* (*block*)$\rho_{imported}$ in 80
  let $\rho_{exported}$ = *d-export-list* (*export*)ρ_{block} in 46
  let $\rho_{module} = \mu(\rho_{empty}, mods \mapsto \{name \mapsto \rho_{exported}\})$ in
  if *is-Unqualified-export*(*export*)
  then *merge-environments* ($\{\rho_{exported}, \rho_{module}\}$) 319
  else ρ_{module}

d-module-block : *Module-block* → *Environment* → *Environment*

d-module-block (*mk-Module-block* (*decls*, *initial*, *final*))ρ ≙
  d-declarations (*decls*)ρ 64

Static Semantics

A local module shall not import itself. If present, the import lists and the declarations of a local module block shall be consistent with one another — see 6.1.8.5. If present, the import lists of a local module shall be valid given the environment that applies at the point of the local module declaration — see 6.1.8.

If present, an export list of a local module shall be valid given the environment that is constructed from the declarations of the local module block — see 6.1.9.

Any identifiers that are explicitly exported from a local module shall not be the identifiers of control variables of for statements of the module block.

The declarations of the local module shall be well-formed given the identifiers that are imported into the module.

If present in the local module heading, the interrupt protection shall be a valid interrupt protection in the context of the environment that applies at the point of the module declaration — see 6.1.11.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wf-module-declaration : *Module-declaration* → *Environment* → $\mathbb{B}$

wf-module-declaration (*mk-Module-declaration* (*name*, *imports*, *export*, *block*, *protection*))ρ $\triangleq$
  name $\notin$ elems *directly-imports* (*imports*) $\wedge$
  let *mk-Module-block* (*decls*, -, -) = *block* in
  consistent-imports (*imports*, *decls*)ρ $\wedge$ 46
  wf-import-lists (*imports*)ρ $\wedge$ 42
  let $\rho_{imports}$ = *d-import-lists* (*imports*)ρ in 42
  wf-declarations (*decls*)$\rho_{imports}$ $\wedge$ 65
  (*export* = nil $\vee$
   let ρ_{decls} = *overwrite-environment* (*d-declarations* (*decls*)ρ_{decls})$\rho_{imports}$ in 319 64
   wf-export-list (*export*)ρ_{decls} $\wedge$ 46
   cases *export* :
     mk-Unqualified-export (*ids*) → *ids* $\cap$ *control-variables* (*block*) = { }, 170
     mk-Qualified-export (*ids*) → *ids* $\cap$ *control-variables* (*block*) = { } 170
   end) $\wedge$
  protection = nil $\vee$ *wf-interrupt-protection* (*protection*)ρ 51

Given the environment that applies at the point of the local module declaration, restricted to the identifiers that are imported into the module:

— the bodies of module and procedure declarations of the module block shall be well-formed — see 6.2.7,

— if present, the body of the module block shall be a well-formed module body — see 6.5.3.

NOTE 1 — The environment that applies at the point of the local module declaration depends upon the 'declare before use' rule that is applied — see 6.2.7.

wf-module : *Module-declaration* → *Environment* → $\mathbb{B}$

wf-module (*decl*)ρ $\triangleq$
  let *mk-Module-declaration* (-, *imports*, -, *block*, -) = *decl* in
  let *mk-Module-block* (*decls*, -, -) = *block* in
  let ρ_{block} = *d-import-lists* (*imports*)ρ in 42
  wf-declaration-bodies (*decls*)ρ_{block} $\wedge$ 67
  wf-module-body (*block*)ρ_{block} 132

Dynamic Semantics

The elaboration of a local module declaration shall elaborate the declarations of the module block, given the meaning of the imported identifiers, and shall associate, in the resulting execution environment, the module identifier with the identifiers that are exported by the local module, and the identifiers that are exported unqualified with their associated entities.

If the local module is a protected module, the protection specified for that local module shall be used to protect the protection domain of the local module, instead of the protection of any enclosing protected module.

NOTE 2 — Only the innermost protection domain applies to procedures exported through more than one enclosing protected module.

m-module-declaration : *Module-declaration* → *Environment* $\xrightarrow{o}$ *Environment*

m-module-declaration (*mk-Module-declaration* (*name*, *imports*, *export*, *block*, *protection*))ρ $\triangleq$
  let $\rho_{imported}$ = *d-import-lists* (*imports*)ρ in 42
  def $\rho_{internal}$ = *m-module-block*(*block*) $\rho_{imported}$ in 132
  def *pval* = if *protection* = nil then nil
    else *evaluate-interrupt-protection*(*protection*) ρ in 51

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

let $\rho_{block} = add\text{-}protection\,(pval)\rho_{internal}$ in 318
let $\rho_{exported} = d\text{-}export\text{-}list\,(export)\rho_{block}$ in 46
let $\rho_{module} = \mu\,(\rho_{empty}, mods \mapsto \{name \mapsto \rho_{block}\})$ in
if $is\text{-}Unqualified\text{-}export(export)$
then return $merge\text{-}environment\,(\rho_{exported}, \rho_{module})$ 319
else return ρ_{module}

NOTE 3 — The local module is executed on static or dynamic module initialization — see 6.5.4 and 6.5.6.

6.2.13 Auxiliary Formal Model Definitions

6.2.13.1 Identifiers Defined in Definitions

identifiers-defined-in : *Definitions* → *Identifier*-set

identifiers-defined-in (*defs*) ≜
⋃ {*identifiers-defined* (*def*) | *def* ∈ elems *defs*} 82

annotations The result of this operation is a set containing those identifiers that have been defined in a sequence of definitions.

identifiers-defined : *Definition* → *Identifier*-set

identifiers-defined (*def*) ≜
identifiers-explicitly-defined (*def*) ∪ *identifiers-implicitly-defined* (*def*) 82 82

identifiers-explicitly-defined : *Definition* → *Identifier*-set

identifiers-explicitly-defined (*def*) ≜
cases *def* :
mk-Constant-declaration (*id*, -) → {*id*},
mk-Type-declaration (*id*, -) → {*id*},
mk-Opaque-type-definition (*id*) → {*id*},
mk-Variable-declaration (*ids*, -) → *ids*,
mk-Proper-procedure-heading (*name*, -) → {*name*},
mk-Function-procedure-heading (*name*, -, -) → {*name*}
end

annotations Return a set containing those identifiers that have been explicitly defined by a definition.

identifiers-implicitly-defined : *Definition* → *Identifier*-set

identifiers-implicitly-defined (*def*) ≜
cases *def* :
mk-Type-declaration (-, *type*) → *identifiers-defined-in-type* (*type*), 83
mk-Variable-declaration (-, *type*) → *identifiers-defined-in-type* (*type*), 83
others → { }
end

annotations Return a set containing those identifiers that have been implicitly defined in a definition - namely enumeration constants.

6.2.13.2 Identifiers Declared in Declarations

identifiers-declared-in : *Declarations* → *Identifier*-set

identifiers-declared-in (*decls*) ≜
⋃ {*identifiers-declared* (*decl*) | *decl* ∈ elems *decls*} 83

annotations The result of this function is a set containing those identifiers that have been declared in a sequence of declarations.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

identifiers-declared : *Declaration* → *Identifier*-set

identifiers-declared (*decl*) ≙
  identifiers-explicitly-declared (*decl*) ∪ *identifiers-implicitly-declared* (*decl*) 83 83

identifiers-explicitly-declared : *Declaration* → *Identifier*-set

identifiers-explicitly-declared (*decl*) ≙
  cases *decl* :
    mk-Constant-declaration (*id*, -) → {*id*},
    mk-Type-declaration (*id*, -) → {*id*},
    mk-Variable-declaration (*ids*, -) → *ids*,
    mk-Proper-procedure-declaration (*head*, -) → {*head.name*},
    mk-Function-proccdure-declaration (*head*, -) → {*head.name*},
    mk-Module-declaration (*name*, -, -, -, -) → {*name*}
  end

annotations Return a set containing those identifiers that have been declared in a declaration.

identifiers-implicitly-declared : *Declaration* → *Identifier*-set

identifiers-implicitly-declared (*decl*) ≙
  cases *decl* :
    mk-Type-declaration (-, *type*) → *identifiers-defined-in-type* (*type*), 83
    mk-Variable-declaration (-, *type*) → *identifiers-defined-in-type* (*type*), 83
    others → { }
  end

annotations Return a set containing those identifiers that have been implicitly declared in a declaration — namely enumeration constants.

6.2.13.3 Identifiers Defined in New Types

identifiers-defined-in-type : *Type* → *Identifier*-set

identifiers-defined-in-type (*type*) ≙
  cases *type* :
    mk-Type-identifier (*qid*) → { },
    mk-Enumeration-type (*values*) → elems *values*,
    mk-Subrange-type (-, -) → { },
    mk-Set-type (*btype*) → *identifiers-defined-in-type* (*btype*), 83
    mk-Pointer-type (*btype*) → *identifiers-defined-in-type* (*btype*), 83
    mk-Proper-procedure-type (-) → { },
    mk-Function-procedure-type (-, -) → { },
    mk-Array-type (*itype*, *ctype*) → *identifiers-defined-in-type* (*itype*) ∪ *identifiers-defined-in-type* (*ctype*), 83 83
    mk-Record-type (*fields*) → *identifiers-defined-in-fields-list* (*fields*) 83
  end

annotations Return a set of identifiers containing all the identifiers defined in a type component of a definition. These are the enumeration type constants (identifiers).

identifiers-defined-in fields-list : *Fields-list* → *Identifier*-set

identifiers-defined-in-fields-list (*fields*) ≙
  if *fields* = nil
  then { }
  else ⋃ {*identifiers-defined-in-fields* (*field*) | *field* ∈ elems *fields*} 84

annotations Return a set of the identifiers defined in a fields list.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

identifiers-defined-in-fields : *Fields* → *Identifier*-set

identifiers-defined-in-fields (*field*) ≜
  cases *field* :
    mk-Fixed-fields (-, *type*) → *identifiers-defined-in-type* (*type*), 83
    mk-Variant-fields (-, -, *variants*, *elsep*) → *identifiers-defined-in-variants* (*variants*) ∪ 84
      identifiers-defined-in-fields-list (*elsep*) 83
  end

annotations Return a set of the identifiers defined in fields. These are the enumeration type constants (identifiers) defined in any type components of the components of the fields.

identifiers-defined-in-variants : *Variant** → *Identifier*-set

identifiers-defined-in-variants (*variants*) ≜
  ⋃ {*identifiers-defined-in-a-variant* (*variant*) | *variant* ∈ elems *variants*} 84

annotations Return a set of the identifiers defined in variants. These are the enumeration type constants (identifiers) defined in the components of the variants.

identifiers-defined-in-a-variant : *Variant* → *Identifier*-set

identifiers-defined-in-a-variant (*variant*) ≜
  let *mk-Variant* (-, *fields*) = *variant* in
  identifiers-defined-in-fields-list (*fields*) 83

annotations Return a set of the identifiers defined in a variant. These are the enumeration type constants (identifiers) defined in the components of the variant.

6.2.13.4 Qualified Identifiers Used in Declarations

identifiers-used-in : *Declarations* → *Qualident*-set

identifiers-used-in (*decls*) ≜
  ⋃ {*identifiers-used* (*decl*) | *decl* ∈ elems *decls*} 84

annotations The result is a set containing those qualified identifiers that have been used in a sequence of declarations.

identifiers-used : *Declaration* → *Qualident*-set

identifiers-used (*decl*) ≜
  cases *decl* :
    mk-Constant-declaration (-, *expr*) → *identifiers-used-in-expression* (*expr*), 84
    mk-Type-declaration (-, *type*) → *identifiers-used-in-type* (*type*), 87
    mk-Variable-declaration (-, *type*) → *identifiers-used-in-type* (*type*), 87
    mk-Proper-procedure-declaration (-, -) → { },
    mk-Function-procedure-declaration (-, -) → { },
    mk-Module-declaration (-, -, -, -, -) → { }
  end

annotations The result is a set containing those qualified identifiers that have been used in the declaration of an identifier.

6.2.13.5 Qualified Identifiers used in Expressions

identifiers-used-in-expression : *Expression* → *Qualident*-set

identifiers-used-in-expression (*expr*) ≜
  (*is-Infix-expression*(*expr*) → *identifiers-used-in-infix-expression* (*expr*), 85
   is-Prefix-expression(*expr*) → *identifiers-used-in-prefix-expression* (*expr*), 85
   is-Value-designator(*expr*) → *identifiers-used-in-value-designator* (*expr*), 85
   is-Function-call(*expr*) → *identifiers-used-in-function-call* (*expr*), 85
   is-Value-constructor(*expr*) → *identifiers-used-in-value-constructor* (*expr*), 86
   is-Constant-literal(*expr*) → { })

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations The result is a set of qualified identifiers used in an expression; this is the set containing all the qualified identifiers used in the components of the expression.

identifiers-used-in-infix-expression : *Infix-expression* → *Qualident*-set

identifiers-used-in-infix-expression (*expr*) ≙
  let *mk-Infix-expression* (*left*, -, *right*) = *expr* in
  identifiers-used-in-expression (*left*) ∪ *identifiers-used-in-expression* (*right*) 84 84

annotations The result is the set of qualified identifiers used in an infix expression; this is the set that is the union of the sets of qualified identifiers used in both of the expression components of the infix expression.

identifiers-used-in-prefix-expression : *Prefix-expression* → *Qualident*-set

identifiers-used-in-prefix-expression (*pexpr*) ≙
  let *mk-Prefix-expression* (-, *expr*) = *pexpr* in
  identifiers-used-in-expression (*expr*) 84

annotations The result is the set of qualified identifiers used in a prefix expression; this is the set of qualified identifiers used in the expression component of the prefix expression.

identifiers-used-in-value-designator : *Value-designator* → *Qualident*-set

identifiers-used-in-value-designator (*vdesig*) ≙
  cases *vdesig* :
    mk-Entire-value (*desig*) → {*desig*},
    mk-Indexed-value (*desig*, *expr*) → *identifiers-used-in-value-designator* (*desig*) ∪ 85
      identifiers-used-in-expression (*expr*), 84
    mk-Selected-value (*desig*, -) → *identifiers-used-in-value-designator* (*desig*), 85
    mk-Dereferenced-value (*desig*) → *identifiers-used-in-value-designator* (*desig*) 85
  end

annotations The result is the set of qualified identifiers used in a value designator; this is the set of qualified identifiers used in either an entire value, an indexed value, a selected value, or a dereferenced value.

The set of qualified identifiers used in an entire value consists of the qualified identifier used to denote the entire value.

The set of qualified identifiers used in an indexed value consists of the qualified identifiers used in the value designator component together with the qualified identifiers used in the expression component.

The set of qualified identifiers used in a selected value consists of the qualified identifiers used in the value designator component.

The set of qualified identifiers used in a dereferenced value consists of the qualified identifiers used in the value designator component.

identifiers-used-in-function-call : *Function-call* → *Qualident*-set

identifiers-used-in-function-call (*fcall*) ≙
  let *mk-Function-call* (*desig*, *args*) = *fcall* in
  identifiers-used-in-value-designator (*desig*) ∪ *identifiers-used-in-actual-parameters* (*args*) 85 85

annotations The result is the set of qualified identifiers used in a function call; this is the set of qualified identifiers used in the function designator component, together with the qualified identifiers used in the actual parameters of the function call.

identifiers-used-in-actual-parameters : *Actual-parameters* → *Qualident*-set

identifiers-used-in-actual-parameters (*aps*) ≙
  ⋃ {*identifiers-used-in-an-actual-parameter* (*ap*) | *ap* ∈ elems *aps*} 85

annotations The result is the set of qualified identifiers used in the actual parameters of a function call; this set consists of all the qualified identifiers used in each of the actual parameters.

identifiers-used-in-an-actual-parameter : *Actual-parameter* → *Qualident*-set

identifiers-used-in-an-actual-parameter (*ap*) ≙
  (*is-Variable-designator*(*ap*) → *identifiers-used-in-variable-designator* (*ap*), 86
   is-Expression(*ap*) → *identifiers-used-in-expression* (*ap*), 84
   is-Type-parameter(*ap*) → let *mk-Type-parameter* (*type*) = *arg* in
     {*type*})

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations The result is a set of qualified identifiers used in an actual parameter to a procedure or function call; this set consists of the qualified identifiers used in the actual parameter.

identifiers-used-in-variable-designator : *Variable-designator* → *Qualident*-set

identifiers-used-in-variable-designator (*vdesig*) ≙
  cases *vdesig* :
    mk-Entire-designator (*desig*) → {*desig*},
    mk-Indexed-designator (*desig*, *expr*) → *identifiers-used-in-variable-designator* (*desig*) ∪ 86
      identifiers-used-in-expression (*expr*), 84
    mk-Selected-designator (*desig*, -) → *identifiers-used-in-variable-designator* (*desig*), 86
    mk-Dereferenced-designator (*desig*) → *identifiers-used-in-variable-designator* (*desig*) 86
  end

annotations The result is the set of qualified identifiers used in a variable designator; this is the set of qualified identifiers used in either an entire designator, an indexed designator, a selected designator, or a dereferenced designator.

The set of qualified identifiers used in an entire designator consists of the qualified identifier used to denote the entire variable.

The set of qualified identifiers used in an indexed designator consists of the qualified identifiers used in the variable designator component together with the qualified identifiers used in the expression component.

The set of qualified identifiers used in a selected designator consists of the qualified identifiers used in the variable designator component.

The set of qualified identifiers used in a dereferenced designator consists of the qualified identifiers used in the variable designator component.

identifiers-used-in-value-constructor : *Value-constructor* → *Qualident*-set

identifiers-used-in-value-constructor (*expr*) ≙
  (*is-Array-constructor*(*expr*) → *identifiers-used-in-array-constructor* (*expr*), 86
   is-Record-constructor(*expr*) → *identifiers-used-in-record-constructor* (*expr*), 86
   is-Set-constructor(*expr*) → *identifiers-used-in-set-constructor* (*expr*)) 86

annotations The result is the set of qualified identifiers used in a value constructor; this set consists of the qualified identifiers used either in an array constructor, a record constructor, or a set constructor.

identifiers-used-in-array-constructor : *Array-constructor* → *Qualident*-set

identifiers-used-in-array-constructor (*expr*) ≙
  let *mk-Array-constructor* (*qid*, *sval*) = *expr* in
  {*qid*} ∪ ⋃ {*identifiers-used-in-element* (*el*) | *el* ∈ elems *sval*} 86

annotations The result is the qualified identifiers used in an array constructor; this set consists of the union of the qualified identifiers used in all the expression components of the array constructor together with the qualified identifier used to denote the array.

identifiers-used-in-record-constructor : *Record-constructor* → *Qualident*-set

identifiers-used-in-record-constructor (*expr*) ≙
  let *mk-Record-constructor* (*qid*, *sval*) = *expr* in
  {*qid*} ∪ ⋃ {*identifiers-used-in-element* (*el*) | *el* ∈ elems *sval*} 86

annotations The result is the set of qualified identifiers used in a record constructor; this set consists of the union of the qualified identifiers used in all the expression components of the record constructor, together with the qualified identifier used to denote the record.

identifiers-used-in-element : *Element* → *Qualident*-set

identifiers-used-in-element (*el*) ≙
  let *mk-Element* (*val*, *rep*) = *el* in
  identifiers-used-in-expression (*val*) ∪ *identifiers-used-in-expression* (*rep*) 84 84

annotations The result is the set of qualified identifiers used in an element of a value constructor.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

identifiers-used-in-set-constructor : *Set-constructor* → *Qualident*-set

identifiers-used-in-set-constructor (*expr*) ≙
  let *mk-Set-constructor* (*qid*, *def*) = *expr* in
  {*qid*} ∪ ⋃ {*identifiers-used-in-member* (*expr*) | *expr* ∈ elems *def*} 87

annotations The result is the set of qualified identifiers used in a set constructor; this set consists of the union of the qualified identifiers used in all the members of the set constructor together with the qualified identifier used to denote the set.

identifiers-used-in-member : *Member* → *Qualident*-set

identifiers-used-in-member (*mem*) ≙
  cases *mem* :
    mk-Singleton (*expr*) → *identifiers-used-in-expression* (*expr*), 84
    mk-Interval (*min*, *max*) → *identifiers-used-in-expression* (*min*) ∪ *identifiers-used-in-expression* (*max*) 84 84
  end

annotations The result is the set of qualified identifiers used in a member; this set consists of the union of the qualified identifiers used in the expression component(s) of the member.

6.2.13.6 Qualified Identifiers Used in Types

identifiers-used-in-type : *Type* → *Qualident*-set

identifiers-used-in-type (*type*) ≙
  cases *type* :
    mk-Type-identifier (*qid*) → {*qid*},
    mk-Enumeration-type (-) → { },
    mk-Subrange-type (*rtype*, *range*) → let *mk-Interval* (*min*, *max*) = *range* in
      {*rtype*} ∪
      identifiers-used-in-expression (*min*) ∪ 84
      identifiers-used-in-expression (*max*), 84
    mk-Set-type (*btype*) → *identifiers-used-in-type* (*btype*), 87
    mk-Pointer-type (-) → { },
    mk-Proper-procedure-type (-) → { },
    mk-Function-procedure-type (-, -) → { },
    mk-Array-type (*itype*, *ctype*) → *identifiers-used-in-type* (*itype*) ∪ *identifiers-used-in-type* (*ctype*), 87 87
    mk-Record-type (*fields*) → *identifiers-used-in-fields-list* (*fields*) 87
  end

annotations The result is the set of qualified identifiers used in a type; the members of this set are the qualified identifiers used to denote type names and any qualified identifiers that appear in any constant expressions. The set does not include those type identifiers that are shielded either by `POINTER TO` or by `PROCEDURE`.

identifiers-used-in-fields-list : *Fields-list* → *Qualident*-set

identifiers-used-in-fields-list (*fields*) ≙
  if *fields* = nil
  then { }
  else ⋃ {*identifiers-used-in-fields* (*field*) | *field* ∈ elems *fields*} 87

annotations The result is the set of qualified identifiers used in a fields list; these are the qualified identifiers used to denote type names in each of the components of the fields list.

identifiers-used-in-fields : *Fields* → *Qualident*-set

identifiers-used-in-fields (*field*) ≙
  cases *field* :
    mk-Fixed-fields (-, *type*) → *identifiers-used-in-type* (*type*), 87
    mk-Variant-fields (-, *tagt*, *variants*, *elsep*) → {*tagt*} ∪
      identifiers-used-in-variants (*variants*) ∪ 88
      identifiers-used-in-fields-list (*elsep*) 87
  end

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations The result is the set of qualified identifiers used in fields; these are the qualified identifiers used to denote type names in each of the components of the fields.

identifiers-used-in-variants : *Variant* → *Qualident*-set

identifiers-used-in-variants (*variants*) ≜
  ⋃ {*identifiers-used-in-a-variant* (*variant*) | *variant* ∈ elems *variants*} 88

annotations The result is the set of qualified identifiers used in variants; these are the qualified identifiers used in each of the components of the variants.

identifiers-used-in-a-variant : *Variant* → *Qualident*-set

identifiers-used-in-a-variant (*variant*) ≜
  let *mk-Variant* (*labels*, *fields*) = *variant* in
  identifiers-used-in-expression (*labels*) ∪ *identifiers-used-in-fields-list* (*fields*) 84 87

annotations The result is the set of qualified identifiers used in a variant; these are the qualified identifiers used in the labels component and fields component of the variant.

6.2.13.7 Qualified Identifiers Used in Parameters

identifiers-used-in-parameters : *Formal-parameter-type-list* → *Qualident*-set

identifiers-used-in-parameters (*parms*) ≜
  ⋃ {*identifiers-used-in-a-parameter* (*parm*) | *parm* ∈ elems *parms*} 88

annotations The result is the set of qualified identifiers used in parameters; these are the qualified identifiers used in each of the components of the parameters.

identifiers-used-in-a-parameter : *Formal-parameter-type* → *Qualident*-set

identifiers-used-in-a-parameter (*parm*) ≜
  cases *parm* :
    mk-Value-formal-type (*ftype*) → *identifiers-used-in-formal-type* (*ftype*), 88
    mk-Variable-formal-type (*ftype*) → *identifiers-used-in-formal-type* (*ftype*) 88
  end

annotations The result is the set of qualified identifiers used in a parameter; these are the qualified identifiers used to denote type names in the formal type of the parameter.

identifiers-used-in-formal-type : *Formal-type* → *Qualident*-set

identifiers-used-in-formal-type (*fptype*) ≜
  cases *fptype* :
    mk-Parameter-formal-type (*type*) → *identifiers-used-in-type* (*type*), 87
    mk-Array-formal-type (*type*) → *identifiers-used-in-array-formal-type* (*type*) 88
  end

annotations The result is the set of qualified identifiers used in a formal type; these are the qualified identifiers used to denote type names in either a parameter formal type or an array formal type.

identifiers-used-in-array-formal-type : *Array-formal-type* → *Qualident*-set

identifiers-used-in-array-formal-type (*ftype*) ≜
  cases *ftype* :
    mk-Type-identifier (*qid*) → {*qid*},
    mk-Array-formal-type (*type*) → *identifiers-used-in-array-formal-type* (*type*) 88
  end

annotations The result is the set of qualified identifiers used in a formal type; these are the qualified identifiers used to denote type names in an array formal type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.2.13.8 Shielded Environments and Shielded Identifiers

build-shielded-environment : *Declaration* → *Environment*

build-shielded-environment (*decl*) ≜
  let *sh* = *shielded* (*decl*) in 89
  merge-environments ({*build-a-shielded-env* (*qid*) | *qid* ∈ *sh*}) 319 89

build-a-shielded-env : *Qualident* → *Environment*

build-a-shielded-env (*qid*) ≜
  if *qid* ∈ *Identifier*
  then μ (ρ*empty*, *types* ↦ {*id* ↦ nil })
  else let *mk-Qualident* (*id*, *qual*) = *qid* in
    let *sub* = *build-a-shielded-env* (*qual*) in 89
    μ (ρ*empty*, *mods* ↦ {*id* ↦ *mk-Unqualified-environment* (*sub*)})

shielded : *Declaration* → *Qualident*-set

shielded (*decl*) ≜
  cases *decl* :
    mk-Type-declaration (-, *type*) → *shielded-in-type* (*type*), 89
    mk-Variable-declaration (-, *type*) → *shielded-in-type* (*type*), 89
    others → { }
  end

annotations The result of this operation is a set consisting of those qualified identifiers that are shielded in a declaration. An identifier is shielded if it is used within a new pointer type or a new procedure type.

shielded-in-type : *Type* → *Qualident*-set

shielded-in-type (*type*) ≜
  cases *type* :
    mk-Type-identifier (-) → { },
    mk-Enumeration-type (-) → { },
    mk-Subrange-type (-, -) → { },
    mk-Set-type (*btype*) → { },
    mk-Pointer-type (*btype*) → *identifiers-used-in-type* (*btype*) ∪ *shielded-in-type* (*btype*), 87 89
    mk-Proper-procedure-type (*parms*) → *identifiers-used-in-parameters* (*parms*), 88
    mk-Function-procedure-type (*parms*, *return*) → *identifiers-used-in-parameters* (*parms*) ∪ {*return*}, 88
    mk-Array-type (*itype*, *ctype*) → *shielded-in-type* (*ctype*), 89
    mk-Record-type (*fields*) → *shielded-in-fields-list* (*fields*) 89
  end

annotations Return a set of qualified identifiers consisting of all the identifiers that are shielded in a type component of a definition.

shielded-in-fields-list : *Fields-list* → *Qualident*-set

shielded-in-fields-list (*fields*) ≜
  if *fields* = nil
  then { }
  else ⋃ {*shielded-in-fields* (*field*) | *field* ∈ elems *fields*} 89

annotations Return the set of qualified identifiers shielded in a fields list; these are the identifiers that are shielded in each of the components of the field list.

shielded-in-fields : *Fields* → *Qualident*-set-set

shielded-in-fields (*field*) ≜
  cases *field* :
    mk-Fixed-fields (-, *type*) → *shielded-in-type* (*type*), 89
    mk-Variant-fields (-, -, *variants*, *elsep*) → *shielded-in-variants* (*variants*) ∪ *shielded-in-fields-list* (*elsep*) 90 89
  end

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

annotations Return the set of qualified identifiers shielded in fields.

shielded-in-variants : *Variant** → *Qualident*-set

shielded-in-variants (*variants*) ≜
  ⋃ {*shielded-in-a-variant* (*variant*) | *variant* ∈ elems *variants*} 90

annotations Return the set of qualified identifiers shielded in variants.

shielded-in-a-variant : *Variant* → *Qualident*-set

shielded-in-a-variant (*variant*) ≜
  let *mk-Variant* (-, *fields*) = *variant* in
  shielded-in-fields-list (*fields*) 89

annotations Return the set of qualified identifiers shielded in a variant.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.3 Data Types

Every value and every variable has a type. The type of a value determines the operations that may be applied to that value — see 6.4.1 and 6.9.3.1. The type of a variable determines the set of values that may be assigned to that variable and determines the other operations that may be applied to that variable — see 6.4.2 and 6.9.3.2.

Some types, such as the number types, are basic to the language. For every basic type, the language provides a set of fundamental operations that may be performed on values of the type. Other types are constructed, using mechanisms provided by the language, as enumeration, subrange, opaque, set, pointer, procedure, array, and record types.

A distinction is made between elementary types and structured (composite) types. Array types and record types are structured types in the sense that values or variables of such types are compositions of other constituent values or variables which may be denoted as such. Values or variables of an elementary type have no constituents that may be denoted as components.

NOTE — The term structured type has been adopted from *Programming in Modula-2*. However, constructed types other than array types and record types also have a structure. For example, procedure types have a specified number of parameters of specified types. Where there might be ambiguity, array and record types are also referred to as composite types in this part of ISO/IEC 10514.

Ordinal types are types for which the permitted values form a discrete set for which there is a complete ordering; ordinal types are therefore elementary types. Of the basic types, the Boolean, character, unsigned, signed, and $\mathbb{Z}$ types are all ordinal types. Of the constructed types, enumeration types and subrange types are ordinal types. The ordinal number of a value of an ordinal type may be ascertained using the standard function procedure `ORD` — see 6.10.4.15.

6.3.1 Type Denoters and Ordinal Type Denoters

Type denoters are used to refer to existing identified types or to new types. The form of type denoter that is used to refer to an identified type is a type identifier. The identified type must either have a predefined identifier or an identifier must be introduced in a type definition or a type declaration — see 6.10.2, section 7, 6.2.4 and 6.2.9.

The form of type denoter that is used to refer to a new type has the effect of establishing a programmer-defined type that is distinct from any other type. This unique type will only become an identified type if the new type is being used as the type denoter in a type definition or a type declaration; otherwise the unique type is an anonymous type.

Apart from their use as type denoters in type declarations, new types may be used as type denoters in variable declarations. In addition to their use as type denoters in type declarations and variable declarations, type identifiers may be used to denote formal parameter types in procedure headings and in procedure types — see 6.2.5 and 6.3.9.

There are constructs in the language which require an ordinal type to be denoted rather than a general type. For example, the base type of a set type and the index type of an array type must be ordinal types. The concrete syntax uses the category 'ordinal type denoter' to indicate that a denoter of some ordinal type is expected. The form of ordinal type denoter that is used to refer to an identified ordinal type is an ordinal type identifier. The form of ordinal type denoter that is used to establish an ordinal type that is distinct from any other type is a new ordinal type. Enumeration types and subrange types are the two forms of new ordinal type.

Concrete Syntax

type denoter = type identifier | new type ;

ordinal type denoter = ordinal type identifier | new ordinal type ;

Abstract Syntax

types

Type = *Type-identifier* | *New-type*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations In the translation from concrete to abstract syntax a value of type *Type-identifier* is used if the concrete syntax contains an ordinal type identifier, and a value of type *New-type* is used if the concrete syntax contains a new ordinal type.

Declaration Semantics

d-type : *Type* → *Environment* → (*Typed* × *Environment*)

d-type (*type*)ρ ≙
  cases *type* :
    mk-Type-identifier (-) → *d-type-identifier* (*type*)ρ, 93
    mk-New-type (-) → *d-new-type* (*type*)ρ 93
  end

annotations The functions specifying the declaration semantics translate a new type into a type structure in which any new type component of the new type has been replaced by a unique type name (a 'hidden' name that cannot be generated in source code). Thus the 'new types' defined by some type declarations and variable declarations are each denoted by a name that is unique with respect to the complete program. Any type identifier occurring within a new type is replaced by its corresponding unique type name. The unique type name is associated with its structure by a mapping that is a component of the environment. Type identifiers are associated with their unique type names by a mapping that is another component of the environment. Thus an identifier denoting a type is mapped by a component of the environment into a unique type name, and that type name is mapped by a component of the environment into the structure of that type.

Static Semantics

wf-type : *Type* → *Environment* → $\mathbb{B}$

wf-type (*type*)ρ ≙
  cases *type* :
    mk-Type-identifier (-) → *wf-type-identifier* (*type*)ρ, 93
    mk-New-type (-) → *wf-new-type* (*type*)ρ 94
  end

6.3.2 Type Identifiers

A type identifier denotes an identified type that is predefined or is defined elsewhere in a type definition or a type declaration — see 6.2.4 and 6.2.9. There are some predefined types for which there are predefined type identifiers that are are pervasive — see 6.10.2. The pervasive identifiers for the basic types are `BOOLEAN`, for the Boolean type; `CARDINAL`, for the unsigned type; `CHAR`, for the character type; `COMPLEX`, for the complex type; `INTEGER`, for the signed type; `LONGCOMPLEX`, for the long complex type; `LONGREAL`, for the long real type; and `REAL`, for the real type. The pervasive identifiers for the predefined constructed types are `BITSET`, for the bitset type (which is a packedset type); and `PROC`, for the proc type (which is a procedure type). `PROTECTION` is a pervasive identifier for the implementation-defined protection type — see 6.10.2.11.

All of the types with pervasive identifiers are elementary types.

Concrete Syntax

type identifier = qualified identifier ;

ordinal type identifier = type identifier ;

Abstract Syntax

types

Type-identifier :: *qid* : *Qualident*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Declaration Semantics

A type identifier shall denote the unique type that is associated with the qualified identifier.

d-type-identifier : *Type-identifier* → *Environment* → (*Typed* × *Environment*)
d-type-identifier (*mk-Type-identifier* (*qid*))ρ $\triangleq$
  let *typed* = *type-of* (*qid*)ρ in 314
  mk- (*typed*, ρ)

Static Semantics

The qualified identifier shall denote a type that is predefined or is defined elsewhere in a type definition or a type declaration — see 6.2.4 and 6.2.9.

wf-type-identifier : *Type-identifier* → *Environment* → $\mathbb{B}$
wf-type-identifier (*mk-Type-identifier* (*qid*))ρ $\triangleq$
  wf-qualident (*qid*)ρ ∧ 53
  is-type (*qid*)ρ 314

6.3.3 New Types

A new type denotes a unique programmer-defined type and specifies the structure of values and variables of that type.

Concrete Syntax

```
new type =
    new ordinal type    | set type        | packedset type   | pointer type     |
    procedure type      | array type      | record type ;

new ordinal type = enumeration type | subrange type ;
```

Abstract Syntax

types

New-type = *Enumeration-type*
  | *Subrange-type*
  | *Set-type*
  | *Packed-set-type*
  | *Pointer-type*
  | *Procedure-type*
  | *Array-type*
  | *Record-type*

Declaration Semantics

Each use of a new type shall be a defining occurrence for a type that is distinct from any other type and that has the structure specified by the new type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

d-new-type : *New-type* → *Environment* → (*Typed* × *Environment*)

d-new-type (*type*)ρ ≜
  cases *type* :
    mk-Enumeration-type (-) → *d-enumeration-type* (*type*)ρ, 94
    mk-Subrange-type (-) → *d-subrange-type* (*type*)ρ, 96
    mk-Set-type (-) → *d-set-type* (*type*)ρ, 97
    mk-Packed-set-type (-) → *d-packed-set-type* (*type*)ρ, 98
    mk-Pointer-type (-) → *d-pointer-type* (*type*)ρ, 99
    mk-Procedure-type (-) → *d-procedure-type* (*type*)ρ, 100
    mk-Array-type (-, -) → *d-array-type* (*type*)ρ, 107
    mk-Record-type (-) → *d-record-type* (*type*)ρ 108
  end

Static Semantics

wf-new-type : *New-type* → *Environment* → $\mathbb{B}$

wf-new-type (*type*)ρ ≜
  cases *type* :
    mk-Enumeration-type (-) → *wf-enumeration-type* (*type*)ρ, 95
    mk-Subrange-type (-) → *wf-subrange-type* (*type*)ρ, 96
    mk-Set-type (-) → *wf-set-type* (*type*)ρ, 98
    mk-Packed-set-type (-) → *wf-packed-set-type* (*type*)ρ, 99
    mk-Pointer-type (-) → *wf-pointer-type* (*type*)ρ, 100
    mk-Procedure-type (-) → *wf-procedure-type* (*type*)ρ, 100
    mk-Array-type (-, -) → *wf-array-type* (*type*)ρ, 107
    mk-Record-type (-) → *wf-record-type* (*type*)ρ 108
  end

6.3.4 Enumeration Types

An enumeration type defines a new ordinal type, whose sequence of values is denoted by the sequence of identifiers contained in its identifier list. The identifiers of an enumeration type are introduced into their scope as constant identifiers of the values of the new type.

Concrete Syntax

enumeration type = left parenthesis, identifier list, right parenthesis ;

identifier list = identifier, { comma, identifier } ;

Abstract Syntax

types

Enumeration-type :: *values* : *Identifier*$^+$

Declaration Semantics

Each identifier occurring in the identifier list of an enumeration type shall be defined as a constant identifier that is associated with the corresponding value of the unique type established for the new type.

The type established by an enumeration type shall be an ordinal type. The ordering of the values of an enumeration type shall be determined by the textual order in which the identifiers of the identifier list are written. The ordinal number of the value associated with the ith identifier of the list shall be the value $(i - 1)$

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

d-enumeration-type : *Enumeration-type* → *Environment* → (*Type-name* × *Environment*)

d-enumeration-type (*mk-Enumeration-type* (*values*))ρ ≙
  let *typenm* = *generate-type-name* (ρ) in 320
  let ρ_{const} = *add-to-constants* (*typenm*, *values*)ρ in 95
  let *enum* = *mk-Enumeration-structure* (*values*) in
  mk- (*typenm*, *overwrite-struc-environment* ({*typenm* ↦ *enum*})ρ_{const}) 315

annotations A unique type name is generated to denote the new type that is being defined. The identifiers occurring in the identifier list are added to the environment as constants, each constant being associated with the value it denotes. The structure of the new type is also added to the environment.

Static Semantics

The identifiers occurring in the identifier list of an enumeration type shall be distinct from each other.

wf-enumeration-type : *Enumeration-type* → *Environment* → 𝔹

wf-enumeration-type (*mk-Enumeration-type* (*values*))ρ ≙
  is-uniques (*values*) 614

Auxiliaries

add-to-constants : *Type-name* × *Identifier*$^+$ → *Environment* → *Environment*

add-to-constants (*typen*, *values*)ρ ≙
  let *consts* = {*id* ↦ *mk-Constant-value* (*typen*, *mk-Enumeration-value* (*id*, *values*)) | *id* ∈ elems *values*} in
  overwrite-const-environment (*consts*)ρ 314

6.3.5 Subrange Types

A subrange type defines a new ordinal type, whose sequence of values is a subsequence of those defined for an associated ordinal type, called the 'host type' of the subrange type. A subrange type specifies the smallest and the largest values of the ordinal type; a value to be assigned to a variable of a subrange type must lie within the closed interval defined by these two values. The operations that are applicable to operands of a subrange type are those that are applicable to operands of its host type.

The host type of a subrange type is determined by the type associated with the range of values from the smallest to the largest value. There are rules for choosing a default type for this range in cases where there are alternatives. For example, if the smallest and largest values are both specified by positive whole number literals, the type of the range is assumed to be the unsigned type and not the signed type. Alternatively, a range type may be explicitly specified with the subrange type.

CHANGE — In *Programming in Modula-2* the term 'base type' is used for the associated ordinal type of both subrange types and set types. In this part of ISO/IEC 10514, the term 'base type' is only used for set types, and the term 'host type' is used for subrange types.

Concrete Syntax

subrange type = [range type], left bracket, constant expression, ellipsis, constant expression, right bracket ;

range type = ordinal type identifier ;

CLARIFICATION — A range type may be specified in the definition of a subrange by providing an identifier denoting a previously defined ordinal type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Abstract Syntax

types

Subrange-type :: *rtype* : *Qualident*
range : *Interval*

annotations If the range type of a subrange type is not explicitly specified, the translation process (from concrete to abstract syntax) supplies a range type according to the table given under Static Semantics below.

Declaration Semantics

The values of a subrange type shall be the values of the range type that are in the closed interval defined by the 'lower bound' and 'upper bound' of the subrange type. These values shall be specified by the values of the first and second constant expressions, respectively.

If a type `X` is a subrange type, then its host type shall be the type that is the host type of the range type of the type `X`; otherwise the host type shall be the type `X`.

The type established by a subrange type shall be an ordinal type. The ordering of the values of a subrange type shall be determined by the ordering of the values of the host type. The ordinal number of a value of a subrange type shall be the same as the ordinal number which the value has in the host type.

d-subrange-type : *Subrange-type* → *Environment* → (*Type-name* × *Environment*)
d-subrange-type (*mk-Subrange-type* (*rtype*, *range*))ρ ≜
  let *rtypen* = *type-of* (*rtype*)ρ in 314
  let *values* = *evaluate-constant-interval* (*range*)ρ in 257
  let *sub* = *mk-Subrange-structure* (*rtypen*, *values*) in
  let *typenm* = *generate-type-name* (ρ) in 320
  mk- (*typenm*, *overwrite-struc-environment* ({*typenm* ↦ *sub*})ρ) 315

annotations The range type of the subrange is determined, and the range is evaluated. A unique type name is generated to denote the new type that is being defined, and the structure of the new type is added to the environment.

Static Semantics

The range type of a subrange type shall be an ordinal type. If the range type is explicitly specified by an ordinal type identifier then this type identifier shall be predefined or have a defining occurrence that appears elsewhere; otherwise the range type of the subrange type `[M..N]` shall be determined according to the following table. In the table the type `T` denotes any ordinal type other than a whole number type.

		N			
		unsigned type	signed type	ℤ-type	the type T
M	unsigned type	unsigned type	error	unsigned type	error
	signed type	error	signed type	signed type	error
	ℤ-type	unsigned type	signed type	M<0 signed type M≥0 unsigned type	error
	the type T	error	error	error	the type T

Both the type of the first constant expression and the type of the second constant expression shall be expression-compatible with the range type. The lower bound shall be less than or equal to the upper bound, and the bounds shall both be values of the range type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wf-subrange-type : *Subrange-type* → *Environment* → 𝔹

wf-subrange-type (*mk-Subrange-type* (*rtype*, *range*))ρ ≙
  wf-qualident (*rtype*)ρ ∧ 53
  is-ordinal-type (*rtype*)ρ ∧ 322
  wf-interval (*range*)ρ ∧ 245
  is-constant-interval (*range*)ρ ∧ 257
  is-expression-compatible (*rtype*, *t-interval* (*range*)ρ)ρ ∧ 121 246
  let *cvalues* = *evaluate-constant-interval* (*range*)ρ in 257
  cvalues ≠ { } ∧
  cvalues ⊆ *values-of* (*rtype*)ρ 329

6.3.6 Set Types

A set type defines a new elementary type whose set of values is the power set of an associated ordinal type, called the 'base type' of the set type. Thus each value of a set type is a set whose elements are values of the base type of the set type.

CHANGES

1 In *Programming in Modula-2* the base type cannot have values that are less than zero.

2 This part of ISO/IEC 10514 does not require a specific representation for set types. In *Programming in Modula-2* there is some advice to the implementor to restrict the number of values of the base type, so that a value of a set type can be fitted into a small number of words of memory. In this part of ISO/IEC 10514, this advice is appropriate for packedset types rather than for set types.

3 In *Programming in Modula-2* the pervasive identifier `BITSET` is the type identifier of a predefined set type, whereas in this part of ISO/IEC 10514, it is the type identifier of a predefined packedset type.

Concrete Syntax

set type = "SET", "OF", base type ;

base type = ordinal type denoter ;

Abstract Syntax

types

Set-type :: *btype* : *Type*

Declaration Semantics

A new type that is a set type shall define the denoted base type as the base type of the unique set type that is established by the new type. The values of the set type shall be the elements of the power set of the base type of the set type.

d-set-type : *Set-type* → *Environment* → (*Type-name* × *Environment*)

d-set-type (*mk-Set-type* (*btype*))ρ ≙
  let *mk-* (*btypen*, ρ_{tenv}) = *d-type* (*btype*)ρ in 92
  let *set* = *mk-Set-structure* (*btypen*) in
  let *typenm* = *generate-type-name* (ρ_{tenv}) in 320
  mk- (*typenm*, *overwrite-struc-environment* ({*typenm* ↦ *set*})ρ_{tenv}) 315

annotations The base type of the set type is elaborated; a unique type name is generated to denote the new type that is being defined; and the structure of the new type is added to the environment.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Static Semantics

The type denoted by the base type of a set type shall be an ordinal type. If the base type is denoted by an ordinal type identifier, then this type identifier shall be predefined or have a defining occurrence that appears elsewhere; otherwise the base type shall be denoted by a new ordinal type.

NOTE — An implementation may place a restriction on the number of values of the base type of a set type — see 4.11

EXAMPLE — Declaration of set types.

All of the following are valid declarations of set types provided that both of the bounds of each subrange type lie within the range of values of the signed type:

```
TYPE
  s1 = SET OF INTEGER [0 .. N - 1] ;
  s2 = SET OF INTEGER [1 .. N] ;
  s3 = SET OF INTEGER [N + 1 .. 2*N] ;
  s4 = SET OF INTEGER [-2*N + 1 .. -N] ;
```

$wf\text{-}set\text{-}type : Set\text{-}type \rightarrow Environment \rightarrow \mathbb{B}$

$wf\text{-}set\text{-}type\,(mk\text{-}Set\text{-}type\,(btype))\rho \triangleq$
$\quad wf\text{-}type\,(btype)\rho \wedge$ 92
$\quad is\text{-}ordinal\text{-}type\,(btype)\rho$ 322

6.3.7 Packedset Types

A packedset type defines a new elementary type whose set of values is the power set of an associated ordinal type, called the 'base type' of the packedset type. Thus each value of a packedset type is a set whose elements are values of the base type of the packedset type.

Packedset types have a representation that can be mapped, in a known way, to the individual bits of a machine register for a particular underlying architecture — see 7.1.7.1. This leads to the ability to implement operations on values of a packedset type in a very efficient manner.

NOTE 1 — The pervasive identifier `BITSET` is the type identifier of a predefined packedset type — see 6.10.2.9.

CHANGE — Packedset types are not explicitly described in *Programming in Modula-2*, where all set types are implicitly packedset types.

Concrete Syntax

```
packedset type = "PACKEDSET", "OF", base type ;
```

Abstract Syntax

types

Packed-set-type :: *btype* : *Type*

Declaration Semantics

A new type that is a packedset type shall define the denoted base type as the base type of the unique packedset type that is established by the new type. The values of the packedset type shall be the elements of the power set of the base type of the packedset type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

d-packed-set-type : *Packed-set-type* → *Environment* → (*Type-name* × *Environment*)

d-packed-set-type (*mk-Packed-set-type* (*btype*))ρ ≜
  let *mk-* (*btypen*, ρ_{tenv}) = *d-type* (*btype*)ρ in 92
  let *set* = *mk-Packed-set-structure* (*btypen*) in
  let *typenm* = *generate-type-name* (ρ_{tenv}) in 320
  mk- (*typenm*, *overwrite-struc-environment* ({*typenm* ↦ *set*})ρ_{tenv}) 315

annotations The base type of the packedset type is elaborated; a unique type name is generated to denote the new type that is being defined; and the structure of the new type is added to the environment.

Static Semantics

The type denoted by the base type of a packedset type shall be an ordinal type. If the base type is denoted by an ordinal type identifier, then this type identifier shall be predefined or have a defining occurrence that appears elsewhere; otherwise the base type shall be denoted by a new ordinal type.

NOTES

2 Since packedset types are intended to mirror the underlying architecture, an implementation may impose narrow restrictions on the range of values of the base type of a packedset type — see 4.11.

3 The representation of packedset types is defined by the semantics of bit-level operations on packedset values which are available by import from the system module SYSTEM — see 7.1.7

wf-packed-set-type : *Packed-set-type* → *Environment* → $\mathbb{B}$

wf-packed-set-type (*mk-Packed-set-type* (*btype*))ρ ≜
  wf-type (*btype*)ρ ∧ 92
  is-ordinal-type (*btype*)ρ 322

6.3.8 Pointer Types

A pointer type defines a new elementary type. One of the values of every pointer type is called the nil value, and is denoted by the pervasive identifier NIL. The values of a pointer type other than the nil value can be used to designate variables of a type called the bound type of the pointer type. A value of a pointer type is frequently obtained by a call to an allocation procedure exported from a storage management module.

NOTE 1 — The type of the nil value is an anonymous type called the 'nil type'. All pointer types are assignment compatible (and hence value-parameter-compatible) with the nil type — see 6.4.2 and 6.9.3.1. All pointer types are expression-compatible with the nil type in pointer relational operations — see 6.8.2.7.5.

Concrete Syntax

pointer type = "POINTER", "TO", bound type ;

bound type = type denoter ;

Abstract Syntax

types

Pointer-type :: *btype* : *Type*

Declaration Semantics

A new type that is a pointer type shall define the denoted bound type as the bound type of the unique pointer type that is established by the new type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

d-pointer-type : *Pointer-type* → *Environment* → (*Type-name* × *Environment*)

d-pointer-type (*mk-Pointer-type* (*btype*))ρ ≜
  let *mk-* (*typen*, ρ_{type}) = *d-type* (*btype*)ρ in 92
  let *point* = *mk-Pointer-structure* (*typen*) in
  let *typenm* = *generate-type-name* (ρ_{type}) in 320
  mk- (*typenm*, *overwrite-struc-environment* ({*typenm* ↦ *point*})ρ_{type}) 315

annotations The bound type is elaborated; a unique type name is generated to denote the new type being defined; and the structure of the new type is added to the environment.

Static Semantics

If the bound type of a pointer type is specified by a type identifier, then this type identifier shall be predefined or have a defining occurrence that appears elsewhere; otherwise the bound type shall be specified by a new type.

NOTE 2 — The declaration of a pointer type may precede the declaration of its bound type — see 6.2.6.

wf-pointer-type : *Pointer-type* → *Environment* → $\mathbb{B}$

wf-pointer-type (*mk-Pointer-type* (*btype*))ρ ≜
  wf-type (*btype*)ρ 92

6.3.9 Procedure Types

A procedure type may be either a proper procedure type or a function procedure type.

Concrete Syntax

procedure type = proper procedure type | function procedure type ;

Abstract Syntax

types

Procedure-type = *Proper-procedure-type* | *Function-procedure-type*

Declaration Semantics

d-procedure-type : *Type* → *Environment* → (*Typed* × *Environment*)

d-procedure-type (*type*)ρ ≜
  cases *type* :
    mk-Proper-procedure-type (-) → *d-proper-procedure-type* (*type*)ρ, 101
    mk-Function-procedure-type (-) → *d-function-procedure-type* (*type*)ρ 102
  end

Static Semantics

wf-procedure-type : *Type* → *Environment* → $\mathbb{B}$

wf-procedure-type (*type*)ρ ≜
  cases *type* :
    mk-Proper-procedure-type (-) → *wf-proper-procedure-type* (*type*)ρ, 101
    mk-Function-procedure-type (-) → *wf-function-procedure-type* (*type*)ρ 102
  end

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.3.9.1 Proper Procedure Types

A proper procedure type defines a new elementary type. Each value of the type is a proper procedure, with a declaration level of 0, and with formal parameters that correspond with those defined in the formal parameter type list of the type.

Concrete Syntax

proper procedure type = "PROCEDURE", [left parenthesis, [formal parameter type list], right parenthesis] ;

Abstract Syntax

types

Proper-procedure-type :: *parms* : [*Formal-parameter-type-list*]

Declaration Semantics

A new type that is a proper procedure type shall establish a unique procedure type that has the associated structure of a proper procedure with the number, positions, formal types, and binding of formal parameters specified by the formal parameter type list.

d-proper-procedure-type : *Proper-procedure-type* → *Environment* → (*Type-name* × *Environment*)
d-proper-procedure-type (*mk-Proper-procedure-type* (*parms*))ρ ≜
  let *nparms* = if *parms* ≠ nil
    then *d-formal-parameter-type-list* (*parms*)ρ else [] in 103
  let *strucc* = *mk-Proper-procedure-structure* (*nparms*) in
  let *typenm* = *generate-type-name* (ρ) in 320
  mk- (*typenm*, *overwrite-struc-environment* ({*typenm* ↦ *struc*})ρ) 315

annotations The type of each of the parameters is elaborated; a unique type name is generated to denote the new type that is being defined; and the structure of the new type is added to the environment.

Static Semantics

wf-proper-procedure-type : *Proper-procedure-type* → *Environment* → $\mathbb{B}$
wf-proper-procedure-type (*mk-Proper-procedure-type* (*parms*))ρ ≜
  ∀ *parm* ∈ elems *parms* · *wf-formal-parameter-type* (*parm*)ρ 103

6.3.9.2 Function Procedure Types

A function procedure type defines a new elementary type. Each value of the type is a function procedure, with a declaration level of 0, with formal parameters that correspond with those defined in the formal parameter type list of the type, and with a result type that is identical to the function result type of the type.

Concrete Syntax

function procedure type =
  "PROCEDURE", left parenthesis, [formal parameter type list], right parenthesis,
  colon, function result type ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Abstract Syntax

types

Function-procedure-type :: *parms* : [*Formal-parameter-type-list*]
result : *Type-identifier*

Declaration Semantics

A new type that is a function procedure type shall establish a unique procedure type that has the associated structure of a function procedure with the number, positions, formal types, and binding of formal parameters specified by the formal parameter type list, and with the result type that is denoted by the function result type of the new type.

d-function-procedure-type : *Function-procedure-type* → *Environment* → (*Type-name* × *Environment*)

d-function-procedure-type (*mk-Function-procedure-type* (*parms*, *result*))ρ ≜
  let *nparms* = if *parms* ≠ nil
    then *d-formal-parameter-type-list* (*parms*)ρ else [] in 103
  let *mk-* (*nreturn*, -) = *d-type* (*result*)ρ in 92
  let *struc* = *mk-Function-procedure-structure* (*nparms*, *nreturn*) in
  let *typenm* = *generate-type-name* (ρ) in 320
  mk- (*typenm*, *overwrite-struc-environment* ({*typenm* ↦ *struc*})ρ) 315

annotations The type of each of the parameters is elaborated; the type of the returned value is elaborated; a unique type name is generated to denote the new type that is being defined; and the structure of the new type is added to the environment.

Static Semantics

The function result type shall be specified by a type identifier that is predefined or has a defining occurrence that appears elsewhere.

CLARIFICATION — In this part of ISO/IEC 10514, the function result type of a function procedure type can be any elementary type or structured (composite) type. It may be identical to the function procedure type.

wf-function-procedure-type : *Function-procedure-type* → *Environment* → $\mathbb{B}$

wf-function-procedure-type (*mk-Function-procedure-type* (*parms*, *retrn*))ρ ≜
  wf-type (*retrn*)ρ ∧ 92
  ∀ *parm* ∈ elems *parms* · *wf-formal-parameter-type* (*parm*)ρ 103

6.3.9.3 Formal Parameter Type Lists

A formal parameter type list specifies the number of formal parameters of a procedure type, the formal type of each of these formal parameters, and whether each formal parameter is a value parameter or a variable parameter — see 6.2.5.3.

Concrete Syntax

formal parameter type list = formal parameter type, { comma, formal parameter type } ;

formal parameter type = variable formal type | value formal type ;

Abstract Syntax

types

Formal-parameter-type-list = *Formal-parameter-type*$^+$;

Formal-parameter-type = *Variable-formal-type* | *Value-formal-type*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Declaration Semantics

d-formal-parameter-type-list : *Formal-parameter-type-list* → *Environment* → *Formal-parameters-typed*
d-formal-parameter-type-list (*fps*)ρ ≜
  if *fps* = [ ]
  then [ ]
  else [*d-formal-parameter-type* (hd *fps*)ρ] ↷ *d-formal-parameter-type-list* (tl *fps*)ρ 103 103

d-formal-parameter-type : *Formal-parameter-type* → *Environment* → (*Typed* × *Environment*)
d-formal-parameter-type (*type*)ρ ≜
  cases *type* :
    mk-Variable-formal-type (-) → *d-variable-formal-type* (*type*)ρ, 103
    mk-Value-formal-type (-) → *d-value-formal-type* (*type*)ρ 104
  end

Static Semantics

wf-formal-parameter-type : *Formal-parameter-type* → *Environment* → $\mathbb{B}$
wf-formal-parameter-type (*type*)ρ ≜
  cases *type* :
    mk-Variable-formal-type (-) → *wf-variable-formal-type* (*type*)ρ, 103
    mk-Value-formal-type (-) → *wf-value-formal-type* (*type*)ρ 104
  end

6.3.9.4 Variable Formal Types

A variable formal parameter type defines the formal type of a variable formal parameter of a procedure type.

Concrete Syntax

variable formal type = "VAR", formal type ;

Abstract Syntax

Variable-formal-type :: *ftype* : *Formal-type*

Declaration Semantics

A variable formal type shall define the corresponding formal parameter of the procedure type to be a variable formal parameter of the declared formal type — see 6.3.10.

d-variable-formal-type : *Variable-formal-type* → *Environment* → *Variable-formal-typed*
d-variable-formal-type (*mk-Variable-formal-type* (*ftype*))ρ ≜
  mk-Variable-formal-typed (*d-formal-type* (*ftype*)ρ) 105

Static Semantics

wf-variable-formal-type : *Variable-formal-type* → *Environment* → $\mathbb{B}$
wf-variable-formal-type (*mk-Variable-formal-type* (*ftype*))ρ ≜
  wf-formal-type (*ftype*)ρ 105

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.3.9.5 Value Formal Types

A value formal parameter type defines the formal type of a value formal parameter of a procedure type.

Concrete Syntax

value formal type = formal type ;

Abstract Syntax

Value-formal-type :: *ftype* : *Formal-type*

Declaration Semantics

A value formal type shall define the corresponding formal parameter of the procedure type to be a value formal parameter of the declared formal type — see 6.3.10.

d-value-formal-type : *Value-formal-type* → *Environment* → *Value-formal-typed*

d-value-formal-type (*mk-Value-formal-type* (*ftype*))ρ $\triangleq$
  mk-Value-formal-typed (*d-formal-type* (*ftype*)ρ) 105

Static Semantics

wf-value-formal-type : *Value-formal-type* → *Environment* → $\mathbb{B}$

wf-value-formal-type (*mk-Value-formal-type* (*ftype*))ρ $\triangleq$
  wf-formal-type (*ftype*)ρ 105

6.3.10 Formal Types

Formal types are used in procedure headings and in procedure types to specify the structure of formal parameters. The structure of a formal parameter may either be a single identified type or an open array of an identified type or of another open array.

Concrete Syntax

formal type = type identifier | open array formal type ;

open array formal type = "ARRAY", "OF", open array component type ;

open array component type = formal type ;

CHANGE — This part of ISO/IEC 10514 permits multidimensional open array parameters.

Abstract Syntax

types

Formal-type = *Parameter-formal-type* | *Array-formal-type*;

Parameter-formal-type :: *type* : *Type-identifier*;

Array-formal-type :: *type* : *Open-component-type*;

Open-component-type = *Type-identifier* | *Array-formal-type*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Declaration Semantics

A formal type that is a type identifier shall define the structure of the corresponding formal parameter, or open array component type, to be the single type that is denoted by the type identifier.

An open array formal type shall define the structure of the corresponding formal parameter, or open array component type, to be an open array with a component type that has the structure defined by the open array component type of the open array formal type.

d-formal-type : *Formal-type* → *Environment* → *Variable-typed*

d-formal-type (*type*)ρ $\triangleq$
  cases *type* :
    mk-Parameter-formal-type (-) → *d-parameter-formal-type* (*type*)ρ, 105
    mk-Array-formal-type (-) → *d-array-formal-type* (*type*)ρ 105
  end

d-parameter-formal-type : *Parameter-formal-type* → *Environment* → *Variable-typed*

d-parameter-formal-type (*mk-Parameter-formal-type* (*type*))ρ $\triangleq$
  let *mk-* (*atype*, -) = *d-type-identifier* (*type*)ρ in 93
  atype

d-array-formal-type : *Array-formal-type* → *Environment* → *Variable-typed*

d-array-formal-type (*mk-Array-formal-type* (*type*))ρ $\triangleq$
  let *mk-* (*typen*, -) = *d-open-component-type* (*type*)ρ in 105
  mk-Open-array-typed (*typen*)

d-open-component-type : *Open-component-type* → *Environment* → (*Typed* × *Environment*

d-open-component-type (*type*)ρ $\triangleq$
  cases *type* :
    mk-Type-identifier (-) → *d-type-identifier* (*type*)ρ, 93
    mk-Array-formal-type (-) → let *vtyped* = *d-array-formal-type* (*type*)ρ in 105
      mk- (*vtyped*, ρ)
  end

Static Semantics

If a formal type is a type identifier, then this type identifier shall be predefined or have a defining occurrence that appears elsewhere.

CLARIFICATION — In this part of ISO/IEC 10514, the defining occurrence of a type identifier used as a formal type may be the definition or declaration of the procedure type in which that formal type occurs; the formal type of a procedure type may thus be of a type that is identical to the procedure type — see 6.2.9.

If a formal type is an open array formal type, then the type identifier of the open array formal type shall be predefined or have a defining occurrence that appears elsewhere.

wf-formal-type : *Formal-type* → *Environment* → $\mathbb{B}$

wf-formal-type (*type*)ρ $\triangleq$
  cases *type* :
    mk-Parameter-formal-type (-) → *wf-parameter-formal-type* (*type*)ρ, 105
    mk-Array-formal-type (-) → *wf-array-formal-type* (*type*)ρ 106
  end

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wf-parameter-formal-type : *Parameter-formal-type* → *Environment* → $\mathbb{B}$

wf-parameter-formal-type (*mk-Parameter-formal-type* (*type*))ρ $\triangleq$
  wf-type-identifier (*type*)ρ 93

wf-array-formal-type : *Array-formal-type* → *Environment* → $\mathbb{B}$

wf-array-formal-type (*mk-Array-formal-type* (*type*))ρ $\triangleq$
  wf-open-component-type (*type*)ρ 106

wf-open-component-type : *Open-component-type* → *Environment* → $\mathbb{B}$

wf-open-component-type (*type*)ρ $\triangleq$
  cases *type* :
    mk-Type-identifier (-) → *wf-type-identifier* (*type*)ρ, 93
    mk-Array-formal-type (-) → *wf-array-formal-type* (*type*)ρ 106
  end

6.3.11 Array Types

An array type defines a new structured (composite) type. An array type is a mapping from a set of values defined by an index type onto a set of distinct components. Each one of these components has the same type, called the component type of the array type.

Concrete Syntax

array type = "ARRAY", index type, { comma, index type }, "OF", component type ;

index type = ordinal type denoter ;

component type = type denoter ;

The concrete syntax allows for a full or abbreviated form of array type in cases where a component type is itself an array type. In the abbreviated form, a single comma shall replace the sequence '`OF ARRAY`' that occurs in the full form. The abbreviated form and the full form shall be equivalent.

EXAMPLE — Abbreviated form for array of array types.

An array type of the form:

```
ARRAY MonthType OF ARRAY [1 .. 31] OF CARDINAL
```

may be abbreviated to:

```
ARRAY MonthType, [1 .. 31] OF CARDINAL
```

Abstract Syntax

Array-type :: *itype* : *Type*
  ctype : *Type*

annotations The abstract syntax only uses the full form.

Declaration Semantics

If the index type ordinal type denoter of an array type (in its full form) is a new ordinal type, a corresponding new ordinal type shall be established for the index type; otherwise, if the index type ordinal type denoter is a type identifier, the index type shall be the corresponding identified type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

If the component type denoter of an array type (in its full form) is a new type, a corresponding new type shall be established for the component type; otherwise, if the component type denoter is a type identifier, the component type shall be the corresponding identified type.

The array type (denoter) shall establish a unique array type that has the specified index type and component type.

The type of each component of the array type shall be the component type of the array type.

$d\text{-}array\text{-}type : Array\text{-}type \to Environment \to (Type\text{-}name \times Environment)$

$d\text{-}array\text{-}type\,(mk\text{-}Array\text{-}type\,(itype, ctype))\rho \triangleq$
  let $mk\text{-}(tni, \rho_1) = d\text{-}type\,(itype)\rho$ in 92
  let $mk\text{-}(tnc, \rho_2) = d\text{-}type\,(ctype)\rho_1$ in 92
  let $array = mk\text{-}Array\text{-}structure\,(tni, tnc)$ in
  let $typenm = generate\text{-}type\text{-}name\,(\rho_2)$ in 320
  $mk\text{-}(typenm, overwrite\text{-}struc\text{-}environment\,(\{typenm \mapsto array\})\rho_2)$ 315

annotations The types of the index and components are elaborated; a unique type name is generated to denote the new type that is being defined; and the structure of the new type is added to the environment.

Static Semantics

The index type of an array type shall be an ordinal type. If the index type is specified by an ordinal type identifier, then this type identifier shall be predefined or have a defining occurrence that appears elsewhere; otherwise the index type shall be specified by a new type that is an enumeration type or a subrange type. If the component type of an array type is specified by a type identifier, then this type identifier shall be predefined or have a defining occurrence that appears elsewhere; otherwise the component type shall be specified by a new type.

NOTE — If the index type of an array type is an enumeration type, the enumeration constant identifiers that are thereby declared may be used in the component type of the array type.

$wf\text{-}array\text{-}type : Array\text{-}type \to Environment \to \mathbb{B}$

$wf\text{-}array\text{-}type\,(mk\text{-}Array\text{-}type\,(itype, ctype))\rho \triangleq$
  $is\text{-}ordinal\text{-}type\,(itype)\rho \wedge$ 322
  $wf\text{-}type\,(itype)\rho \wedge$ 92
  $wf\text{-}type\,(ctype)\rho$ 92

6.3.12 Record Types

A record type defines a new structured (composite) type. Each value of the type has a collection of components. These components, called the fields of the record type, may be of different types. The individual fields of a value or variable of a record type are selected by their names (called field identifiers). A record type may specify that some groups of fields (called fixed fields) are always to be present in values and variables of that type. A record type may specify that other groups of fields (called variant fields) include alternatives (called variants) whose presence within a value or variable of that type depends upon the value of another field (called the tag field). The variant that applies, given the value of the tag field, is called the 'active variant'.

Concrete Syntax

record type = "RECORD", field list, "END" ;

field list = fields, { semicolon, fields } ;

fields = [fixed fields | variant fields] ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Abstract Syntax

types

Record-type :: *fields* : *Field-list*;

Field-list = *Fields**;

Fields = *Fixed-fields* | *Variant-fields*

Declaration Semantics

A new type that is a record type shall establish a unique record type that has the structure of the group or groups of fields specified by the record type field list. Any enumeration constant identifiers introduced by enumeration types within a group of fields shall be defined for possible use within subsequent groups of fields.

d-record-type : *Record-type* → *Environment* → (*Type-name* × *Environment*)
d-record-type (*mk-Record-type* (*fields*))ρ ≙
  let *mk-* (*nfields*, ρ_1) = *build-fields-list-structure* (*fields*)ρ in 108
  let *struc* = *mk-Record-structure* (*nfields*) in
  let *typenm* = *generate-type-name* (ρ_1) in 320
  mk- (*typenm*, *overwrite-struc-environment* ({*typenm* ↦ *struc*})ρ_1) 315

annotations The type of each of the components of the record is elaborated; a unique type name is generated to denote the new type that is being defined; and the structure of the new type is added to the environment.

build-fields-list-structure : *Field-list* → *Environment* → (*Fields-list-structure* × *Environment*)
build-fields-list-structure (*fields*)ρ ≙
  if *fields* = [] ∨ *fields* = nil
  then *mk-* ([], ρ)
  else let *mk-* (*firstfield*, ρ_1) = *d-fields* (hd *fields*)ρ in 108
    let *mk-* (*remainder*, ρ_2) = *build-fields-list-structure* (tl *fields*)ρ_1 in 108
    mk- ([*firstfield*] ↷ *remainder*, ρ_2)

d-fields : *Fields* → *Environment* → (*Fields-structure* × *Environment*)
d-fields (*fields*)ρ ≙
  cases *fields* :
    mk-Fixed-fields (-, -) → *d-fixed-fields* (*fields*)ρ, 109
    mk-Variant-fields (-, -, -, -) → *d-variant-fields* (*fields*)ρ 110
  end

Static Semantics

All the field identifiers of a record type shall be distinct.

wf-record-type : *Record-type* → *Environment* → 𝔹
wf-record-type (*mk-Record-type* (*fields*))ρ ≙
  is-distinct-identifiers (*fields*) ∧ 113
  wf-field-list (*fields*)ρ 108

wf-field-list : *Fields** → *Environment* → 𝔹
wf-field-list (*fieldss*)ρ ≙
  ∀ *fields* ∈ elems *fieldss* · *wf-fields* (*fields*)ρ 108

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wf-fields : *Fields* → *Environment* → $\mathbb{B}$

wf-fields (*fields*)ρ $\triangleq$
  cases *fields* :
    mk-Fixed-fields (-, -) → *wf-fixed-fields* (*fields*)ρ, 109
    mk-Variant-fields (-, -, -, -) → *wf-variant-fields* (*fields*)ρ 111
  end

6.3.12.1 Fixed Fields

A group of fields specified as fixed fields of a record type share the same field type; the group shall always be present in values and variables of the record type, unless the group is part of an inactive variant.

Concrete Syntax

fixed fields = identifier list, colon, field type ;

field type = type denoter ;

Abstract Syntax

types

Fixed-fields :: *ids* : *Identifier*$^+$
  type : *Type*

Declaration Semantics

If the field type of a group of fixed fields is a new type, a corresponding new type shall be established which shall be the type of each field in the group of fixed fields; otherwise, if the field type is a type identifier, the type of each field shall be the corresponding identified type.

The identifiers in the identifier list of the group of fixed fields shall be defined as the field identifiers of the fixed fields.

d-fixed-fields : *Fixed-fields* → *Environment* → (*Fixed-fields-structure* × *Environment*)

d-fixed-fields (*mk-Fixed-fields* (*ids*, *type*))ρ $\triangleq$
  let *mk-* (tn, ρ_1) = *d-type* (*type*)ρ in 92
  mk- (*mk-Fixed-fields-structure* (*ids*, tn), ρ_1)

annotations The type of the fixed fields component of the record is elaborated; a unique type name is generated to denote any new type that may be defined; the result is the structure of the component.

Static Semantics

If the field type of a group of fixed fields is specified by a type identifier, this type identifier shall be predefined or have a defining occurrence that appears elsewhere; otherwise the field type shall be specified by a new type.

wf-fixed-fields : *Fixed-fields* → *Environment* → $\mathbb{B}$

wf-fixed-fields (*mk-Fixed-fields* (*ids*, *type*))ρ $\triangleq$
  wf-type (*type*)ρ 92

6.3.12.2 Variant Fields

A groups of fields specified as variant fields of a record type include alternatives (called variants) whose presence within a value or variable of that type depends upon the value of another field (called the tag field). The variant that applies, given the value of the tag field, is called the 'active variant'.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Concrete Syntax

variant fields = "CASE", tag field, "OF", variant list, "END" ;

tag field = [tag identifier], colon, tag type ;

tag identifier = identifier ;

tag type = ordinal type identifier ;

variant list = variant, { case separator, variant }, [variant else part] ;

variant else part = "ELSE", field list ;

Abstract Syntax

types

Variant-fields :: *tag* : [*Identifier*]
tagt : *Qualident*
variants : *Variant**
elsep : [*Field-list*]

Declaration Semantics

The tag type of a group of variant fields shall be the ordinal type denoted by the tag type ordinal type identifier. If present, the tag identifier shall be defined as the field identifier of the tag field; otherwise, the group of variant fields shall not have an identifiable tag field.

The variants of a group of variant fields shall have the structure specified by each variant of the variant list and, if present, the variant else part. Any enumeration constant identifiers introduced by enumeration types within a variant shall be defined for possible use within subsequent variants, including a variant else part.

The variant else part, if present, shall have the structure of the group or groups of fields specified by the variant else part field list.

d-variant-fields : *Variant-fields* → *Environment* → (*Variant-fields-structure* × *Environment*)
d-variant-fields (*mk-Variant-fields* (*tag*, *tagt*, *variants*, *elsep*))ρ ≜
  let *tagnt* = *type-of* (*tagt*)ρ in 314
  let *mk-* (*nvariants*, ρ_1) = *build-variant-structure* (*variants*)ρ in 110
  let *mk-* (*elseps*, ρ_2) = *build-fields-list-structure* (*elsep*)ρ_1 in 108
  mk- (*mk-Variant-fields-structure* (*tag*, *tagnt*, *nvariants*, *elseps*), ρ_2)

annotations The type of the tag field of the variant field is obtained; the structures of the variant components and of the optional variant else part list component are elaborated; the result is the structure of the component.

build-variant-structure : *Variant** → *Environment* → (*Variant-structure** × *Environment*)
build-variant-structure (*variants*)ρ ≜
  if *variants* = [ ]
  then *mk-* ([ ], ρ)
  else let *mk-* (*nhvariant*, ρ_1) = *d-variant* (hd *variants*)ρ in 112
    let *mk-* (*tnvariants*, ρ_2) = *build-variant-structure* (tl *variants*)ρ_1 in 110
    mk- ([*nhvariant*] ↷ *tnvariants*, ρ_2)

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Static Semantics

The tag type of a group of variant fields shall be an ordinal type. The tag type shall be specified by an ordinal type identifier that is predefined or has a defining occurrence that appears elsewhere. The types of the constant expressions of the variant label list of each variant shall be expression compatible with the tag type.

The values contained in different variant label lists of a variant shall be distinct from each other.

NOTE — In addition to the requirement that the values contained in different variant label lists of a variant shall be distinct from each other, it is also required that the values contained in each variant label list shall be distinct from each other.

EXAMPLE 1 — Duplication of variant labels in record declarations.

The alternatives illustrated in the following source code are illegal for the reasons given in the associated comments.

```
TYPE
  TagRange = [1..19];
RECORD
  CASE TagFieldIdentifier : TagRange OF
  | (*a*)  1           : (* duplicate of label in list d *)
  | (*b*)  11          : (* label 11 in b duplicated in range 11..13 in list i *)
  | (*c*)  15..17      : (* range 15..17 in c overlaps with range 16..19 in list j *)
  | (*d*)  1           : (* duplicate of label in list a *)
  | (*e*)  2, 2        : (* label 2 is duplicated in list e *)
  | (*f*)  3..5, 3..5  : (* range 3..5 is duplicated in list f *)
  | (*g*)  6, 6..7     : (* label 6 duplicated in range 6..7 in list g *)
  | (*h*)  8..9, 8..10 : (* range 8..9 overlaps range 8..10 in list h *)
  | (*i*)  11..13      : (* label 11 in b duplicated in range 11..13 in list i *)
  | (*j*)  16..19      : (* range 15..17 in c overlaps with range 16..19 in list j *)
  END
END
```

If there is no variant else part, there shall be a value contained in the variant label lists for each value of the tag type.

EXAMPLE 2 — Coverage of variant labels in record declarations.

The requirement that if there is no variant else part there shall be a value contained in the variant label lists for each value of the tag type means that example (a) is illegal, whereas examples (b) and (c) are legal.

```
(a)                  (b)                  (c)
RECORD               RECORD               RECORD
CASE : INTEGER OF    CASE : INTEGER OF    CASE : INTEGER OF
| 1: x: BOOLEAN     | 1: x: BOOLEAN      | 1: x: BOOLEAN
| 2: y: CARDINAL    | 2: y: CARDINAL     | 2: y: CARDINAL
END                  ELSE (*empty*)       | MIN(INTEGER)..0: (*empty*)
END                  END                  | 3..MAX(INTEGER): (*empty*)
                     END                  END
                                          END
```

$wf\text{-}variant\text{-}fields : Variant\text{-}fields \rightarrow Environment \rightarrow \mathbb{B}$

$wf\text{-}variant\text{-}fields\ (mk\text{-}Variant\text{-}fields\ (tag, tagt, variants, elsep))\rho \triangleq$

$wf\text{-}qualident\ (tagt)\rho \wedge$ 53

let $type = type\text{-}of\ (tagt)\rho$ in 314

$is\text{-}ordinal\text{-}type\ (type)\rho \wedge$ 322

$is\text{-}compatible\text{-}variant\text{-}labels\ (type, variants)\rho \wedge$ 114

$is\text{-}disjoint\text{-}variant\text{-}labels\ (variants)\rho \wedge$ 114

$\forall\ variant \in \text{elems}\ variants \cdot wf\text{-}variant\ (variant)\rho \wedge$ 112

$(elsep = \text{nil} \wedge is\text{-}complete\text{-}variant\text{-}labels\ (type, variants)\rho$ 114

$\vee$

$wf\text{-}field\text{-}list\ (elsep)\rho) \wedge$ 108

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Variants

A variant specifies the value or values of the tag field for which the variant is to be active, and the group or groups of fields that are present when the variant is active.

Concrete Syntax

variant = [variant label list, colon, field list] ;

variant label list = variant label, { comma, variant label } ;

variant label = constant expression, [ellipsis, constant expression] ;

Abstract Syntax

types

Variant :: *labels* : *Set-definition*
fields : *Field-list*;

Set-definition = *Member*-set

annotations The translation process (from concrete to abstract syntax) will ensure that the number of members in the *labels* component of a *Variant* component of the abstract representation is equal to the number of variant labels in the variant label list of the concrete representation.

Declaration Semantics

The values of the tag field for which the variant is to be active shall be the set of values contained in the variant label list. A value shall be contained in a variant label list if it is equal to the value of one of the constant expressions of the variant label list, or if the value lies within the range specified by a pair of constant expressions of the variant label list.

The group or groups of fields that are present when the variant is active shall be the those specified by the variant field list.

$d\text{-}variant : Variant \to Environment \to (Variant\text{-}structure \times Environment)$

$d\text{-}variant\,(mk\text{-}Variant\,(labels, fields))\rho \triangleq$
  let $nlabels = evaluate\text{-}constant\text{-}set\text{-}definition\,(labels)\rho$ in 257
  let $mk\text{-}(nfields, \rho_1) = build\text{-}fields\text{-}list\text{-}structure\,(fields)\rho$ in 108
  $mk\text{-}(mk\text{-}Variant\text{-}structure\,(nlabels, nfields), \rho_1)$

annotations The value of the labels that are associated with the variant are evaluated; the structures of the fields are elaborated; and the result is the structure of the component.

Static Semantics

Each variant label shall be a constant expression, or a range indicated by a pair of constant expressions. The values contained in a variant label list shall be distinct from each other.

$wf\text{-}variant : Variant \to Environment \to \mathbb{B}$

$wf\text{-}variant\,(mk\text{-}Variant\,(labels, fields))\rho \triangleq$
  $wf\text{-}set\text{-}definition\,(labels)\rho \wedge$ 243
  $is\text{-}disj\,(labels)\rho \wedge$ 614
  $is\text{-}constant\text{-}set\text{-}definition\,(labels)\rho \wedge$ 256
  $wf\text{-}field\text{-}list\,(fields)\rho$ 108

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Auxiliaries

is-distinct-identifiers : *Field-list* → 𝔹

is-distinct-identifiers (*fields*) ≜
  let *ids* = *identifiers-of-field-list* (*fields*) in 113
  is-uniques (*ids*) 614

annotations Check that all the field identifiers of a field list are distinct.

identifiers-of-field-list : *Field-list* → *Identifier**

identifiers-of-field-list (*fields*) ≜
  if *fields* = [ ]
  then [ ]
  else *identifiers-of-fields* (hd *fields*) ⌢ *identifiers-of-field-list* (tl *fields*) 113 113

annotations The result of this function is a sequence containing the field identifiers of a record type. The order of the identifiers in the sequence is identical to the order in which they occur in the record type.

identifiers-of-fields : *Fields* → *Identifier**

identifiers-of-fields (*field*) ≜
  cases *field* :
    mk-Fixed-fields (-, -) → *identifiers-of-fixed-fields* (*field*), 113
    mk-Variant-fields (-, -, -, -) → *identifiers-of-variant-fields* (*field*) 113
  end

annotations The result of this function is a sequence containing the field identifiers of a fixed fields or a variant fields. The order of the identifiers in the sequence is identical to the order in which they occur in the fixed fields or variant fields.

identifiers-of-fixed-fields : *Fixed-fields* → *Identifier**

identifiers-of-fixed-fields (*field*) ≜
  let *mk-Fixed-fields* (*ids*, -) = *field* in
  ids

identifiers-of-variant-fields : *Variant-fields* → *Identifier**

identifiers-of-variant-fields (*field*) ≜
  let *mk-Variant-fields* (*tag*, -, *variants*, *elsep*) = *field* in
  let *tagid* = if *tag* = nil then [ ] else [*tag*] in
  tagid ⌢ *identifiers-of-variants* (*variants*) ⌢ *identifiers-of-field-list* (*elsep*) 113 113

identifiers-of-variants : *Variant** → *Identifier**

identifiers-of-variants (*variants*) ≜
  if *variants* = [ ]
  then [ ]
  else *identifiers-of-a-variant* (hd *variants*) ⌢ *identifiers-of-variants* (tl *variants*) 113 113

annotations The result of this function is a sequence containing the field identifiers of a variant field component of a record. The order of the identifiers in the sequence is identical to the order in which they occur in the variant field component.

identifiers-of-a-variant : *Variant* → *Identifier**

identifiers-of-a-variant (*variant*) ≜
  let *mk-Variant* (-, *fields*) = *variant* in
  identifiers-of-field-list (*fields*) 113

annotations The result of this function is a sequence containing the field identifiers of a variant component of a variant field of a record. The order of the identifiers in the sequence is identical to the order in which they occur in the variant component.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

is-compatible-variant-labels : *Typed* × *Variant** → *Environment* → $\mathbb{B}$

is-compatible-variant-labels (*type*, *variants*)ρ $\triangleq$
  $\forall$ *mk-Variant* (*labels*, -) $\in$ elems *variants* · *is-expression-compatible* (*type*, *t-set-definition* (*labels*)ρ)ρ 121 243

annotations Check that all of the variant labels are expression-compatible with the tag type.

is-disjoint-variant-labels : *Variant** → *Environment* → $\mathbb{B}$

is-disjoint-variant-labels (*variants*)ρ $\triangleq$
  $\forall i, j \in$ inds *variants* ·
    $i = j \vee$
    let *mk-Variant* (l_i, -) = *variants* (i) in
    let *mk-Variant* (l_j, -) = *variants* (j) in
    evaluate-constant-set-definition (l_i)ρ $\cap$ *evaluate-constant-set-definition* (l_j)ρ = { } 257 257

annotations Check that the sets of values defined by each of the variant label lists are disjoint.

is-complete-variant-labels : *Typed* × *Variant** → *Environment* → $\mathbb{B}$

is-complete-variant-labels (*type*, *variants*)ρ $\triangleq$
  $\bigcup$ {*evaluate-constant-set-definition* (*labels*)ρ | *mk-Variant* (*labels*, -) $\in$ elems *variants*} = 257
  values-of (*type*)ρ 329

annotations Check that there is a value contained in the variant label lists for each value of the tag type.

6.3.13 Storage Allocation

Program storage is allocated during the elaboration of variable declarations and during the binding of actual parameters to formal parameters that occurs on procedure calls.

a-typed : *Typed* × *Storage* → *Environment* → (*Variable* × *Storage*)

a-typed (*typed*, *store*)ρ $\triangleq$
  (*is-Basic-type*(*typed*) → *a-basic-type* (*typed*, *store*)ρ, 115
  is-Type-name(*typed*) → *a-structure* (*typed*, *store*)ρ, 114
  is-System-basic-type(*typed*) → *a-system-storage* (*typed*, *store*)ρ) 120

annotations Note that a value of type *Type* has been replaced by a value of type *Typed* at this point — information on the type defined in the environment is used for storage allocation, rather than the type information defined by the source code.

6.3.13.1 Allocating Storage for Constructed Types

a-structure : *Typed* × *Storage* → *Environment* → (*Variable* × *Storage*)

a-structure (*struc*, *store*)ρ $\triangleq$
  let *struc* = *structure-of* (*type*)ρ in 315
  (*is-Enumeration-structure*(*struc*) → *a-enumeration-structure* (*struc*, *store*)ρ, 115
  is-Subrange-structure(*struc*) → *a-subrange-structure* (*struc*, *store*)ρ, 115
  is-Set-structure(*struc*) → *a-set-structure* (*struc*, *store*)ρ, 115
  is-Packed-set-structure(*struc*) → *a-packed-set-structure* (*struc*, *store*)ρ, 115
  is-Pointer-structure(*struc*) → *a-pointer-structure* (*struc*, *store*)ρ, 115
  is-Procedure-structure(*struc*) → *a-procedure-structure* (*struc*, *store*)ρ, 116
  is-Array-structure(*struc*) → *a-array-structure* (*struc*, *store*)ρ, 116
  is-Record-type(*struc*) → *a-record-structure* (*struc*, *store*)ρ, 117
  struc = OPAQUE → *mk-* (*struc*, *store*))

annotations Identifiers that have been declared as identifiers of variables or parameters of a constructed type have their type name looked up in the environment to give their structure, and storage is allocated for this structure.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.3.13.2 Allocating Storage for Elementary Types

6.3.13.2.1 Allocating Storage for Basic Types

All basic types are elementary types — see 6.10.2.

a-basic-type : *Typed* × *Storage* → *Environment* → (*Variable* × *Storage*)

a-basic-type (*typed*, *store*)ρ $\triangleq$
  let *mk-* (*loc*, *nstore*) = *generate-loc* (*store*) in 338
  if *is-whole-number-type* (*typed*)ρ 323
  then *mk-* (*mk-Elementary-variable* (*loc*, *values-of* (*typed*)), *nstore*) 329
  else *mk-* (*mk-Elementary-variable* (*loc*, nil), *nstore*)

annotations The storage for a whole-number variable is tagged withthe values included in the whole-number type; there are no individual storage-allocation functions for other basic types since they have no associated structure.

6.3.13.2.2 Allocating Storage for Enumeration Types

a-enumeration-structure : *Enumeration-structure* × *Storage* → *Environment* → (*Variable* × *Storage*)

a-enumeration-structure (*mk-Enumeration-structure* (-), *store*)ρ $\triangleq$
  let *mk-* (*loc*, *nstore*) = *generate-loc* (*store*) in 338
  mk- (*mk-Elementary-variable* (*loc*, nil), *nstore*)

6.3.13.2.3 Allocating Storage for Subrange Types

a-subrange-structure : *Subrange-structure* × *Storage* → *Environment* → (*Variable* × *Storage*)

a-subrange-structure (*mk-Subrange-structure* (*rtype*, *values*), *store*)ρ $\triangleq$
  let *mk-* (*loc*, *nstore*) = *generate-loc* (*store*) in 338
  mk- (*mk-Elementary-variable* (*loc*, *values*), *nstore*)

annotations The storage for a subrange variable is tagged with the values included in the subrange.

6.3.13.2.4 Allocating Storage for Set Types

a-set-structure : *Set-structure* × *Storage* → *Environment* → (*Variable* × *Storage*)

a-set-structure (*mk-Set-structure* (-), *store*)ρ $\triangleq$
  let *mk-* (*loc*, *nstore*) = *generate-loc* (*store*) in 338
  mk- (*mk-Elementary-variable* (*loc*, nil), *nstore*)

6.3.13.2.5 Allocating Storage for Packedset Types

a-packed-set-structure : *Packed-set-structure* × *Storage* → *Environment* → (*Variable* × *Storage*)

a-packed-set-structure (*mk-Packed-set-structure* (-), *store*)ρ $\triangleq$
  let *mk-* (*loc*, *nstore*) = *generate-loc* (*store*) in 338
  mk- (*mk-Elementary-variable* (*loc*, nil), *nstore*)

6.3.13.2.6 Allocating Storage for Pointer Types

a-pointer-structure : *Pointer-structure* × *Storage* → *Environment* → (*Variable* × *Storage*)

a-pointer-structure (*mk-Pointer-structure* (*type*), *store*)ρ $\triangleq$
  let *mk-* (*loc*, *nstore*) = *generate-loc* (*store*) in 338
  mk- (*mk-Pointer-variable* (*loc*, *type*), *nstore*)

annotations The storage for a pointer variable is tagged with the type of the bound type of the pointer type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.3.13.2.7 Allocating Storage for Procedure Types

a-procedure-structure : *Procedure-structure* × *Storage* → *Environment* → (*Variable* × *Storage*)

a-procedure-structure (*struc*, *store*)ρ ≙
  cases *struc* :
    mk-Proper-procedure-structure (-) → *a-proper-procedure-structure* (*struc*, *store*)ρ, 116
    mk-Function-procedure-structure (-) → *a-function-procedure-structure* (*struc*, *store*)ρ 116
  end

Allocating Storage for Proper Procedure Types

a-proper-procedure-structure : *Proper-procedure-structure* × *Storage* → *Environment* → (*Variable* × *Storage*)

a-proper-procedure-structure (*mk-Proper-procedure-structure* (-), *store*)ρ ≙
  let *mk-* (*loc*, *nstore*) = *generate-loc* (*store*) in 338
  mk- (*mk-Elementary-variable* (*loc*, nil), *nstore*)

Allocating Storage for Function Procedure Types

a-function-procedure-structure : *Function-procedure-structure* × *Storage* → *Environment* → (*Variable* × *Storage*)

a-function-procedure-structure (*mk-Function-procedure-structure* (-), *store*)ρ ≙
  let *mk-* (*loc*, *nstore*) = *generate-loc* (*store*) in 338
  mk- (*mk-Elementary-variable* (*loc*, nil), *nstore*)

6.3.13.3 Allocating Storage for Structured (Composite) Types

The meaning given to programs by this part of ISO/IEC 10514 is independent of any particular layout strategy in mapping components of a variable of a structured (composite) type onto the addressable locations of an underlying computer system. The storage model of this part of ISO/IEC 10514, therefore allocates storage for each component of a variable of a structured (composite) type independently of any other components of that variable.

NOTE — Conforming implementations are required to document the layout strategy that is used for a particular target computer system — see 4.9e.

6.3.13.3.1 Allocating Storage for Array Types

Dynamic Semantics

The allocation of storage for a variable of an array structure shall allocate storage for each component of the array structure such that each component may be accessed using an indexed designator — see 6.7.2.

a-array-structure : *Array-structure* × *Storage* → *Environment* → (*Array-variable* × *Storage*)

a-array-structure (*mk-Array-structure* (*itype*, *ctype*), *store*)ρ ≙
  let *index-set* = *values-of* (*itype*)ρ in 329
  let *mk-* (*acomp*, *nstore*) = *a-array-components* (*ctype*, *index-set*, *store*)ρ in 116
  mk- (*mk-Array-variable* (*acomp*), *nstore*)

annotations The result is an array variable, which is a mapping from each value in the set specified by its index type to a distinct variable whose type is the component type of the array type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

a-array-components : *Typed* × *Value*-set × *Storage* → *Environment* → (*Array-component* × *Storage*)

a-array-components (*ctype*, *iset*, *store*)ρ ≙
  if *iset* = { }
  then *mk-* ({↦}, *store*)
  else let *d* ∈ *iset* in
    let *mk-* (*loc*, *pstore*) = *a-typed* (*ctype*, *store*)ρ in 114
    let *mk-* (*rest*, *nstore*) = *a-array-components* (*ctype*, *iset* − {*d*}, *pstore*)ρ in 116
    mk- ({*d* ↦ *loc*} ∪ *rest*, *nstore*)

6.3.13.3.2 Allocating Storage for Record Types

Dynamic Semantics

The allocation of storage for a variable of a record structure shall allocate storage for each group of fixed fields of the record structure such that each field may be accessed using a selected designator — see 6.7.3.

The allocation of storage for a variable of a record structure shall allocate storage for each group of variant fields of the record structure such that each field of the active variant, and the tag field, if present, may be accessed using a selected designator — see 6.7.3.

NOTE — In the case of a group of variant fields with an identified tag field, storage is allocated for the tag field and the active variant is determined by the value of that field. If there is no identified tag field, the storage model in this part of ISO/IEC 10514 constructs an implicit tag field so that it can be specified that an exception occurs if a field of an inactive variant is accessed by a value designator — see 6.8.4.3.

a-record-structure : *Record-structure* × *Storage* → *Environment* → (*Record-variable* × *Storage*)

a-record-structure (*mk-Record-structure* (*fields*), *store*)ρ ≙
  let *mk-* (*fcomp*, *nstore*) = *a-field-list* (*fields*, *store*)ρ in 117
  mk- (*mk-Record-variable* (*fcomp*), *nstore*)

annotations The result is a record variable that consists of a mapping from the identifier field names to the associated storage.

Allocating Storage for Fields

a-field-list : *Field-list-structure* × *Storage* → *Environment* → (*Record-components* × *Storage*)

a-field-list (*fields*, *store*)ρ ≙
  if *fields* = [ ]
  then *mk-* ({↦}, *store*)
  else let *i* ∈ inds *fields* in
    let *mk-* (*loc*, *pstore*) = *a-fields-structure* (*fields*(*i*), *store*)ρ in 117
    let *mk-* (*rest*, *nstore*) = *a-field-list* (*del* (*fields*, *i*), *pstore*)ρ in 117 615
    mk- (*loc* ∪ *rest*, *nstore*)

annotations The result is a record variable that consists of a mapping from the identifier field names to the associated storage.

a-fields-structure : *Fields-structure* × *Storage* → *Environment* → (*Record-components* × *Storage*)

a-fields-structure (*struc*, *store*)ρ ≙
  cases *struc* :
    mk-Fixed-fields-structure (-, -) → *a-fixed-fields-structure* (*struc*, *store*)ρ, 118
    mk-Variant-fields-structure (-, -, -, -) → *a-variant-fields-structure* (*struc*, *store*)ρ 118
  end

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Allocating Storage for Fixed Fields

a-fixed-fields-structure : *Fixed-fields-structure* × *Storage* → *Environment* → (*Record-components* × *Storage*)
a-fixed-fields-structure (*mk-Fixed-fields-structure* (*ids*, *type*), *store*)ρ ≜
  let *fids* = elems *ids* in
  a-record-component (*type*, *fids*, *store*)ρ 118

a-record-component : *Field-list* × *Identifier*-set × *Storage* → *Environment* → (*Record-components* × *Storage*)
a-record-component (*ftype*, *ids*, *store*)ρ ≜
  if *ids* = { }
  then *mk-* ({↦}, *store*)
  else let *id* ∈ *ids* in
    let *mk-* (*loc*, *pstore*) = *a-typed* (*ftype*, *store*)ρ in 114
    let *mk-* (*rest*, *nstore*) = *a-record-component* (*ftype*, *ids* − {*id*}, *pstore*)ρ in 118
    mk- ({*id* ↦ *loc*} ∪ *rest*, *nstore*)

annotations The result is a record variable that consists of a mapping from the identifier field names to the associated storage.

Allocating Storage for Variant Fields

a-variant-fields-structure : *Variant-fields-structure* × *Storage* → *Environment* → (*Record-components* × *Storage*)
a-variant-fields-structure (*mk-Variant-fields-structure* (*tag*, *tagt*, *variants*, *elsep*), *store*)ρ ≜
  if *tag* = nil
  then *allocate-tagged-variables* (*tagt*, *variants*, *elsep*, *store*)ρ 118
  else *allocate-variant-variables* (*tag*, *tagt*, *variants*, *elsep*, *store*)ρ 119

Allocating Storage for Implicitly Tagged Variants

allocate-tagged-variables : *Typed* × *Variant-structure** × *Fields-list-structure* × *Storage* → *Environment* →
  (*Record-components* × *Storage*)
allocate-tagged-variables (*tagt*, *variants*, *elsep*, *store*)ρ ≜
  let *mk-* (*tagloc*, *qstore*) = *a-typed* (*tagt*, *store*)ρ in 114
  let *mk-* (*varlocs*, *pstore*) = *a-variants* (*variants*, *qstore*)ρ in 119
  let *mk-* (*elselocs*, *nstore*) = *a-elsepart* (*varlocs*, *elsep*, *pstore*)ρ in 119
  let *tagvar* = *mk-Tag-variable* (*tagloc*, *varlocs* ∪ *elselocs*) in
  mk- (merge {*tag-tagged-variable* (*tagloc*, *loc*) | *loc* ∈ (*varlocs* ∪ *elselocs*)}, *nstore*) 118

annotations Allocate storage for each of the components of a variant fields component of a variant record.

tag-tagged-variable : *Tag-variable* × *Variant-component* → *Record-Variable*
tag-tagged-variable (*tagloc*, *varc*) ≜
  let *mk-Variant-component* (*labels*, *loc*) = *varc* in
  {*id* ↦ *mk-Tagged-variable* (*loc* (*id*), *tagloc*, *labels*) | *id* ∈ dom *loc*}

annotations Construct a tagged variable from the tag location and the range of values associated with that variant component.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Allocating Storage for Explicitly Tagged Variants

allocate-variant-variables :
Identifier × *Typed* × *Variant-structure** × *Fields-list-structure* × *Storage* → *Environment*
→ (*Record-component* × *Storage*)

allocate-variant-variables (*tag*, *tagt*, *variants*, *elsep*, *store*)ρ ≙
let *mk-* (*tagloc*, *qstore*) = *a-typed* (*tagt*, *store*)ρ in 114
let *mk-* (*varlocs*, *pstore*) = *a-variants* (*variants*, *qstore*)ρ in 119
let *mk-* (*elselocs*, *nstore*) = *a-elsepart* (*elsep*, *pstore*)ρ in 119
mk- ({*tag* ↦ *mk-Tag-variable* (*tagloc*, *varlocs* ∪ *elselocs*)}
∪ merge {*tag-variant-variable*(*tagloc*, *var*) | *var* ∈ (*varlocs* ∪ *elselocs*)}, *nstore*)

annotations Allocate a storage location for the tag field. Allocate storage for each variant, and for the variant else part if it is present.

tag-variant-variable : *Elementary-variable* × *Variant-component* → *Record-variable*

tag-variant-variable (*tagloc*, *varc*) ≙
let *mk-Variant-component* (*labels*, *loc*) = *varc* in
{*id* ↦ *mk-Variant-variable* (*loc* (*id*), *tagloc*, *labels*) | *id* ∈ dom *loc*}

annotations Construct a mapping that associates a variant variable with each of the label values of a variant component.

Auxiliaries

a-variants : *Variant-structure** × *Storage* → *Environment* → (*Variant-component*-set × *Storage*)

a-variants (*variants*, *store*)ρ ≙
if *variants* = [ ]
then *mk-* ({↦}, *store*)
else let *id* ∈ inds *variants* in
let *mk-* (*loc*, *pstore*) = *a-variant-structure* (*variants*(*i*), *store*)ρ in 119
let *mk-* (*rest*, *nstore*) = *a-variants* (*del* (*variants*, *i*), *pstore*)ρ in 119 615
mk- ({*loc*} ∪ *rest*, *nstore*)

a-variant-structure : *Variant-structure* × *Storage* → *Environment* → (*Variant-component* × *Storage*)

a-variant-structure (*mk-Variant-structure* (*labels*, *fields*), *store*)ρ ≙
let *mk-Set-value* (*value*) = *labels* in
let *mk-* (*locs*, *nstore*) = *a-field-list* (*fields*, *store*)ρ in 117
mk- (*mk-Variant-component* (*value*, *locs*), *nstore*)

annotations Construct a sequence of variant structures from a sequence of variants.

a-elsepart :
Variant-component-set × *Fields-list-structure* × *Storage* → *Environment*
→ (*Variant-component*-set × *Storage*)

a-elsepart (*variants*, *elsep*, *store*)ρ ≙
if *elsep* = nil
then { }
else let *elseplabels* = *values-of* (*tagt*)ρ − merge {*variant.labels* | *variant* ∈ elems *variants*} in 329
let *mk-* (*locs*, *nstore*) = *a-field-list* (*elsep*, *store*)ρ in 117
mk- ({*mk-Variant-component* (*elseplabels*, *locs*)}, *nstore*)

annotations Allocate storage space for the variant else part if it is present.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.3.13.4 Allocating Storage for System Storage Types

$a\text{-}system\text{-}storage : Typed \times Storage \to Environment \to (Variable \times Storage)$

$a\text{-}system\text{-}storage\,(typed, store)\rho \triangleq$
  cases $typed$:
    LOC-TYPE → $a\text{-}basic\text{-}type\,(typed, store)\rho$, 115
    COROUTINE-TYPE → $a\text{-}basic\text{-}type\,(typed, store)\rho$, 115
    EXCEPTIONSOURCE-TYPE → $a\text{-}basic\text{-}type\,(typed, store)\rho$ 115
  end

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.4 Expression and Assignment Compatibility

The concept of compatibility between entities of possibly differing type is fundamental to Modula-2. Expression compatibility determines the types that may be combined in forming expressions; assignment compatibility determines, for a given type of variable, the type or types of the values of expressions that may be assigned to variables of that type.

There is a third form of compatibility, called 'parameter compatibility', which determines for a given type and structure of formal parameter, the type and structure of the actual parameters that may be passed to formal parameters of that type and structure — see 6.9.3.

6.4.1 Expression Compatibility

Modula-2 limits the types that may be combined in forming expressions to identical types except that there are special rules where subrange types are involved and where literal values are involved. Expression compatibility is a symmetric relation — a type of an expression T_{e1} is expression compatible with a type of an expression T_{e2} if T_{e2} is expression compatible with T_{e1}.

Static Semantics

The types of two expressions shall be expression-compatible if any of the following statements is true:

a) The types are identical types.

b) One type is a complex number type and the other type is the $\mathbb{C}$-type.

c) One type is a real number type and the other type is the $\mathbb{R}$-type.

d) One type is a whole number type and the other type is the $\mathbb{Z}$-type.

e) One type is the character type and the other type is the character literal type (the string literal type of length 0 or of length 1).

CHANGE — In *Programming in Modula-2* a string literal of length 0 represents the empty string only and the character type is not expression-compatible with the type of the empty string.

NOTES

1 A value of a subrange type is also a value of the host type of that subrange type, and the type of expressions that are value designators, or function calls, is defined in terms of the host type of any subrange type that might otherwise apply — see 6.8.4 and 6.8.5. Values of subranges of the same host type are therefore expression compatible with each other, and with that host type, under rule 6.4.1a.

2 There are other special cases of expression compatibility for pointer relational operations and for procedure relational operations; these are dealt with where they arise — see 6.8.2.7.

$$\mathit{is\text{-}expression\text{-}compatible} : \mathit{Expression\text{-}typed} \times \mathit{Expression\text{-}typed} \rightarrow \mathit{Environment} \rightarrow \mathbb{B}$$

$$\begin{aligned}
&\mathit{is\text{-}expression\text{-}compatible}\ (e1\mathit{type}, e2\mathit{type})\rho \triangleq \\
&\quad e1\mathit{type} = e2\mathit{type} \vee \\
&\quad \mathit{is\text{-}complex\text{-}compatible}\ (e1\mathit{type}, e2\mathit{type})\rho \vee && 122 \\
&\quad \mathit{is\text{-}real\text{-}compatible}\ (e1\mathit{type}, e2\mathit{type})\rho \vee && 122 \\
&\quad \mathit{is\text{-}whole\text{-}number\text{-}compatible}\ (e1\mathit{type}, e2\mathit{type})\rho \vee && 122 \\
&\quad \mathit{is\text{-}character\text{-}compatible}\ (e1\mathit{type}, e2\mathit{type})\rho && 122
\end{aligned}$$

annotations The '*t*-...' functions used in applications of the function *is-expression-compatible* return the host type for values of a subrange type; thus this function does not need to deal with special rules for subrange types.

In parts of the formal definition, expression compatibility between the types of two expressions, *expra* and *exprb*, is tested using relations of the form $t\text{-}expression\ (expra)\rho = t\text{-}expression\ (exprb)\rho$

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

$is\text{-}complex\text{-}compatible : Expression\text{-}typed \times Expression\text{-}typed \rightarrow Environment \rightarrow \mathbb{B}$

$is\text{-}complex\text{-}compatible\,(e1type, e2type)\rho \triangleq$
$e1type = \mathbb{C}\text{-TYPE} \wedge is\text{-}Complex\text{-}number\text{-}type(e2type) \vee$
$is\text{-}Complex\text{-}number\text{-}type(e1type) \wedge e2type = \mathbb{C}\text{-TYPE}$

$is\text{-}real\text{-}compatible : Expression\text{-}typed \times Expression\text{-}typed \rightarrow Environment \rightarrow \mathbb{B}$

$is\text{-}real\text{-}compatible\,(e1type, e2type)\rho \triangleq$
$e1type = \mathbb{R}\text{-TYPE} \wedge is\text{-}Real\text{-}number\text{-}type(e2type) \vee$
$is\text{-}Real\text{-}number\text{-}type(e1type) \wedge e2type = \mathbb{R}\text{-TYPE}$

$is\text{-}whole\text{-}number\text{-}compatible : Expression\text{-}typed \times Expression\text{-}typed \rightarrow Environment \rightarrow \mathbb{B}$

$is\text{-}whole\text{-}number\text{-}compatible\,(e1type, e2type)\rho \triangleq$
$e1type = \mathbb{Z}\text{-TYPE} \wedge is\text{-}whole\text{-}number\text{-}type\,(e2type)\rho \vee$ 323
$is\text{-}whole\text{-}number\text{-}type\,(e1type)\rho \wedge e2type = \mathbb{Z}\text{-TYPE}$ 323

$is\text{-}character\text{-}compatible : Expression\text{-}typed \times Expression\text{-}typed \rightarrow Environment \rightarrow \mathbb{B}$

$is\text{-}character\text{-}compatible\,(e1type, e2type)\rho \triangleq$
$is\text{-}character\text{-}type\,(e1type)\rho \wedge is\text{-}character\text{-}constant\text{-}type\,(e2type)\rho \vee$ 323 324
$is\text{-}character\text{-}constant\text{-}type\,(e1type)\rho \wedge is\text{-}character\text{-}type\,(e2type)\rho$ 324 323

6.4.2 Assignment Compatibility

Modula-2 limits the types of value that may be assigned to a variable to identical types except that there are special rules for subrange types, literal values, whole number types, and the address type. Assignment compatibility is not a symmetric relation since the types of variables and of values are treated differently.

Within a procedure block, formal parameters are treated as variables to which assignments may be made. However, formal parameters are of a formal type and may have the structure of an open array — see 6.3.10. No direct assignments to open array formal parameters may be made, even from another open array formal parameter of identical structure.

Static Semantics

The type T_v of a variable denoted by a variable designator shall be assignment-compatible with the type T_e of the value of an expression if any of the following statements is true:

a) T_v is identical to the type T_e and that type is not a formal type having an open array structure.

b) T_v is a subrange of the type T_e.

c) T_v is the unsigned type or a subrange of the unsigned type and T_e is the signed type or is the $\mathbb{Z}$-type.

d) T_v is the signed type or a subrange of the signed type and T_e is the unsigned type or is the $\mathbb{Z}$-type.

e) T_v is a real number type and T_e is the $\mathbb{R}$-type.

f) T_v is a complex number type and T_e is the $\mathbb{C}$-type.

g) T_v is a pointer type and T_e is the nil type.

h) T_v is a proper procedure type or a function procedure type, and the expression designates a procedure value, or procedure constant value, of a procedure that has the same structure as the procedure type T_v, and that has been declared at declaration level 0.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

i) T_v is the character type, or a subrange of the character type, and T_e is the character literal type (the string literal type of length 0 or of length 1).

j) T_v is an array type having the character type as its component type and T_e is the string literal type of length less than or equal to the number of components in arrays of the type T_v.

k) T_v is the address type and T_e is a pointer type, or T_v is a pointer type and T_e is the address type.

NOTES

1 A variable designator only occurs on the left-hand side of an assignment statement, or as an actual parameter that corresponds to a variable formal parameter; only the first of these uses is relevant to assignment compatibility. Variable identifiers that occur in expressions are *value* designators.

2 The component type of a concrete string type is the character type. Since it must be possible to assign the value of the character type that has been defined by an implementation as its string terminating character to a component of a concrete string variable, a quoted string is not assignment-compatible with an array whose component type is a subrange of the character type.

CHANGES

1 In *Programming in Modula-2* a string literal of length zero represents the empty string only and the character type is not assignment-compatible with the type of the empty string.

2 A concrete string type, i.e. an array whose component type is the character type, that is either the target of a string assignment, or a formal parameter to which a string is passed as an actual parameter, may have an index type of any ordinal type.

CLARIFICATION — This part of ISO/IEC 10514 has not adopted a change to Modula-2 (described in the fourth edition of *Programming in Modula-2*) of always requiring a string terminator for a string value. The length of a string literal may therefore equal the number of components in an array to which it is assigned.

$is\text{-}assignment\text{-}compatible : Typed \times Expression\text{-}typed \rightarrow Environment \rightarrow \mathbb{B}$

$is\text{-}assignment\text{-}compatible\ (ltype, rtype)\rho \triangleq$
$\quad wf\text{-}simple\text{-}assignment\ (ltype, rtype)\rho \vee$ 123
$\quad wf\text{-}subrange\text{-}assignment\ (ltype, rtype)\rho \vee$ 123
$\quad wf\text{-}unsigned\text{-}assignment\ (ltype, rtype)\rho \vee$ 124
$\quad wf\text{-}signed\text{-}assignment\ (ltype, rtype)\rho \vee$ 124
$\quad wf\text{-}real\text{-}assignment\ (ltype, rtype)\rho \vee$ 124
$\quad wf\text{-}complex\text{-}assignment\ (ltype, rtype)\rho \vee$ 124
$\quad wf\text{-}pointer\text{-}assignment\ (ltype, rtype)\rho \vee$ 124
$\quad wf\text{-}procedure\text{-}assignment\ (ltype, rtype)\rho \vee$ 124
$\quad wf\text{-}character\text{-}assignment\ (ltype, rtype)\rho \vee$ 124
$\quad wf\text{-}string\text{-}constant\text{-}assignment\ (ltype, rtype)\rho \vee$ 124
$\quad wf\text{-}pointer\text{-}and\text{-}address\text{-}assignment\ (ltype, rtype)\rho$ 125

$wf\text{-}simple\text{-}assignment : Typed \times Expression\text{-}typed \rightarrow Environment \rightarrow \mathbb{B}$

$wf\text{-}simple\text{-}assignment\ (ltype, rtype)\rho \triangleq$
$\quad ltype = rtype \wedge$
$\quad \neg\ is\text{-}open\text{-}array\ (rtype)\rho$ 328

annotations The type of the variable designator and the type of the expression are identical. The result type of the expression is not an open array type.

$wf\text{-}subrange\text{-}assignment : Typed \times Expression\text{-}typed \rightarrow Environment \rightarrow \mathbb{B}$

$wf\text{-}subrange\text{-}assignment\ (ltype, rtype)\rho \triangleq$
$\quad is\text{-}subrange\text{-}type\ (lype)\rho \wedge$ 323
$\quad host\text{-}type\text{-}of\ (ltype)\rho = rtype$ 325

annotations The type of the variable designator is a subrange of the type of the expression.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wf-unsigned-assignment : *Typed* × *Expression-typed* → *Environment* → 𝔹

wf-unsigned-assignment (*ltype*, *rtype*)ρ ≙
  host-type-of (*ltype*)ρ = UNSIGNED-TYPE ∧ 325
  (*rtype* = SIGNED-TYPE ∨ *rtype* = ℤ-TYPE)

annotations The type of the variable designator is the unsigned type or a subrange of the unsigned type, and the type of the expression is the signed type or ℤ-type.

wf-signed-assignment : *Typed* × *Expression-typed* → *Environment* → 𝔹

wf-signed-assignment (*ltype*, *rtype*)ρ ≙
  host-type-of (*ltype*)ρ = SIGNED-TYPE ∧ 325
  (*rtype* = UNSIGNED-TYPE ∨ *rtype* = ℤ-TYPE)

annotations The type of the variable designator is the signed type or a subrange of the signed type, and the type of the expression is the unsigned type or ℤ-type.

wf-real-assignment : *Typed* × *Expression-typed* → *Environment* → 𝔹

wf-real-assignment (*ltype*, *rtype*)ρ ≙
  is-Real-number-type(*ltype*) ∧
  rtype = ℝ-TYPE

annotations The type of the variable designator is a real type and the type of the expression is the ℝ-type.

wf-complex-assignment : *Typed* × *Expression-typed* → *Environment* → 𝔹

wf-complex-assignment (*ltype*, *rtype*)ρ ≙
  is-Complex-type(*ltype*) ∧
  rtype = ℂ-TYPE

annotations The type of the variable designator is a complex type and the type of the expression is the ℂ-type.

wf-pointer-assignment : *Typed* × *Expression-typed* → *Environment* → 𝔹

wf-pointer-assignment (*ltype*, *rtype*)ρ ≙
  is-pointer-type (*ltype*)ρ ∧ *rtype* = NIL-TYPE 322

annotations The type of the variable designator is a pointer type and the type of the expression is the nil type.

wf-procedure-assignment : *Typed* × *Expression-typed* → *Environment* → 𝔹

wf-procedure-assignment (*ltype*, *rtype*)ρ ≙
  is-procedure-type (*ltype*) ∧ 321
  structure-of (*ltype*)ρ = *procedure-structure* (*rtype*) ∧ 315 316
  procedure-level (*rtype*) = 0 316

annotations The type of the variable designator is a proper procedure type or function procedure type, and the type of the expression is a proper procedure or function procedure value with the same structure, declared at level 0.

wf-character-assignment : *Typed* × *Expression-typed* → *Environment* → 𝔹

wf-character-assignment (*ltype*, *rtype*)ρ ≙
  host-type-of (*ltype*)ρ = CHARACTER-TYPE ∧ 325
  is-character-constant-type (*rtype*)ρ 324

annotations The type of the variable designator is a character type, and the type of the expression is the character literal type.

wf-string-constant-assignment : *Typed* × *Expression-typed* → *Environment* → 𝔹

wf-string-constant-assignment (*ltype*, *rtype*)ρ ≙
  is-array-type (*ltype*)ρ ∧ ¬ *is-open-array* (*ltype*)ρ ∧ *component-type-of* (*ltype*)ρ = CHARACTER-TYPE ∧ 321 328 325
  is-String-type(*rtype*) ∧
  rtype.length ≤ *upper-bound-of* (*ltype*)ρ − *lower-bound-of* (*ltype*)ρ + 1 328 328

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations The type of the variable for string literal assignment is an array of characters. The result of the expression is a string literal, and the length of the string literal is less than or equal to the number of elements in the array associated with the type *ltype*.

$$wf\text{-}pointer\text{-}and\text{-}address\text{-}assignment : Typed \times Expression\text{-}typed \rightarrow Environment \rightarrow \mathbb{B}$$

$$wf\text{-}pointer\text{-}and\text{-}address\text{-}assignment\,(ltype, rtype)\rho \triangleq$$
$$(is\text{-}address\text{-}type\,(ltype)\rho \wedge is\text{-}pointer\text{-}type\,(rtype)\rho) \vee \quad \text{324 322}$$
$$(is\text{-}pointer\text{-}type\,(ltype)\rho \wedge is\text{-}address\text{-}type\,(rtype)\rho) \quad \text{322 324}$$

annotations The type of the variable is the address type, and the result of the expression is a pointer type, or vice-versa.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.5 Blocks

Blocks consist of a possibly empty sequence of declarations which may be followed by a body; they are a component of program modules, implementation modules, local modules, and full procedure declarations. The body contains the statements that are to be executed when a module is initialized or finalized, or when a procedure is activated.

The presence of a body is required only for function procedure blocks; the bodies of function procedures return a result value whereas the bodies of modules and of proper procedures do not return a result value.

The presence of a body is marked by the keyword `BEGIN` which is followed by a block body. In the case of module blocks, the block body marked by the keyword `BEGIN` is the initialization body; it may be followed by a finalization body, marked by the keyword `FINALLY`. Block bodies have a normal part and, optionally, an exceptional part to handle exceptions raised during execution of the normal part.

The execution of the block bodies of protected modules and of protected procedures is protected from the occurrence of interrupts — see 6.1.11.

6.5.1 Proper Procedure Blocks

A proper procedure block defines the sequence of declarations and the optional procedure body that are associated with the proper procedure identifier. On a call of the procedure, the initialization bodies of any local modules declared in the declarations are executed. If a procedure body is present, it is then executed in an environment that includes identifiers introduced in the declarations of the procedure block.

Concrete Syntax

proper procedure block = declarations, [procedure body], "END" ;

procedure body = "BEGIN", block body ;

Abstract Syntax

types

Proper-procedure-block :: *decls* : *Declarations*
body : [*Block-body*]

annotations In the translation from concrete to abstract syntax, a proper procedure block without a procedure body is represented by a *Proper-procedure-block* with a nil value for the *body* component.

Static Semantics

If present, the block body of a proper procedure block shall be a well-formed block body that does not return a result.

$wf\text{-}proper\text{-}procedure\text{-}body : Proper\text{-}procedure\text{-}block \to Environment \to \mathbb{B}$

$wf\text{-}proper\text{-}procedure\text{-}body\ (mk\text{-}Proper\text{-}procedure\text{-}block\ (decls, body))\rho \triangleq$
  let $\rho_{body} = overwrite\text{-}environment\ (d\text{-}declarations\ (decls)\rho_{body})\rho$ in 319 64
  $body = \mathsf{nil} \vee$
  $(wf\text{-}block\text{-}body\ (decls, body)\rho_{body} \wedge wf\text{-}non\text{-}result\text{-}body\ (body)\rho_{body})$ 142 142

Dynamic Semantics

If there are local modules contained in the declarations of the proper procedure block, they shall be initialized in the textual order in which they occur — see 6.5.6.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

On the execution of a proper procedure block that is not protected and does not have an exceptional part, the procedure body shall be executed in the state of exceptional execution or in the state of normal execution, according to the state of execution of the caller of the procedure.

NOTE — If the procedure block has no exceptional part, the exceptional part that is the current exception handler is found as follows:

— If the procedure has been called, directly or indirectly, from a normal part of another procedure block, or from the initialization body or finalization body of a module block, and the calling procedure or module block has an exceptional part, then that exceptional part will be the current exception handler.

— If the procedure has been called, directly or indirectly, from the exceptional part of a static module, no exceptional part will be the current exception handler.

— If the procedure has been called, directly or indirectly, from the exceptional part of another procedure, the current exception handler will be the exception handler for the caller of that other procedure.

— If the procedure has been called, directly or indirectly, from the exceptional part of a dynamic module, the current exception handler will be the exception handler for the caller of the procedure in which the dynamic module is declared.

If the procedure block has no exceptional part, and no other exceptional part is the current exception handler, the raising of an exception during the execution of the normal part is a termination event — see 6.1.7.

On completion of the execution of the procedure body, if there are local modules contained in the declarations of the proper procedure block, they shall be finalized in the reverse order of their initialization — see 6.5.7.

If an exception is raised during execution of the procedure body the local dynamic modules shall not be finalized.

m-proper-procedure-block : *Proper-procedure-block* → *Environment* $\xrightarrow{o}$ ()

m-proper-procedure-block (*mk-Proper-procedure-block* (*decls*, *body*))ρ $\triangleq$
  def ρ_{decls} = *m-declarations*(*decls*) ρ in 66
  def ρ_{cont} = *c-tixe*({EXCEPTION ↦ (*deallocate-declarations*(*decls*) ρ_{decls}; 345 128
    c-exit(EXCEPTION) ρ)}) ρ_{decls} in 345
  (*initialize-dynamic-modules*(*decls*) ρ_{cont} ; 136
   execute-non-result-body(*body*) ρ_{cont}; 143
   finalize-dynamic-modules(*decls*) ρ_{cont} ; 138
   deallocate-declarations(*decls*) ρ_{decls}) 128

If the procedure is a protected procedure, the protection domain shall be entered before the local dynamic modules are initialized and shall be left after the local dynamic modules have been finalized.

If an exception is raised during execution of the procedure body the protection domain shall not be left.

m-proper-procedure-block-p : *Proper-procedure-block* → *Environment* $\xrightarrow{o}$ ()

m-proper-procedure-block-p (*mk-Proper-procedure-block* (*decls*, *body*))ρ $\triangleq$
  def ρ_{decls} = *m-declarations*(*decls*) ρ in 66
  def ρ_{cont} = *c-tixe*({EXCEPTION ↦ (*deallocate-declarations*(*decls*) ρ_{decls}; 345 128
    c-exit(EXCEPTION) ρ)}) ρ_{decls} in 345
  def *pdom* = *activate-entry-protection*(ρ) in 51
  (*initialize-dynamic-modules*(*decls*) ρ_{cont} ; 136
   execute-non-result-body(*body*) ρ_{cont}; 143
   finalize-dynamic-modules(*decls*) ρ_{cont} ; 138
   activate-exit-protection(*pdom*) ρ; 51
   deallocate-declarations(*decls*) ρ_{decls}) 128

On the execution of a proper procedure block that has an exceptional part, the procedure body shall be executed in the state of normal execution.

On completion of the execution of the block body, the execution state shall revert to that which applied before the execution of the block body.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

If execution of the procedure body completed in the state of exceptional execution, the exception shall be re-raised after the local dynamic modules have been finalized.

If an exception is raised during execution of the exceptional part of the procedure body, the local dynamic modules shall not be finalized.

```
m-proper-procedure-block-h : Proper-procedure-block → Environment ~o~> ()
m-proper-procedure-block-h (mk-Proper-procedure-block (decls, body))ρ ≙
  def ρ_decls = m-declarations(decls) ρ in                                  66
  def ρ_cont = c-tixe({EXCEPTION ↦ (deallocate-declarations(decls) ρ_decls;  345 128
                                    c-exit(EXCEPTION) ρ)}) ρ_decls in       345
  (initialize-dynamic-modules(decls) ρ_cont ;                               136
   push-exception-context(NORMAL-EXECUTION) ;                               342
   execute-non-result-body-h(body) ρ_cont;                                  143
   def context = pop-exception-context() in                                 342
   (finalize-dynamic-modules(decls) ρ_cont ;                                138
    deallocate-declarations(decls) ρ_decls ;                                128
    if is-exceptional (context)                                             342
    then (swop-exception-context(context) ;                                 342
          c-exit(EXCEPTION) ρ )                                             345
    else skip))
```

On the execution of a proper procedure block that is protected and whose body has an exceptional part, if execution of the procedure body completes in the state of exceptional execution, the exception shall be re-raised after the protection domain has been left.

If an exception is raised during execution of the exceptional part of the procedure body, the protection domain shall not be left.

```
m-proper-procedure-block-hp : Proper-procedure-block → Environment ~o~> ()
m-proper-procedure-block-hp (mk-Proper-procedure-block (decls, body))ρ ≙
  def ρ_decls = m-declarations(decls) ρ in                                  66
  def ρ_cont = c-tixe({EXCEPTION ↦ (deallocate-declarations(decls) ρ_decls;  345 128
                                    c-exit(EXCEPTION) ρ)}) ρ_decls in       345
  def pdom = activate-entry-protection(ρ) in                                51
  (initialize-dynamic-modules(decls) ρ_cont ;                               136
   push-exception-context(NORMAL-EXECUTION) ;                               342
   execute-non-result-body-h(body) ρ_cont;                                  143
   def context = pop-exception-context() in                                 342
   (finalize-dynamic-modules(decls) ρ_cont ;                                138
    activate-exit-protection(pdom) ρ ;                                      51
    deallocate-declarations(decls) ρ_decls ;                                128
    if is-exceptional (context)                                             342
    then (swop-exception-context(context) ;                                 342
          c-exit(EXCEPTION) ρ )                                             345
    else skip))
```

Auxiliaries

```
deallocate-declarations : Declarations → Environment ~o~> ()
deallocate-declarations (decls)ρ ≙
  let ids = identifiers-declared-in (decls) in                              82
  deallocate-storage(ids) ρ                                                 337
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.5.2 Function Procedure Blocks

A function procedure block defines the sequence of declarations and the procedure body that are associated with the procedure identifier. On a call of the procedure, the initialization bodies of any local modules declared in the declarations are executed. The procedure body is then executed in an environment that includes identifiers introduced in the declarations of the procedure block.

Concrete Syntax

function procedure block = declarations, function body, "END" ;

function body = "BEGIN", block body ;

CHANGE — *Programming in Modula-2* allows the body of a function procedure to be empty, but the well-formed rules in this part of ISO/IEC 10514 demand at least one return statement. The concrete syntax has been changed to reflect this.

Abstract Syntax

types

Function-procedure-block :: *decls* : *Declarations*
body : *Block-body*

Static Semantics

The block body of a function procedure block shall be a well-formed block body that returns a result.

wf-function-procedure-body : *Function-procedure-block* → *Environment* → 𝔹

wf-function-procedure-body (*mk-Function-procedure-block* (*decls*, *body*))ρ ≜
let ρ_{body} = *overwrite-environment* (*d-declarations* (*decls*)ρ_{body})ρ in 319 64
wf-block-body (*decls*, *body*)ρ_{body} ∧ *wf-result-body* (*body*)ρ_{body} 142 144

Dynamic Semantics

If there are local modules contained in the declarations of the procedure block, they shall be initialized in the textual order in which they occur — see 6.5.6.

On the execution of a function procedure block, that is not protected and does not have an exceptional part, the procedure body shall be executed in the state of exceptional execution or in the state of normal execution, according to the state of execution of the caller of the procedure.

NOTE — If the procedure block has no exceptional part, the exceptional part that is the current exception handler is found as follows:

- If the procedure has been called, directly or indirectly, from an normal part of another procedure block, or from the initialization body or finalization body of a module block, and the calling procedure or module block has an exceptional part, then that exceptional part will be the current exception handler.
- If the procedure has been called, directly or indirectly, from the exceptional part of a static module, no exceptional part will be the current exception handler.
- If the procedure has been called, directly or indirectly, from the exceptional part of another procedure, the current exception handler will be the exception handler for the caller of that other procedure.
- If the procedure has been called, directly or indirectly, from the exceptional part of a dynamic module, the current exception handler will be the exception handler for the caller of the procedure in which the dynamic module is declared.

If the procedure block has no exceptional part, and no other exceptional part is the current exception handler, the raising of an exception during the execution of the normal part is a termination event — see 6.1.7.

On completion of the execution of the procedure body, if there are local modules contained in the declarations of the procedure block, they shall be finalized in the reverse order of their initialization — see 6.5.7.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

If an exception is raised during execution of the procedure body the local dynamic modules shall not be finalized.

m-function-procedure-block : *Function-procedure-block* → *Environment* $\xrightarrow{o}$ ()
m-function-procedure-block (*mk-Function-procedure-block* (*decls*, *body*))ρ $\triangleq$
  def ρ_{decls} = *m-declarations*(*decls*) ρ in 66
  def ρ_{cont} = *c-tixe*({EXCEPTION ↦ (*deallocate-declarations*(*decls*) ρ_{decls}; 345 128
    c-exit(EXCEPTION) ρ)}) ρ_{decls} in 345
  (*initialize-dynamic-modules*(*decls*) ρ_{cont} ; 136
   def *result* = *execute-result-body*(*body*) ρ_{cont} in 144
   (*finalize-dynamic-modules*(*decls*) ρ_{cont} ; 138
    deallocate-declarations(*decls*) ρ_{decls} ; 128
    return *result*))

If the procedure is a protected procedure, the protection domain shall be entered before the local dynamic modules are initialized and shall be left after the local dynamic modules have been finalized.

If an exception is raised during execution of the procedure body the protection domain shall not be left.

m-function-procedure-block-p : *Function-procedure-block* → *Environment* $\xrightarrow{o}$ ()
m-function-procedure-block-p (*mk-Function-procedure-block* (*decls*, *body*))ρ $\triangleq$
  def ρ_{decls} = *m-declarations*(*decls*) ρ in 66
  def ρ_{cont} = *c-tixe*({EXCEPTION ↦ (*deallocate-declarations*(*decls*) ρ_{decls}; 345 128
    c-exit(EXCEPTION) ρ)}) ρ_{decls} in 345
  def *pdom* = *activate-entry-protection*(ρ) in 51
  (*initialize-dynamic-modules*(*decls*) ρ_{cont} ; 136
   let *mk-Block-body* (*normal*, -) = *body* in
   def *result* = *execute-result-body*(*body*) ρ_{cont} in 144
   (*finalize-dynamic-modules*(*decls*) ρ_{cont} ; 138
    activate-exit-protection(*pdom*) ρ ; 51
    deallocate-declarations(*decls*) ρ_{decls} ; 128
    return *result*))

On the execution of a function procedure block that is not protected but does have an exceptional part, the procedure body shall be executed in the state of normal execution.

On completion of the execution of the block body, the execution state shall revert to that which applied before the execution of the block body.

If execution of the procedure body completed in the state of exceptional execution, the exception shall be re-raised after the local dynamic modules have been finalized.

If an exception is raised during execution of the exceptional part of the procedure body, the local dynamic modules shall not be finalized.

m-function-procedure-block-h : *Function-procedure-block* → *Environment* $\xrightarrow{o}$ ()
m-function-procedure-block-h (*mk-Function-procedure-block* (*decls*, *body*))ρ $\triangleq$
  def ρ_{decls} = *m-declarations*(*decls*) ρ in 66
  def ρ_{cont} = *c-tixe*({EXCEPTION ↦ (*deallocate-declarations*(*decls*) ρ_{decls}; 345 128
    c-exit(EXCEPTION) ρ)}) ρ_{decls} in 345
  (*initialize-dynamic-modules*(*decls*) ρ_{cont} ; 136
   push-exception-context(NORMAL-EXECUTION) ; 342
   def *result* = *execute-result-body-h*(*body*) ρ_{cont} in 145
   def *context* = *pop-exception-context*() in 342

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
(finalize-dynamic-modules(decls) ρcont ;                                138
 deallocate-declarations(decls) ρdecls ;                                128
 if is-exceptional (context)                                            342
 then (swop-exception-context(context) ;                                342
       c-exit(EXCEPTION) ρ )                                            345
 else return result ))
```

On the execution of a function procedure block that is protected and has an exceptional part, if execution of the procedure body completes in the state of exceptional execution, the exception shall be re-raised after the protection domain has been left.

If an exception is raised during execution of the exceptional part of the procedure body, the protection domain shall not be left.

```
m-function-procedure-block-hp : Function-procedure-block → Environment ⟶° ()
m-function-procedure-block-hp (mk-Function-procedure-block (decls, body))ρ ≙
  def ρdecls = m-declarations(decls) ρ in                               66
  def ρcont = c-tixe({EXCEPTION ↦ (deallocate-declarations(decls) ρdecls;  345 128
                                   c-exit(EXCEPTION) ρ)}) ρdecls in     345
  def pdom = activate-entry-protection(ρ) in                            51
  (initialize-dynamic-modules(decls) ρcont ;                            136
   push-exception-context(NORMAL-EXECUTION) ;                           342
   def result = execute-result-body-h(body) ρcont in                    145
   def context = pop-exception-context() in                             342
   (finalize-dynamic-modules(decls) ρcont ;                             138
    activate-exit-protection(pdom) ρ ;                                  51
    deallocate-declarations(decls) ρdecls ;                             128
    if is-exceptional (context)                                         342
    then (swop-exception-context(context) ;                             342
          c-exit(EXCEPTION) ρ )                                         345
    else return result ))
```

6.5.3 Module Blocks

A module block consists of a sequence of declarations and an optional module body. It is a component of a program module, an implementation module, or a local module.

Module bodies have an initialization body and, optionally, a finalization body. The initialization body is executed during static module initialization or dynamic module initialization — see 6.5.4 and 6.5.6. The finalization body, if present, is executed during static module finalization or dynamic module finalization — see 6.5.5 and 6.5.7. Initialization and finalization takes place in an environment that includes identifiers introduced in the declarations of the module block.

Concrete Syntax

```
module block = declarations, [ module body ], "END" ;

module body = initialization body, [ finalization body ] ;

initialization body = "BEGIN", block body ;

finalization body = "FINALLY", block body ;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Abstract Syntax

types

Module-block :: *decls* : *Declarations*
initial : [*Block-body*]
final : [*Block-body*]

annotations In the translation from concrete to abstract syntax, a module block with no module body is represented by a *Module-block* with nil values for both the *initial* and *final* components and a module body with no finalization body is represented by a *Module-block* with a nil value for the *final* component.

Static Semantics

If present, The initialization body of a module block shall be a well-formed block body that does not return a result.

If a module block has a finalization body, that body shall be a well-formed block body that does not return a result.

wf-module-body : *Module-block* → *Environment* → $\mathbb{B}$

wf-module-body (*mk-Module-block* (*decls*, *initial*, *final*))ρ $\triangleq$
let ρ_{body} = *overwrite-environment* (*d-declarations* (*decls*)ρ_{body})ρ in 319 64
(*initial* = nil ∨ *wf-block-body* (*decls*, *initial*)ρ_{body} ∧ *wf-non-result-body* (*initial*)ρ_{body}) ∧ 142 142
(*final* = nil ∨ *wf-block-body* (*decls*, *final*)ρ_{body} ∧ *wf-non-result-body* (*final*)ρ_{body}) 142 142

Dynamic Semantics

The elaboration of a module block shall allocate storage for the declarations of of the block.

m-module-block : *Module-block* → *Environment* $\xrightarrow{o}$ *Environment*

m-module-block (*mk-Module-block* (*decls*, -, -))ρ $\triangleq$
m-declarations(*decls*) ρ 66

6.5.4 Static Module Initialization

Static module initialization applies to the initialization of the module blocks of implementation modules, to the execution of the program module block, and, recursively, to the initialization of local modules declared in these module blocks — see 6.1.2 and 6.1.4.1.

A module block comprises a sequence of deelarations and, optionally, a module body. The local modules contained in the declarations of a static module block are initialized in the textual order in which they occur.

A module body comprises an initialization body and, optionally, a finalization body. Finalization bodies of static modules are executed during program termination — see 6.1.7 and 6.5.5. If a coroutine causes termination, and starts to finalize the static modules, no further static modules are initialized — see 6.1.7.

The initialization body of a static module is executed in the environment defined by its import lists and declarations.

Dynamic Semantics

If program termination is in progress, there shall be no further initialization of static modules.

execute-static-module-initialization : *Module-block* → *Environment* $\xrightarrow{o}$ ()

execute-static-module-initialization (*mk-Module-block* (*decls*, *initial*, *final*))ρ $\triangleq$
def *terminating* = *is-program-terminating*() in 340
if *terminating*
then skip

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

else (¬ *is-handled* (*initial*) ∧ ¬ *is-protected-module* (ρ) → *execute-static-module-init*(*decls*, *initial*, *final*) ρ, 76 51 133
¬ *is-handled* (*initial*) ∧ *is-protected-module* (ρ) → *execute-static-module-init-p*(*decls*, *initial*, *final*) ρ, 76 51 133
is-handled (*initial*) ∧ ¬ *is-protected-module* (ρ) → *execute-static-module-init-h*(*decls*, *initial*, *final*) ρ, 76 51 134
is-handled (*initial*) ∧ *is-protected-module* (ρ) → *execute-static-module-init-hp*(*decls*, *initial*, *final*) ρ) 76 51 134

The local modules contained in the declarations of the static module block shall be initialized in the textual order in which they occur.

NOTES

1 The use of an imported variable or procedure in the initialization body of a local module might cause an exception to occur if that variable or procedure is declared in a local module that is declared later in the same block: since local modules are initialized in the textual order in which they occur, the local module that exports the variable or procedure will not have initialized that variable or any local variables which the procedure might use.

This situation does not usually arise with variables or procedures that are imported from separate modules because the rules for initialization of separate modules guarantee that the initialization body of the module that exports the variable is executed before that of the module that imports it — provided that no separate module indirectly imports itself — see 6.1.6. It will be prevented from arising with local modules in implementations that choose the 'declare before use' model for the ordering of declarations — see 6.2.7.

2 The use of an imported variable or procedure in the initialization body of a local module might cause an exception to occur if that variable or procedure is declared (earlier or later) in the same block as the local module: since local modules are initialized before the execution of the body of the block in which they are declared, that body will not have initialized the variable or any local variables which the procedure might use.

On the execution of an initialization body that does not have an exceptional part, the initialization body shall be executed without changing the current execution state.

NOTE 3 — The current execution state will always be that of normal execution during static module initialization. If the static module initialization body has no exceptional part, the raising of an exception is a termination event — see 6.1.7.

execute-static-module-init : *Declarations* × *Block-body* × *Block-body* → *Environment* $\xrightarrow{o}$ ()

execute-static-module-init (*decls*, *initial*, *final*)ρ ≙
(*initialize-static-modules*(*decls*) ρ ; 134
if *final* ≠ nil
then *queue-termination*(*final*, ρ) 341
else skip;
execute-non-result-body(*initial*) ρ) 143

annotations After the initialization of local static modules, the finalization body is queued for execution during program termination.

If the static module is a protected module, the protection domain for that module shall be entered before the local static modules are initialized and shall be left after execution of the initialization body has completed.

If an exception is raised during execution of the static module initialization body, the protection domain shall not be left.

execute-static-module-init-p : *Declarations* × *Block-body* × *Block-body* → *Environment* $\xrightarrow{o}$ ()

execute-static-module-init-p (*decls*, *initial*, *final*)ρ ≙
def *pdom* = *activate-entry-protection*(ρ) in 51
(*initialize-static-modules*(*decls*) ρ ; 134
if *final* ≠ nil
then *queue-termination*(*final*, ρ) 341
else skip;
execute-non-result-body(*initial*) ρ; 143
activate-exit-protection(*pdom*) ρ) 51

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

If the static module initialization body has an exceptional part, completion of the initialization body in the state of exceptional execution shall be a termination event — see 6.1.7.

NOTE 4 — The raising of an exception during execution of the exceptional part of the initialization body is also a termination event.

execute-static-module-init-h : *Declarations* × *Block-body* × *Block-body* → *Environment* $\xrightarrow{o}$ ()

execute-static-module-init-h (*decls*, *initial*, *final*)ρ $\triangleq$
  (*initialize-static-modules*(*decls*) ρ ; 134
   if *final* $\neq$ nil
   then *queue-termination*(*final*, ρ) 341
   else skip;
   execute-non-result-body-h(*initial*) ρ; 143
   def *context* = *top-exception-context*() in 342
   if *is-exceptional* (*context*) 342
   then *c-exit*(EXCEPTION) ρ 345
   else skip)

If the static module is protected and the initialization body has an exceptional part then, if execution of the initialization body completes in the state of exceptional execution, the protection domain shall be left.

If an exception is raised during execution of the exceptional part of the initialization body, the protection domain shall not be left.

execute-static-module-init-hp : *Declarations* × *Block-body* × *Block-body* → *Environment* $\xrightarrow{o}$ ()

execute-static-module-init-hp (*decls*, *initial*, *final*)ρ $\triangleq$
  def *pdom* = *activate-entry-protection*(ρ) in 51
  (*initialize-static-modules*(*decls*) ρ ; 134
   if *final* $\neq$ nil
   then *queue-termination*(*final*, ρ) 341
   else skip;
   execute-non-result-body-h(*initial*) ρ; 143
   def *context* = *top-exception-context*() in 342
   (*activate-exit-protection*(*pdom*) ρ ; 51
    if *is-exceptional* (*context*) 342
    then *c-exit*(EXCEPTION) ρ 345
    else skip))

Auxiliaries

initialize-static-modules : *Declarations* → *Environment* $\xrightarrow{o}$ ()

initialize-static-modules (*decls*)ρ $\triangleq$
  if *decls* = []
  then skip
  else (let *decl* = hd *decls* in
        if *is-Module-declaration*(*decl*)
        then *initialize-a-static-module*(*decl*) ρ 134
        else skip;
        initialize-static-modules(tl *decls*) ρ) 134

initialize-a-static-module : *Module-declaration* → *Environment* $\xrightarrow{o}$ ()

initialize-a-static-module (*mk-Module-declaration* (*name*, -, -, *block*, -))ρ $\triangleq$
  let ρ_{block} = *associated-module* (*name*)ρ in 317
  execute-static-module-initialization(*block*) ρ_{block} 132

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations The environment for the execution of the block of a static module is retrieved from the given environment.

6.5.5 Static Module Finalization

The finalization body of a static module is executed on program termination – see 6.1.7. Execution is in the environment defined by the import lists and declarations of the module block.

Dynamic Semantics

On the finalization of a static module, the finalization body shall be executed.

finalize-a-static-module : *module-finalization* $\xrightarrow{o}$ ()

finalize-a-static-module (*mt*) $\triangleq$
(*is-Sourced-module-finalization*(*mt*) → *finalize-a-sourced-module*(*mt*) , 135
is-Abstract-module-finalization(*mt*) → *finalize-an-abstract-module*(*mt*)) 136

finalize-a-sourced-module : *Sourced-module-finalization* $\xrightarrow{o}$ ()

finalize-a-sourced-module (*mk-Sourced-module-finalization* (*final*, ρ)) $\triangleq$
($\neg$ *is-handled* (*final*) $\wedge$ $\neg$ *is-protected-module* (ρ) → *execute-sourced-module-final*(*final*) ρ, 76 51 135
$\neg$ *is-handled* (*final*) $\wedge$ *is-protected-module* (ρ) → *execute-sourced-module-final-p*(*final*) ρ, 76 51 135
is-handled (*final*) $\wedge$ $\neg$ *is-protected-module* (ρ) → *execute-sourced-module-final-h*(*final*) ρ, 76 51 136
is-handled (*final*) $\wedge$ *is-protected-module* (ρ) → *execute-sourced-module-final-hp*(*final*) ρ) 76 51 136

On the execution of a finalization body that does not have an exceptional part, the finalization body shall be executed without changing the current execution state.

NOTE 1 — In normal termination, the coroutine execution state is that of normal execution; in exceptional termination, the coroutine execution state is that of exceptional execution — see 6.1.7. Normal termination can switch to exceptional termination because of the raising of an exception during termination that is not handled. Exceptional termination can switch to normal termination during the execution of a finalization body that has an exception handler or that calls a procedure that has an exception handler. If the standard procedure HALT is then called, subsequent static module finalization will remain in the state of normal execution.

execute-sourced-module-final : *Block-body* → *Environment* $\xrightarrow{o}$ ()

execute-sourced-module-final (*final*)ρ $\triangleq$
execute-non-result-body(*body*) ρ; 143

If the static module is a protected module, the protection domain for that module shall be entered before the finalization body is executed, and shall be left after execution of the finalization body has completed.

If an exception is raised during execution of the static module finalization body, the protection domain shall not be left.

execute-sourced-module-final-p : *Block-body* → *Environment* $\xrightarrow{o}$ ()

execute-sourced-module-final-p (*final*)ρ $\triangleq$
def *pdom* = *activate-entry-protection*(ρ) in 51
(*execute-non-result-body*(*final*) ρ; 143
activate-exit-protection(*pdom*) ρ) 51

If the static module finalization body has an exceptional part, the finalization body shall be executed in the state of normal execution. Completion of the finalization body in the state of exceptional execution shall be a termination event — see 6.1.7.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NOTE 2 — The raising of an exception during execution of the exceptional part of the finalization body is also a termination event.

```
execute-sourced-module-final-h : Block-body → Environment ~o→ ()
execute-sourced-module-final-h (final)ρ △
  (push-exception-context(NORMAL-EXECUTION) ;                          342
   execute-non-result-body-h(final) ρ;                                 143
   def context = pop-exception-context() in                            342
   (if is-exceptional (context)                                        342
    then (swop-exception-context(context) ;                            342
          c-exit(EXCEPTION) ρ )                                        345
    else skip))
```

If the static module is protected and the finalization body has an exceptional part then, if execution of the finalization body completes in the state of exceptional execution, the protection domain shall be left.

If an exception is raised during execution of the exceptional part of the finalization body, the protection domain shall not be left.

```
execute-sourced-module-final-hp : Block-body → Environment ~o→ ()
execute-sourced-module-final-hp (final)ρ △
  def pdom = activate-entry-protection(ρ) in                           51
  (push-exception-context(NORMAL-EXECUTION) ;                          342
   execute-non-result-body-h(final) ρ;                                 143
   def context = pop-exception-context() in                            342
   (activate-exit-protection(pdom) ρ ;                                 51
    if is-exceptional (context)                                        342
    then (swop-exception-context(context) ;                            342
          c-exit(EXCEPTION) ρ )                                        345
    else skip))

finalize-an-abstract-module : Abstract-module-finalization ~o→ ()
finalize-an-abstract-module (mk-Abstract-module-finalization (final, ρ)) △
  let handler = current-exception-handler (ρ) in                       318
  final() handler
```

6.5.6 Dynamic Module Initialization

Dynamic module initialization applies to the initialization of the module blocks of local modules declared in a procedure block and recursively to the initialization of local modules declared in these module blocks — see 6.5.1 and 6.5.2.

Dynamic Semantics

The local modules contained in the declarations of a procedure block shall be initialized in the textual order in which they occur.

NOTE 1 — Since the local modules contained in the declarations of a procedure block are initialized in the textual order in which they occur, and are initialized before the procedure body is executed, the direct or indirect use of variables initialized in following local modules, or in the procedure body, might cause an exception to occur.

```
initialize-dynamic-modules : Declarations → Environment ~o→ ()
initialize-dynamic-modules (decls)ρ △
  if decls = [ ]
  then skip
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  else (let decl = hd decls in
        if is-Module-declaration(decl)
        then initialize-a-dynamic-module(decl) ρ                    137
        else skip;
        initialize-dynamic-modules(tl decls) ρ )                    136
```

```
initialize-a-dynamic-module : Module-declaration → Environment ~o~> ()
initialize-a-dynamic-module (mk-Module-declaration (name, -, -, block, -))ρ ≜
  let ρ_block = associated-module (name)ρ in                        317
  execute-dynamic-module-initialization(block) ρ_block              137
```

annotations The environment for the execution of the block of a local dynamic module is the environment constructed by the elaboration of its declarations given the imported identifiers.

```
execute-dynamic-module-initialization : Module-block → Environment ~o~> ()
execute-dynamic-module-initialization (mk-Module-block (decls, initial, -))ρ ≜
  ( ¬ is-handled (initial) ∧ ¬ is-protected-module (ρ) → execute-dynamic-module-init(decls, initial) ρ,    76 51 137
    ¬ is-handled (initial) ∧ is-protected-module (ρ)   → execute-dynamic-module-init-p(decls, initial) ρ,  76 51 137
    is-handled (initial) ∧ ¬ is-protected-module (ρ)   → execute-dynamic-module-init-h(decls, initial) ρ,  76 51 137
    is-handled (initial) ∧ is-protected-module (ρ)     → execute-dynamic-module-init-hp(decls, initial) ρ  76 51 137
  )
```

The local modules contained in the declarations of the dynamic module block shall be initialized in the textual order in which they occur.

An initialization body that does not have an exceptional part shall be executed without changing the current execution state.

NOTE 2 — If the initialization body of a dynamic module has no exceptional part, the exceptional part that is the current exception handler is the exceptional part that is the current exception handler for the caller of the procedure whose dynamic modules are being initialized — see 6.5.1 and 6.5.2.

```
execute-dynamic-module-init : Declarations × Block-body → Environment ~o~> ()
execute-dynamic-module-init (decls, initial)ρ ≜
  (initialize-dynamic-modules(decls) ρ ;                            136
   execute-non-result-body(initial) ρ )                             143
```

If the dynamic module is a protected module, the protection domain for that module shall be entered before the local dynamic modules are initialized, and shall be left after execution of the initialization body has completed.

If an exception is raised during execution of the dynamic module initialization body, the protection domain shall not be left.

```
execute-dynamic-module-init-p : Declarations × Block-body → Environment ~o~> ()
execute-dynamic-module-init-p (decls, initial)ρ ≜
  def pdom = activate-entry-protection(ρ) in                        51
  (initialize-dynamic-modules(decls) ρ ;                            136
   execute-non-result-body(initial) ρ;                              143
   activate-exit-protection(pdom) ρ ;                               51
   )
```

If the initialization body of the dynamic module has an exceptional part, the initialization body shall be executed in the state of normal execution.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

On completion of the execution of the block body, the execution state shall revert to that which applied before the execution of the block body.

If execution of the initialization body completed in the state of exceptional execution, the exception shall be re-raised.

```
execute-dynamic-module-init-h : Declarations × Block-body → Environment ->o ()
execute-dynamic-module-init-h (decls, initial)ρ ≜
  (initialize-dynamic-modules(decls) ρ ;                          136
   push-exception-context(NORMAL-EXECUTION) ;                     342
   execute-non-result-body-h(initial) ρ;                          143
   def context = pop-exception-context() in                       342
   if is-exceptional (context)                                    342
   then (swop-exception-context(context) ;                        342
         c-exit(EXCEPTION) ρ )                                    345
   else skip)
```

On the initialization of a dynamic module that is protected and whose initialization body has an exceptional part, if execution of the initialization body completes in the state of exceptional execution, the exception shall be re-raised after the protection domain has been left.

If an exception is raised during execution of the exceptional part of the initialization body, the protection domain shall not be left.

```
execute-dynamic-module-init-hp : Declarations × Block-body → Environment ->o ()
execute-dynamic-module-init-hp (decls, initial)ρ ≜
  def pdom = activate-entry-protection(ρ) in                      51
  (initialize-dynamic-modules(decls) ρ ;                          136
   push-exception-context(NORMAL-EXECUTION) ;                     342
   if initial ≠ nil
   then let mk-Block-body (normal, except) = initial in
        execute-normal-except(normal, except) ρ                   143
   else skip;
   def context = pop-exception-context() in                       342
   (activate-exit-protection(pdom) ρ ;                            51
    if is-exceptional (context)                                   342
    then (swop-exception-context(context) ;                       342
          c-exit(EXCEPTION) ρ )                                   345
    else skip))
```

6.5.7 Dynamic Module Finalization

Dynamic module finalization applies to the finalization of the module blocks of local modules declared in a procedure block and recursively to the finalization of local modules declared in these module blocks — see 6.5.1 and 6.5.2.

Dynamic Semantics

The local modules contained in the declarations of a procedure block shall be finalized in the reverse order of their initialization.

```
finalize-dynamic-modules : Declarations → Environment ->o ()
finalize-dynamic-modules (decls)ρ ≜
  if decls = []
  then skip
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  else (finalize-dynamic-modules(tl decls) ρ;                                    138
        let decl = hd decls in
        if is-Module-declaration(decl)
        then finalize-a-dynamic-module(decl) ρ                                   139
        else skip)
```

```
finalize-a-dynamic-module : Module-declaration → Environment ~o~> ()
finalize-a-dynamic-module (mk-Module-declaration (name, -, -, block, -))ρ △
  let ρ_block = associated-module (name)ρ in                                     317
  execute-dynamic-module-finalization(block) ρ_block                             139
```

annotations The environment for the execution of the block of a local dynamic module is the environment constructed by the elaboration of its declarations given the imported identifiers.

```
execute-dynamic-module-finalization : Module-block → Environment ~o~> ()
execute-dynamic-module-finalization (mk-Module-block (decls, -, final))ρ △
  if final ≠ nil
  then ( ¬ is-handled (final) ∧ ¬ is-protected-module (ρ) → execute-dynamic-module-final(decls, final) ρ,        76 51 139
         ¬ is-handled (final) ∧ is-protected-module (ρ)   → execute-dynamic-module-final-p(decls, final) ρ,      76 51 139
         is-handled (final) ∧ ¬ is-protected-module (ρ)   → execute-dynamic-module-final-h(decls, final) ρ,      76 51 140
         is-handled (final) ∧ is-protected-module (ρ)     → execute-dynamic-module-final-hp(decls, final) ρ)    76 51 140
  else skip
```

A finalization body that does not have an exceptional part shall be executed without changing the current execution state.

NOTE — If the finalization body of a dynamic module has no exceptional part, the exceptional part that is the current exception handler is the exceptional part that is the current exception handler for the caller of the procedure whose dynamic modules are being finalized — see 6.5.1 and 6.5.2.

The local modules contained in the declarations of the dynamic module block shall be finalized in the reverse order of their initialization.

```
execute-dynamic-module-final : Declarations × Block-body → Environment ~o~> ()
execute-dynamic-module-final (decls, final)ρ △
  (execute-non-result-body(final) ρ;                                             143
   finalize-dynamic-modules(decls) ρ )                                           138
```

If the dynamic module is a protected module, the protection domain for that module shall be entered before the finalization body is executed, and shall be left after the local modules contained in the declarations of the dynamic module block have been finalized.

If an exception is raised during execution of the dynamic module finalization body, the protection domain shall not be left.

```
execute-dynamic-module-final-p : Declarations × Block-body → Environment ~o~> ()
execute-dynamic-module-final-p (decls, final)ρ △
  def pdom = activate-entry-protection(ρ) in                                     51
  (execute-non-result-body(final) ρ;                                             143
   finalize-dynamic-modules(decls) ρ;                                            138
   activate-exit-protection(pdom) ρ )                                            51
```

If the finalization body of the dynamic module has an exceptional part, the finalization body shall be executed in the state of normal execution.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

On completion of the execution of the block body, the execution state shall revert to that which applied before the execution of the block body.

If execution of the finalization body completed in the state of exceptional execution, the exception shall be re-raised after the local modules contained in the declarations of the dynamic module block have been finalized.

If an exception is raised during execution of the exceptional part of the finalization body, the local modules contained in the declarations of the dynamic module block shall not be finalized.

```
execute-dynamic-module-final-h : Declarations × Block-body → Environment ~o→ ()
execute-dynamic-module-final-h (decls, final)ρ ≜
  (push-exception-context(NORMAL-EXECUTION) ;                          342
   execute-non-result-body-h(final) ρ;                                 143
   def context = pop-exception-context() in                            342
   (finalize-dynamic-modules(decls) ρ;                                 138
    if is-exceptional (context)                                        342
    then (swop-exception-context(context) ;                            342
          c-exit(EXCEPTION) ρ )                                        345
    else skip))
```

On the finalization of a dynamic module that is protected and whose finalization body has an exceptional part, if execution of the finalization body completes in the state of exceptional execution, the exception shall be re-raised after the protection domain has been left.

If an exception is raised during execution of the exceptional part of the finalization body, the protection domain shall not be left.

```
execute-dynamic-module-final-hp : Declarations × Block-body → Environment ~o→ ()
execute-dynamic-module-final-hp (decls, final)ρ ≜
  def pdom = activate-entry-protection(ρ) in                           51
  (push-exception-context(NORMAL-EXECUTION) ;                          342
   if final ≠ nil
   then let mk-Block-body (normal, except) = final in
        execute-normal-except(normal, except) ρ  else skip;            143
   def context = pop-exception-context() in                            342
   (finalize-dynamic-modules(decls) ρ;                                 138
    activate-exit-protection(pdom) ρ ;                                 51
    if is-exceptional (context)                                        342
    then (swop-exception-context(context) ;                            342
          c-exit(EXCEPTION) ρ )                                        345
    else skip))
```

6.5.8 Block Bodies and Exception Handling

Block bodies contain the statements that are to be executed when a module is initialized or finalized, or when a procedure is activated.

The block bodies of function procedures return a result value, whereas the block bodies of modules and of proper procedures do not return a result value.

Block bodies have a normal part and, optionally, an exceptional part to handle exceptions raised during execution of the normal part.

Execution of a coroutine takes place in normal execution state or in exceptional execution state. Initially, in each coroutine, the execution state is that of normal execution, and no exceptional part is selected as the current exception

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

handler.

If a block body has an exceptional part, that exceptional part becomes the new current exception handler during execution of the normal part.

If an exception is raised in the state of normal execution, the execution state becomes that of exceptional execution. If there is an exceptional part that is the current exception handler for the coroutine, that exceptional part is executed in the state of exceptional execution.

The execution of a retry statement within the exceptional part of a block body causes the normal part to be retried, in the state of normal execution.

The execution of a return statement within the exceptional part of a block body causes normal completion of execution of the block body.

If an exception is raised in the state of exceptional execution, it is re-raised in the calling context; finalization of local dynamic modules will not take place, and the protection domain, if any, will not be left.

Exceptional completion occurs if there is no return or retry statement executed in the exceptional part; the finalization of local dynamic modules will take place, and the protection domain, if any, will be left. The exception will then be re-raised in the calling context.

If there is no exceptional part selected as the current exception handler for the coroutine, the raising of the exception is a termination event — see 6.1.7. In that case, the subsequent execution of the finalization bodies of the static modules of the program will take place in the state of exceptional execution — see 6.5.5.

EXAMPLE — Exception handling.

In the following example, use is made of the example module `GeneralUserExceptions` — see 7.3.4.4.

```
FROM LibModule IMPORT
  Fly, ReplaceRubberBand, SupplyRubberBands, IsLibException, LibException, LibExceptions;
FROM GeneralUserExceptions IMPORT
  GeneralExceptions, RaiseGeneralException;

PROCEDURE TryFlying();
BEGIN
  Fly
EXCEPT
  (* statements in exceptional execution *)
  IF IsLibException() THEN
    CASE LibException() OF
    |brokenRubberBand:
      ReplaceRubberBand;
      RETRY
    ELSE
    END
  END
  (* Exceptional return re-raises the exception *)
END TryFlying;

PROCEDURE KeepFlying();
BEGIN
  (* statements in normal execution *)
  TryFlying;
  (* statements in normal execution *)
EXCEPT
  (* statements in exceptional execution *)
  IF IsLibException() THEN
    CASE LibException() OF
    |outOfRubberBands:
      SupplyRubberBands;
      RETRY
    ELSE
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
        (* can keep flying no longer - normal return *)
        RETURN
      END
    ELSE
      RaiseGeneralException(disaster, "Unexpected exception from LibModule")
    END
  END KeepFlying;
```

Concrete Syntax

block body = normal part, ["EXCEPT", exceptional part] ;

normal part = statement sequence ;

exceptional part = statement sequence ;

Abstract Syntax

types

Block-body :: *normal* : *Statement-sequence*
except : [*Statement-sequence*]

annotations In the translation from concrete to abstract syntax, a block body that has no exceptional part shall be represented by a *Block-body* with a nil value for the *except* component.

Static Semantics

The control variable of any for statement contained within a block body shall be declared as a simple variable within the declarations of the block body.

Any exit statements contained within the block body shall be contained within a loop statement inside the statement part.

There shall be no retry statements within the normal part of a block body.

$wf\text{-}block\text{-}body : Declarations \times Block\text{-}body \to Environment \to \mathbb{B}$

$wf\text{-}block\text{-}body\,(decls, mk\text{-}Block\text{-}body\,(normal, except))\rho \triangleq$
$wf\text{-}statement\text{-}sequence\,(normal)\rho \wedge$ 147
$(except = \mathsf{nil} \vee wf\text{-}statement\text{-}sequence\,(except)\rho) \wedge$ 147
$wf\text{-}control\text{-}variables\,(decls, normal)\rho \wedge$ 170
$(except = \mathsf{nil} \vee wf\text{-}control\text{-}variables\,(decls, except)\rho) \wedge$ 170
$wf\text{-}exit\text{-}statements\,(normal)\rho \wedge$ 177
$(except = \mathsf{nil} \vee wf\text{-}exit\text{-}statements\,(except)\rho) \wedge$ 177
$no\text{-}retry\text{-}statements\,(normal)\rho$ 177

6.5.8.1 Non-result Bodies

Non-result bodies are the block bodies of proper procedure blocks or the initialization or finalization bodies of module blocks.

Static Semantics

Any return statements contained within non-result bodies shall be simple return statements.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wf-non-result-body : *Block-body* → *Environment* → $\mathbb{B}$

wf-non-result-body (*mk-Block-body* (*normal*, *except*))ρ $\triangleq$
  wf-simple-return-statements (*normal*)ρ ∧ 176
  (*except* = nil ∨ *wf-simple-return-statements* (*except*)ρ) 176

Dynamic Semantics

If there is no exceptional part to a non-result body, the execution of such a body shall cause execution of the normal part with no change to the current exception handler.

execute-non-result-body : *Block-body* → *Environment* $\xrightarrow{o}$ ()

execute-non-result-body (*body*)ρ $\triangleq$
  if *body* ≠ nil
  then let *mk-Block-body* (*normal*, -) = *body* in
    execute-normal(*normal*) ρ 143
  else skip

The execution of a simple return statement contained in the normal part shall cause the execution of the block body to terminate prematurely.

execute-normal : *Statement-sequence* → *Environment* $\xrightarrow{o}$ ()

execute-normal (*normal*)ρ $\triangleq$
  def ρ_{cont} = *c-tixe*({RETURN ↦ skip}) ρ in 345
  m-statement-sequence(*normal*) ρ_{cont} 148

If there is an exceptional part to a non-result body, the execution of such a body shall cause execution of the normal part with that exceptional part as the new current exception handler.

execute-non-result-body-h : *Block-body* → *Environment* $\xrightarrow{o}$ ()

execute-non-result-body-h (*body*)ρ $\triangleq$
  if *body* ≠ nil
  then let *mk-Block-body* (*normal*, *except*) = *body* in
    execute-normal-except(*normal*, *except*) ρ 143
  else skip

The execution of a simple return statement contained in the normal part shall cause the execution of the block body to complete prematurely.

The raising of an exception during execution of the normal part shall cause execution of the exceptional part in the state of exceptional execution.

execute-normal-except : *Statement-sequence* × *Statement-sequence* → *Environment* $\xrightarrow{o}$ ()

execute-normal-except (*normal*, *except*)ρ $\triangleq$
  def ρ_{cont} = *c-tixe*({RETURN ↦ skip, 345
    EXCEPTION ↦ *execute-except*(*except*, *normal*) ρ}) ρ in 144
  m-statement-sequence(*normal*) ρ_{cont} 148

The execution of a simple return statement contained in the exceptional part shall cause the execution of the block body to complete prematurely in the state of normal execution.

The execution of a retry statement contained in the exceptional part shall cause reversion to the state of normal execution and cause the normal part to be retried.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NOTE — The exceptional part will again be the handler for any exceptions raised during execution of the normal part.

If an exception is raised during execution of the exceptional part, it shall replace the exception that was being handled and shall be raised in the calling context.

If execution reaches the end of the exceptional part, execution of the handled body shall complete in the state of exceptional execution.

execute-except : *Statement-sequence* × *Statement-sequence* → *Environment* $\xrightarrow{o}$ ()

execute-except (*except*, *normal*)ρ $\triangleq$
  def ρ_{cont} = *c-tixe*({RETURN ↦ *swop-exception-context*(NORMAL-EXECUTION) , 345 342
    RETRY ↦ (*swop-exception-context*(NORMAL-EXECUTION) ; 342
      execute-normal-except(*normal*, *except*) ρ), 143
    EXCEPTION ↦ def *context* = *pop-exception-context*() in (*swop-exception-context*(*context*) ; 342 342
      c-exit(EXCEPTION) ρ)}) ρ in 345
  m-statement-sequence(*except*) ρ_{cont} 148

6.5.8.2 Result Bodies

Result bodies are the block bodies of function procedure blocks.

Static Semantics

The return statements of result-bodies shall be function return statements returning a value of the result type of the function. The normal part of a block body shall have at least one function return statement.

wf-result-body : *Typed* × *Block-body* → *Environment* → $\mathbb{B}$

wf-result-body (*rtype*, *mk-Block-body* (*normal*, *except*))ρ $\triangleq$
  wf-function-return-statements (*rtype*, *normal*)ρ ∧ 176
  (*except* = nil ∨ *wf-function-return-statements* (*rtype*, *except*)ρ ∨ *return-statements-of* (*except*) = { }) 176 177

Dynamic Semantics

If there is no exceptional part to a non-result body, the execution of such a body shall cause execution of the normal part with no change to the current exception handler.

execute-result-body : *Block-body* → *Environment* $\xrightarrow{o}$ *Value*

execute-result-body (*body*)ρ $\triangleq$
  let *mk-Block-body* (*normal*, -) = *body* in
  execute-function-normal(*normal*) ρ 144

The execution of a function return statement contained in the normal part shall cause the execution of the block body to terminate prematurely. The value of the function return statement expression shall be the value that is returned as the value of the call.

It shall be an exception if the completion of the normal part of a result body is not by the execution of a return statement.

execute-function-normal : *Statement-sequence* → *Environment* $\xrightarrow{o}$ ()

execute-function-normal (*normal*)ρ $\triangleq$
  def ρ_{cont} = *c-tixe*({RETURN ↦ λ *result* : *Value* · return *result*}) ρ in 345
  (*m-statement-sequence*(*normal*) ρ_{cont} ; 148
   let *handler* = *current-exception-handler* (ρ) in *mandatory-detection*(NORETURN) *handler*) 318 396

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

If there is an exceptional part to a result body, the execution of such a body shall cause execution of the normal part with that exceptional part as the new current exception handler.

execute-result-body-h : *Block-body* → *Environment* $\xrightarrow{o}$ *Value*

execute-result-body-h (*body*)ρ ≙
  let *mk-Block-body* (*normal*, *except*) = *body* in
  execute-function-normal-except(*normal*, *except*) ρ 145

The execution of a function return statement contained in the normal part shall cause the execution of the block body to complete prematurely. The value of the function return statement expression shall be the value that is returned as the value of the call.

The raising of an exception during execution of the normal part shall cause execution of the exceptional part in the state of exceptional execution.

execute-function-normal-except : *Statement-sequence* × *Statement-sequence* → *Environment* $\xrightarrow{o}$ ()

execute-function-normal-except (*normal*, *except*)ρ ≙
  def ρ_{cont} = *c-tixe*({RETURN ↦ λ *result* : *Value* · return *result*, 345
    EXCEPTION ↦ *execute-function-except*(*except*, *normal*) ρ}) ρ in 145
  (*m-statement-sequence*(*normal*) ρ_{cont} ; 148
   let *handler* = *current-exception-handler* (ρ) in *mandatory-detection*(NORETURN) *handler*) 318 396

The execution of a function return statement contained in the exceptional part shall cause the execution of the block body to complete prematurely in the state of normal execution. The value of the function return statement expression shall be the value that is returned as the value of the call.

The execution of a retry statement contained in the exceptional part shall cause reversion to the state of normal execution, and cause the normal part to be retried.

NOTE — The exceptional part will again be the handler for any exceptions raised during execution of the normal part.

If an exception is raised during execution of the exceptional part, it shall replace the exception that was being handled, and shall be raised in the calling context.

If execution reaches the end of the exceptional part, execution of the handled body shall complete in the state of exceptional execution.

execute-function-except : *Statement-sequence* × *Statement-sequence* → *Environment* $\xrightarrow{o}$ ()

execute-function-except (*except*, *normal*)ρ ≙
  def ρ_{cont} = *c-tixe*({RETURN ↦ λ *result* : *Value* · (*swop-exception-context*(NORMAL-EXECUTION) ; return *result*) 345 342
    RETRY ↦ (*swop-exception-context*(NORMAL-EXECUTION) ; 342
      execute-function-normal-except(*normal*, *except*) ρ); 145
    EXCEPTION ↦ def *context* = *pop-exception-context*() in (*swop-exception-context*(*context*) ; 342 342
      c-exit(EXCEPTION) ρ)}) ρ in 345
  (*m-statement-sequence*(*except*) ρ_{cont} ; 148
   return nil)

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.6 Statements

Except for the empty statement, statements denote actions. Statements are executed and their execution has an effect that is a transformation of the program state. The empty statement, the assignment statement, the procedure call, the return statement, the retry statement, and the exit statement are not composed of any parts that are themselves statements. The remaining statements have components that are themselves statements; these are the with statement, the if statement, the case statement, the while statement, the repeat statement, the loop statement, and the for statement.

Concrete Syntax

```
statement =
    empty statement   | assignment statement | procedure call | return statement |
    retry statement   | with statement       | if statement   | case statement   |
    while statement   | repeat statement     | loop statement | exit statement   |
    for statement ;
```

Abstract Syntax

types

$$
\begin{aligned}
\mathit{Statement} = {} & \mathit{Empty\text{-}statement} \\
& \mid \mathit{Assignment\text{-}statement} \\
& \mid \mathit{Procedure\text{-}call} \\
& \mid \mathit{Return\text{-}statement} \\
& \mid \mathit{Retry\text{-}statement} \\
& \mid \mathit{With\text{-}statement} \\
& \mid \mathit{If\text{-}statement} \\
& \mid \mathit{Case\text{-}statement} \\
& \mid \mathit{While\text{-}statement} \\
& \mid \mathit{Repeat\text{-}statement} \\
& \mid \mathit{Loop\text{-}statement} \\
& \mid \mathit{Exit\text{-}statement} \\
& \mid \mathit{For\text{-}statement}
\end{aligned}
$$

Static Semantics

$$\mathit{wf\text{-}statement} : \mathit{Statement} \to \mathit{Environment} \to \mathbb{B}$$

$$
\begin{aligned}
&\mathit{wf\text{-}statement}\,(st)\rho \triangleq \\
&\quad (\mathit{is\text{-}Empty\text{-}statement}(st) && \to \mathit{wf\text{-}empty\text{-}statement}\,(st)\rho, && 148\\
&\quad \mathit{is\text{-}Assignment\text{-}statement}(st) && \to \mathit{wf\text{-}assignment\text{-}statement}\,(st)\rho, && 149\\
&\quad \mathit{is\text{-}Procedure\text{-}call}(st) && \to \mathit{wf\text{-}procedure\text{-}call}\,(st)\rho, && 152\\
&\quad \mathit{is\text{-}Return\text{-}statement}(st) && \to \mathit{wf\text{-}return\text{-}statement}\,(st)\rho, && 153\\
&\quad \mathit{is\text{-}Retry\text{-}statement}(st) && \to \mathit{wf\text{-}retry\text{-}statement}\,(st)\rho, && 156\\
&\quad \mathit{is\text{-}With\text{-}statement}(st) && \to \mathit{wf\text{-}with\text{-}statement}\,(st)\rho, && 156\\
&\quad \mathit{is\text{-}If\text{-}statement}(st) && \to \mathit{wf\text{-}if\text{-}statement}\,(st)\rho, && 158\\
&\quad \mathit{is\text{-}Case\text{-}statement}(st) && \to \mathit{wf\text{-}case\text{-}statement}\,(st)\rho, && 160\\
&\quad \mathit{is\text{-}While\text{-}statement}(st) && \to \mathit{wf\text{-}while\text{-}statement}\,(st)\rho, && 162\\
&\quad \mathit{is\text{-}Repeat\text{-}statement}(st) && \to \mathit{wf\text{-}repeat\text{-}statement}\,(st)\rho, && 163\\
&\quad \mathit{is\text{-}Loop\text{-}statement}(st) && \to \mathit{wf\text{-}loop\text{-}statement}\,(st)\rho, && 164\\
&\quad \mathit{is\text{-}Exit\text{-}statement}(st) && \to \mathit{wf\text{-}exit\text{-}statement}\,(st)\rho, && 164\\
&\quad \mathit{is\text{-}For\text{-}statement}(st) && \to \mathit{wf\text{-}for\text{-}statement}\,(st)\rho) && 165
\end{aligned}
$$

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Dynamic Semantics

$m\text{-}statement : Statement \to Environment \xrightarrow{o} ()$

$m\text{-}statement\,(st)\rho \triangleq$
$\quad (\ is\text{-}Empty\text{-}statement(st) \to m\text{-}empty\text{-}statement(st)\,\rho,$ 148
$\quad\ is\text{-}Assignment\text{-}statement(st) \to m\text{-}assignment\text{-}statement(st)\,\rho,$ 149
$\quad\ is\text{-}Procedure\text{-}call(st) \to m\text{-}procedure\text{-}call(st)\,\rho,$ 153
$\quad\ is\text{-}Return\text{-}statement(st) \to m\text{-}return\text{-}statement(st)\,\rho,$ 153
$\quad\ is\text{-}Retry\text{-}statement(st) \to m\text{-}retry\text{-}statement(st)\,\rho,$ 156
$\quad\ is\text{-}With\text{-}statement(st) \to m\text{-}with\text{-}statement(st)\,\rho,$ 157
$\quad\ is\text{-}If\text{-}statement(st) \to m\text{-}if\text{-}statement(st)\,\rho,$ 158
$\quad\ is\text{-}Case\text{-}statement(st) \to m\text{-}case\text{-}statement(st)\,\rho,$ 160
$\quad\ is\text{-}While\text{-}statement(st) \to m\text{-}while\text{-}statement(st)\,\rho,$ 162
$\quad\ is\text{-}Repeat\text{-}statement(st) \to m\text{-}repeat\text{-}statement(st)\,\rho,$ 163
$\quad\ is\text{-}Loop\text{-}statement(st) \to m\text{-}loop\text{-}statement(st)\,\rho,$ 164
$\quad\ is\text{-}Exit\text{-}statement(st) \to m\text{-}exit\text{-}statement(st)\,\rho,$ 164
$\quad\ is\text{-}For\text{-}statement(st) \to m\text{-}for\text{-}statement(st)\,\rho)$ 167

6.6.1 Statement Sequences

A statement sequence specifies that a sequence of actions is to be performed. Each action is specified by a statement. The statements of a statement sequence are separated by semicolons.

Concrete Syntax

statement sequence = statement, { semicolon, statement } ;

Abstract Syntax

types

$Statement\text{-}sequence :: actions : Statement^{+}$

Static Semantics

$wf\text{-}statement\text{-}sequence : Statement\text{-}sequence \to Environment \to \mathbb{B}$

$wf\text{-}statement\text{-}sequence\,(mk\text{-}Statement\text{-}sequence\,(actions))\rho \triangleq$
$\quad \forall\, action \in \text{elems}\ actions \cdot wf\text{-}statement\,(action)\rho$ 146

annotations Check each statement of the statement sequence.

Dynamic Semantics

The execution of a statement sequence shall cause the execution of the first statement in the sequence, followed by the execution, in order, of any remaining statements of the sequence.

NOTES

1 The execution of a statement sequence may be prematurely terminated by the execution of a return statement, or by the execution of an exit statement if the statement sequence is contained within a loop statement.

2 The execution of a statement sequence may be prematurely terminated if an exception is raised.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

m-statement-sequence : *Statement-sequence* → *Environment* $\xrightarrow{o}$ ()
m-statement-sequence (*mk-Statement-sequence* (*actions*))ρ $\triangleq$
  let [*first*] $\curvearrowright$ *rest* = *actions* in
  (*m-statement*(*first*) ρ ; 147
   if *rest* $\neq$ []
   then *m-statement-sequence*(*rest*) ρ 148
   else skip)

6.6.2 Empty Statements

An empty statement contains no symbols and denotes no action. Its use permits the relaxation of punctuation rules in statement sequences.

Concrete Syntax

empty statement = ;

Abstract Syntax

types

Empty-statement ::

Static Semantics

wf-empty-statement : *Empty-statement* → *Environment* → $\mathbb{B}$
wf-empty-statement (*mk-Empty-statement* ())ρ $\triangleq$
  true

Dynamic Semantics

An empty statement shall denote no action; its execution shall produce no changes to the state.

m-empty-statement : *Empty-statement* → *Environment* $\xrightarrow{o}$ ()
m-empty-statement (*mk-Empty-statement* ())ρ $\triangleq$
  skip

6.6.3 Assignment Statements

An assignment statement specifies that a variable is to be given a value. The variable may be either an elementary variable or a structured (composite) variable.

Concrete Syntax

assignment statement = variable designator, assignment operator, expression ;

Abstract Syntax

types

Assignment-statement :: *desig* : *Variable-designator*
  expr : *Expression*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Static Semantics

The variable designator shall denote a variable. The type of the variable designator shall be assignment-compatible with the type of the expression — see 6.4.2.

wf-assignment-statement : *Assignment-statement* → *Environment* → $\mathbb{B}$

wf-assignment-statement (*mk-Assignment-statement* (*desig*, *expr*))ρ $\triangleq$
  wf-variable-designator (*desig*)ρ $\wedge$ 182
  wf-expression (*expr*)ρ $\wedge$ 190
  is-assignment-compatible (*t-variable-designator* (*desig*)ρ, *t-expression* (*expr*)ρ)ρ 123 182 1

Dynamic Semantics

The execution of an assignment statement shall cause the expression to be evaluated and the resulting value to become the new value of the variable denoted by the variable designator. The order of evaluation of the variable designator and the expression shall be implementation-dependent.

If the assignment is to an elementary variable, and the value to be assigned is not a value of the range of values defined by the type of the variable designator, an exception shall be raised.

If the assignment is of a string constant or string literal to an array whose component type is the character type, and the string constant does not fill the array, the string shall be terminated with a string terminator, and the remainder of the array shall become undefined.

NOTE 1 — For assignment to a concrete string variable to work as described, it is not sufficient for the variable's component type to be a subrange of **CHAR**. It must be identical to **CHAR** so that it can contain a string terminator.

If the assignment is to a tag field of a record, and the new value of the tag field causes a new variant to be selected, the value of each field of the field list of the variant associated with the old value of the tag field shall become undefined. If the new value of the tag field does not cause a new variant to be selected, the assignment to the tag field shall produce no change to the associated variant.

NOTES

2 The evaluations of the variable designator and the expression could produce a transformation of the program state, i.e. have side-effects.

3 An exception occurs during the evaluation of the expression if the value of an elementary variable that is a factor of that expression is undefined. If the value of a variable of a record or array structure is to be assigned, and the value of any of the components is undefined, then no exception occurs. Undefined values can therefore be assigned to the components of a structured (composite) variable, but only if the assignment is applied to the entire variable.

4 An exception occurs during the evaluation of the variable designator if the variable designator is a selected designator and the assignment is to a field of a variant for which there is an explicit tag field and the variant is inactive — see 6.7.3.

5 An exception occurs if assignment is to a variable whose designator is an indexed designator and the index is not a valid value of the variable's index type.

m-assignment-statement : *Assignment-statement* → *Environment* $\xrightarrow{o}$ ()

m-assignment-statement (*mk-Assignment-statement* (*desig*, *expr*))ρ $\triangleq$
  def *lhs* = *m-variable-designator*(*desig*) ρ, 182
    rhs = *m-expression*(*expr*) ρ in 190
  let *handler* = *current-exception-handler* (ρ) in 318
  assign(*lhs*, *rhs*) *handler* 150

annotations The above evaluation order for the variable designator and the expression is just one of those permitted; the expression may instead be evaluated before the variable designator, or both may be evaluated in parallel — see 4.1.

After the two evaluations, the result of the expression evaluation is assigned to the target of the assignment.

When the variable designator is a selected designator and the target is a field of a variant, then in the case of explicit tag fields, the evaluation of the selected designator will check that the values of the tag fields that select the field are correctly

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

set. In the case of implicit tag fields selecting the variant, these implicit tag fields will be set to appropriate values.

Auxiliaries

assign : *Variable* × *Value* → *Exception-handler* $\xrightarrow{o}$ ()

assign (*var*, *val*)*handler* $\triangleq$
  cases *var*:
    mk-Elementary-variable (-, -) → *elementary-variable-assignment*(*var*, *val*) *handler*, 150
    mk-Array-variable (-) → *array-variable-assignment*(*var*, *val*) *handler*, 150
    mk-Record-variable (-) → *record-variable-assignment*(*var*, *val*) *handler*, 151
    mk-Tag-variable (-, -) → *tag-variable-assignment*(*var*, *val*) *handler*, 151
    mk-Variant-variable (-, -, -) → *variant-variable-assignment*(*var*, *val*) *handler*, 151
    mk-Tagged-variable (-, -, -) → *tagged-variable-assignment*(*var*, *val*) *handler* 151
  end

annotations This function is used to assign to elementary variables, and is used recursively to assign to array, record, tag, variant, and tagged variables. An undefined value can only occur as part of an array or record value; it cannot occur as an elementary value.

elementary-variable-assignment : *Elementary-variable* × *Value* → *Exception-handler* $\xrightarrow{o}$ ()

elementary-variable-assignment (*var*, *oval*)*handler* $\triangleq$
  let *mk-Elementary-variable* (*loc*, *range*) = *var* in
  let *val* = if *oval* : *String-value* then *to-char* (*oval*) else *oval* in 253
  if *range* = nil
  then *change-value*(*loc*, *val*) 335
  elseif *val* ∈ *range* ∨ *val* = UNDEFINED
  then *change-value*(*loc*, *val*) 335
  else *mandatory-detection*(ASSIGN-RANGE) *handler* 396

annotations Check that the value to be assigned is within range or is undefined: if it is, change the value of the target designator; otherwise an exception shall be raised. Character literals are interpreted as values of the character type.

array-variable-assignment : *Array-variable* × *Value* → *Exception-handler* $\xrightarrow{o}$ ()

array-variable-assignment (*var*, *val*)*handler* $\triangleq$
  if *val* : *String-value*
  then *array-string-assignment*(*var*, *val*) *handler* 150
  else *array-value-assignment*(*var*, *val*) *handler* 150

array-value-assignment : *Array-variable* × *Value* → *Exception-handler* $\xrightarrow{o}$ ()

array-value-assignment (*var*, *val*)*handler* $\triangleq$
  for all *i* ∈ dom *var*
  do *assign*(*var*(*i*), *val*(*i*)) *handler* 150

annotations Component by component assignment of the array value to the array variable occurs, and assignment of an undefined value is allowed for.

array-string-assignment : *Array-variable* × *String-value* → *Exception-handler* $\xrightarrow{o}$ ()

array-string-assignment (*var*, *val*)*handler* $\triangleq$
  let *sval* : $\mathbb{N} \xrightarrow{m}$ char be st
      dom *sval* = dom *var* ∧
      ∀ *i* ∈ dom *sval* ·
          i < len *val* ∧ *sval* (*i*) = *val* (*i* + 1) ∨
          i = len *val* ∧ *sval* (*i*) = END-OF-STRING-CHAR ∨
          i > len *val* ∧ *sval* (*i*) = UNDEFINED in
  array-value-assignment(*var*, *sval*) *handler* 150

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations If the assignment is a string literal to an array of CHAR, and the string constant does not fill the array, the string is terminated with a string terminator (denoted by END-OF-STRING-CHAR) and the remainder of the array is set to undefined. The well-formed conditions guarantee that the length of the string constant is less than or equal to the number of the components of the array.

record-variable-assignment : *Record-variable* × *Value* → *Exception-handler* $\xrightarrow{o}$ ()

record-variable-assignment (*var*, *val*)*handler* $\triangleq$
  for all *id* ∈ dom *var*
  do *assign*(*var*(*id*), *val*(*id*)) *handler* 150

annotations Assignment is component by component, and assignment of an undefined value is allowed for.

tag-variable-assignment : *Tag-variable* × *Value* → *Exception-handler* $\xrightarrow{o}$ ()

tag-variable-assignment (*var*, *val*)*handler* $\triangleq$
  let *mk-Tag-variable* (*tvar*, *variants*) = *var* in
  def *tval* = *variable-value*(*tvar*) in 224
  def *current* = *tagged-variant*(*tval*, *variants*) , 151
    new = *tagged-variant*(*val*, *variants*) *handler* in 151
  if *current* ≠ *new*
  then (*set-to-undefined*(*current*) ; 151
    assign(*tvar*, *val*) *handler*) 150
  else *assign*(*tvar*, *val*) 150

annotations If the assignment is to a tag field of a record and the new value of the tag field causes a new variant to be selected, the variables of the field list of the variant associated with the old value of the tag field are set to undefined. If the new value of the tag field does not cause a new variant to be selected, the assignment to the tag field produces no change to the associated variant.

set-to-undefined : *Record-components* $\xrightarrow{o}$ ()

set-to-undefined (*field*) $\triangleq$
  let *locs* = *locs-of* (*field*) in 338
  for all *loc* ∈ *locs*
  do *change-value*(*loc*, UNDEFINED) 335

annotations Set the variables that are components of a variant to undefined.

tagged-variant : *Value* × *Variant-component*-set → *Exception-handler* $\xrightarrow{o}$ *Record-components*

tagged-variant (*val*, *variants*)*handler* $\triangleq$
  if *val* ∈ $\bigcup$ {*v.labels* | *v* ∈ *variants*}
  then let *v* ∈ *variants* be st *val* ∈ *v.labels* in
    return *v.fields*
  elseif *val* = UNDEFINED
  then return { }

annotations Return the (variant) component of a record that corresponds to the value of the associated tag field. If the tag field contains an undefined value, the result is an empty record.

variant-variable-assignment : *Variant-variable* × *Value* → *Exception-handler* $\xrightarrow{o}$ ()

variant-variable-assignment (*vvar*, *val*)*handler* $\triangleq$
  let *mk-Variant-variable* (*var*, -, -) = *vvar* in
  assign(*var*, *val*) *handler* 150

annotations Assign a value to a field of a variant for the case where there is an explicit tag field.

tagged-variable-assignment : *Tagged-variable* × *Value* → *Exception-handler* $\xrightarrow{o}$ ()

tagged-variable-assignment (*tvar*, *val*)*handler* $\triangleq$
  let *mk-Tagged-variable* (*var*, -, -) = *tvar* in
  assign(*var*, *val*) *handler* 150

annotations Assign a value to a field of a variant for the case where there is an implicit tag field.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.6.4 Procedure Calls

A procedure call specifies the evaluation of any actual parameters and the execution of the actions associated with the procedure designator of the procedure call.

Concrete Syntax

procedure call = procedure designator, [actual parameters] ;

procedure designator = value designator ;

Abstract Syntax

types

Procedure-call :: *desig* : *Procedure-designator*
aps : *Actual-parameters*;

Procedure-designator = *Value-designator*
| *Standard-procedure-designator*
| *System-procedure-designator*

annotations The identifiers of predefined procedures are distinguished since these do not denote values (that can be assigned or compared in expressions). If the actual parameter list is not present in the concrete representation, then the translation process (from concrete to abstract syntax) will supply an empty sequence for the *aps* component of *Procedure-call*.

Static Semantics

The procedure designator of a procedure call shall be a value designator that denotes a proper procedure or the identifier of a predefined proper procedure.

NOTES

1 A value designator that denotes a proper procedure is either a procedure identifier of a proper procedure heading (possibly qualified by the name or names of modules from which the procedure identifier is exported), a value designator that denotes a constant that has a proper procedure value, or a value designator that denotes a variable that has a proper procedure type.

2 The identifiers of predefined procedures are not value designators since predefined procedures are not values that can be assigned, compared in expressions, or passed as procedure parameters.

In the case of a call of a predefined proper procedure, the actual parameters shall correspond in number and in type to those required for that predefined proper procedure; otherwise, each actual parameter shall be parameter compatible with the corresponding formal parameter of the designated proper procedure.

wf-procedure-call : *Procedure-call* → *Environment* → $\mathbb{B}$

wf-procedure-call (*mk-Procedure-call* (*desig*, *aps*))ρ $\triangleq$
wf-actual-parameters (*aps*)ρ ∧ 258
(*is-Value-designator*(*desig*) → *wf-value-designator* (*desig*)ρ ∧ 223
let *type* = *t-value-designator* (*desig*)ρ in 223
let *struc* = *structure-of* (*type*)ρ in 315
is-Proper-procedure-structure(*struc*) ∧
let *mk-Proper-procedure-structure* (*parms*) = *struc* in
is-parameters-match (*parms*, *aps*)ρ, 261
is-Standard-procedure-designator(*desig*) → *wf-standard-procedure-designator* (*desig*)(*aps*)ρ, 281
is-System-procedure-designator(*desig*) → *wf-system-procedure-designator* (*desig*)(*aps*)ρ) 347

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Dynamic Semantics

The execution of a procedure call shall be the evaluation of any actual parameters to give corresponding arguments, the binding of the results of this evaluation to the corresponding formal parameters, and the execution of the actions associated with the procedure designator. In the case of a call of a predefined proper procedure, the actions are as specified for that predefined procedure; otherwise, the actions are as specified by the block of the proper procedure declaration.

The order of evaluation, the order of accessing, and the order of binding of the actual parameters shall be implementation-dependent. The order of evaluation of the procedure designator and the actual parameters shall be implementation-dependent.

The actual parameters shall be evaluated before the execution of the actions associated with the proper procedure designator.

m-procedure-call : *Procedure-call* → *Environment* $\xrightarrow{o}$ ()

m-procedure-call (*mk-Procedure-call* (*desig*, *aps*))ρ $\triangleq$
  def *argvals* = *m-actual-parameters*(*aps*) ρ in 259
  (*is-Value-designator*(*desig*) → def *procid* = *m-value-designator*(*desig*) ρ in 223
    let *handler* = *current-exception-handler* (ρ) in 318
    call-proper-procedure-value(*procid*) (*argvals*) *handler*, 339
  is-Standard-procedure-designator(*desig*) → *m-standard-procedure-designator*(*desig*) (*argvals*) ρ, 281
  is-System-procedure-designator(*desig*) → *m-system-procedure-designator*(*desig*) (*argvals*) ρ) 347

annotations The above evaluation order for the procedure designator and the actual parameters is one of those permitted; the actual parameters may instead be evaluated before the procedure designator, or both may be evaluated in parallel — see 4.1.

6.6.5 Return Statements

A return statement specifies the termination of the execution of the block of a proper procedure, a function procedure, or a module.

Concrete Syntax

return statement = simple return statement | function return statement ;

Abstract Syntax

types

Return-statement = *Simple-return-statement* | *Function-return-statement*

Static Semantics

wf-return-statement : *Return-statement* → *Environment* → $\mathbb{B}$

wf-return-statement (*st*)ρ $\triangleq$
  (*is-Simple-return-statement*(*st*) → *wf-simple-return-statement* (*st*)ρ, 154
  is-Function-return-statement(*st*) → *wf-function-return-statement* (*st*)ρ) 155

annotations The checks that simple return statements are only contained in modules and proper procedures, and that function return statements must be contained in function procedures are performed by *wf-module-block* (see 6.5.3), *wf-proper-procedure-block* (see 6.5.1), and *wf-function-procedure-block* (see 6.5.2).

Dynamic Semantics

m-return-statement : *Return-statement* → *Environment* $\xrightarrow{o}$ ()

m-return-statement (*st*)ρ $\triangleq$
  (*is-Simple-return-statement*(*st*) → *m-simple-return-statement*(*st*) ρ, 154

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

is-Function-return-statement(*st*) → *m-function-return-statement*(*st*) ρ) 155

6.6.5.1 Simple Return Statements

A simple return statement specifies the termination of the execution of a proper procedure block or a module block.

Concrete Syntax

simple return statement = "RETURN" ;

Abstract Syntax

types

Simple-return-statement :: RETURN

Static Semantics

wf-simple-return-statement : *Simple-return-statement* → *Environment* → 𝔹
wf-simple-return-statement (*mk-Simple-return-statement* (RETURN))ρ ≜
true

Dynamic Semantics

The execution of a simple return statement shall terminate the execution of a proper procedure block or a module block.

NOTE — There is an implicit simple return statement at the end of the proper procedure block of a proper procedure. An explicit simple return statement within the proper procedure block means that there is an additional point at which the execution of the block is to be terminated.

m-simple-return-statement : *Simple-return-statement* → *Environment* $\xrightarrow{o}$ ()
m-simple-return-statement (*mk-Simple-return-statement* (RETURN))ρ ≜
c-exit(RETURN) ρ 345

6.6.5.2 Function Return Statements

A function return statement specifies the termination of the execution of a function procedure block, and defines the value to be returned by that function procedure.

Concrete Syntax

function return statement = "RETURN", expression ;

Abstract Syntax

types

Function-return-statement :: *type* : *Typed*
expr : *Expression*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Static Semantics

The result type (see 6.2.11.2) of the function procedure in which the function return statement appears shall be assignment-compatible with the type of the expression.

wf-function-return-statement : *Function-return-statement* → *Environment* → 𝔹

wf-function-return-statement (*mk-Function-return-statement* (*type*, *expr*))ρ ≙
  is-assignment-compatible (*type*, *t-expression* (*expr*)ρ)ρ ∧ 123 190
  wf-expression (*expr*)ρ 190

Dynamic Semantics

The execution of a function return statement shall terminate the execution of a function procedure block and return a value. If the type of the function result is a subrange type, the value returned shall belong to the set of values defined by the subrange type; otherwise an exception shall be raised.

NOTES

1 There may be several function return statements in a function procedure block, although only one of these will be executed on any activation of the function procedure.

2 An exception is raised if the termination of a function procedure block is not by the execution of a function return statement —see 6.5.2.

m-function-return-statement : *Function-return-statement* → *Environment* $\xrightarrow{o}$ *Value*

m-function-return-statement (*mk-Function-return-statement* (*type*, *expr*))ρ ≙
  def *res* = *m-expression*(*expr*) ρ in 190
  let *result* = if *result* : *String-value* then *to-char* (*res*) in 253
  if *is-subrange-type* (*type*)ρ ∨ *is-whole-number-type* (*type*)ρ 323 323
  then let *values* = *values-of* (*type*)ρ in 329
    if *result* ∈ *values*
    then *c-exit*(RETURN) (ρ) *result* 345
    else let *handler* = *current-exception-handler* (ρ) in 318
      mandatory-detection(RETURN-RANGE) *handler* 396
  else *c-exit*(RETURN) (ρ) *result* 345

annotations The expression of the return statement is evaluated. If the result is one of the values of the type defined by the return type of the function procedure, the result is returned, otherwise an exception is raised. A character literal is converted to a character value before the range is checked.

6.6.6 Retry Statements

A retry statement may occur only within the exceptional part of a module body or procedure body. It specifies that the normal part of the block body is to be re-executed — see 6.5.8.

Concrete Syntax

retry statement = "RETRY" ;

Abstract Syntax

types

Retry-statement :: RETRY

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Static Semantics

A retry statement shall be contained within an exceptional part of a block body.

wf-retry-statement : *Retry-statement* → *Environment* → $\mathbb{B}$
wf-retry-statement (*mk-Retry-statement* (RETRY))ρ $\triangleq$
true

annotations The check that the retry statement is contained within an exceptional part of a block body is included in the consistency check for the procedure/function/module that contains the retry statement — see 6.5.1, 6.5.2, and 6.5.3.

Dynamic Semantics

The execution of a retry statement shall revert to the state of normal execution and cause the handled block body to be retried — see 6.5.8.

m-retry-statement : *Retry-statement* → *Environment* $\xrightarrow{o}$ ()
m-retry-statement (*mk-Retry-statement* (RETRY))ρ $\triangleq$
c-exit(RETRY) ρ 345

6.6.7 With Statements

A with statement specifies a record designator and a statement sequence; within that statement sequence, the field identifiers of the designated record need not be qualified by the record designator.

Concrete Syntax

with statement = "WITH", record designator, "DO", statement sequence, "END" ;

record designator = variable designator | value designator ;

Abstract Syntax

types

With-statement :: *desig* : *Record-designator*
body : *Statement-sequence*;

Record-designator = *Variable-designator* | *Value-designator*

Static Semantics

The record designator shall denote a record variable or a record value. Within the statement sequence, the field identifiers of the record variable or record value may be used without selection.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wf-with-statement : *With-statement* → *Environment* → $\mathbb{B}$

wf-with-statement (*mk-With-statement* (*desig*, *body*))ρ $\triangleq$
  if *is-Variable-designator*(*desig*)
  then *wf-variable-designator* (*desig*)ρ $\wedge$ 182
    let *type* = *t-variable-designator* (*desig*)ρ in 182
    is-record-type (*type*)ρ $\wedge$ 321
    let *renv* = *field-types-of-record* (*type*)ρ in 326
    let ρ_{block} = *overwrite-var-environment* (*renv*)ρ in 316
    wf-statement-sequence (*body*)ρ_{block} 147
  elseif *is-Value-designator*(*desig*)
  then *wf-value-designator* (*desig*)ρ $\wedge$ 223
    is-constant-expression (*desig*)ρ $\wedge$ 254
    let *type* = *t-expression* (*desig*)ρ in 190
    is-record-type (*type*)ρ $\wedge$ 321
    let *renv* = *field-types-of-record* (*type*)ρ in 326
    let *vals* = *evaluate-constant-expression* (*desig*)ρ in 257
    let *venv* = $\{id \mapsto$ *mk-Constant-value* (*renv*(*id*), *vals*(*id*)) | *id* $\in$ dom *renv*$\}$ in
    let ρ_{block} = *overwrite-const-environment* (*venv*)ρ in 314
    wf-statement-sequence (*body*)ρ_{block} 147
  else false

annotations Check that the designator designates either a record variable or record value. Add the field identifiers with their corresponding types to the current environment to give the environment for the body of the with statement.

Dynamic Semantics

The record variable designator or record value designator shall be evaluated before the statement sequence of the with statement is executed, and that evaluation shall establish a reference to the variable or value throughout the execution of the statement sequence of the with statement.

m-with-statement : *With-statement* → *Environment* $\xrightarrow{o}$ ()

m-with-statement (*mk-With-statement* (*desig*, *body*))ρ $\triangleq$
  if *is-Variable-designator*(*desig*)
  then def *mk-Record-variable* (*fields*) = *m-variable-designator*(*desig*) ρ in 182
    let ρ_{block} = *overwrite-var-environment* (*fields*)ρ in 316
    m-statement-sequence(*body*) ρ_{block} 148
  else let *type* = *t-value-designator* (*desig*)ρ in 223
    let *renv* = *field-types-of-record* (*type*)ρ in 326
    def *vals* = *m-value-designator*(*desig*) $\overset{s}{\rho}$ in 223
    let *fields* = $\{id \mapsto$ *mk-Constant-value* (*renv* (*id*), *value* (*id*)) | *id* $\in$ dom *renv*$\}$ in
    let ρ_{block} = *overwrite-const-environment* (*fields*)ρ in 314
    m-statement-sequence(*body*) ρ_{block} 148

annotations Add the field identifiers with their corresponding variables or values to the current environment to give the environment for the body of the with statement.

6.6.8 If Statements

An if statement is used to select one statement sequence for execution from an ordered list of alternatives guarded by Boolean expressions.

Concrete Syntax

if statement = guarded statements, [if else part], "END" ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

guarded statements =
"IF", boolean expression, "THEN", statement sequence,
{ "ELSIF", boolean expression, "THEN", statement sequence } ;

if else part = "ELSE", statement sequence ;

boolean expression = expression ;

Abstract Syntax

types

If-statement :: *thens* : *Guarded-statement**
elsep : [*Statement-sequence*];

Guarded-statement :: *guard* : *Expression*
body : *Statement-sequence*

annotations An if statement always contains at least one statement sequence (which follows the first occurrence of a THEN symbol); thus after the translation process (from concrete to abstract syntax) there is always a non-empty sequence of guarded statements. A missing if else part in the concrete syntax is denoted by nil in the abstract syntax.

Static Semantics

All if statement boolean expressions shall be of the Boolean type.

wf-if-statement : *If-statement* → *Environment* → $\mathbb{B}$

wf-if-statement (*mk-If-statement* (*thens*, *elsep*))ρ $\triangleq$
$\forall$ *mk-Guarded-statement* (*guard*, *body*) $\in$ elems *thens* ·
wf-expression (*guard*)ρ $\wedge$ 190
t-expression (*guard*)ρ = BOOLEAN-TYPE $\wedge$ 190
wf-statement-sequence (*body*)ρ $\wedge$ 147
(*elsep* = nil $\vee$ *wf-statement-sequence* (*elsep*)ρ) 147

Dynamic Semantics

The boolean expressions of the guarded statements shall be evaluated in the order they occur, as far as the first boolean expression (if any) that has the value *true*. If there is such a boolean expression, its associated statement sequence shall then be executed; however, if all the boolean expressions have the value *false*, the statement sequence of the else part (if present) shall be executed. An if statement shall denote no action if all the boolean expressions have the value *false*, and there is no else part.

NOTE — The evaluation of the if statement boolean expressions could produce a transformation of the program state, i.e. have side-effects.

m-if-statement : *If-statement* → *Environment* $\xrightarrow{o}$ ()

m-if-statement (*mk-If-statement* (*thens*, *elsep*))ρ $\triangleq$
if *thens* = [ ]
then *execute-else-statements*(*elsep*) ρ 159
else let *mk-Guarded-statement* (*guard*, *body*) = hd *thens* in
def *value* = *m-expression*(*guard*) ρ in 190
if *value*
then *m-statement-sequence*(*body*) ρ 148
else *m-if-statement*(*mk-If-statement* (tl *thens*, *elsep*)) ρ 158

annotations If there are no more elements remaining in the sequence of guarded statements then the if else part of the if statement (if present) is executed. If there are more elements remaining in the sequence of guarded statements, the Boolean expression of the first guarded statement in the sequence is evaluated, and if the result is true, the corresponding statement sequence

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

is executed. If the result of the evaluation of the Boolean expression is false, then this element is removed from the sequence of guarded statements and the remainder of the sequence executed.

Auxiliaries

execute-else-statements : [*Statement-sequence*] → *Environment* $\xrightarrow{o}$ ()

execute-else-statements (*elsep*)ρ $\triangleq$
  if *elsep* = nil
  then skip
  else *m-statement-sequence*(*elsep*) ρ 148

annotations If an if else part is present, and no other component of the if statement has been executed, then the if else part is executed.

6.6.9 Case Statements

A case statement is used to select one statement sequence for execution from alternatives; the selection of the statement sequence depends on the value of an expression.

Concrete Syntax

case statement = "CASE", case selector, "OF", case list, "END" ;

case selector = ordinal expression ;

case list = case alternative, { case separator, case alternative }, [case else part] ;

case else part = "ELSE", statement sequence ;

Abstract Syntax

types

Case-statement :: *expr* : *Expression*
  cases : *Case*-set
  elsep : [*Statement-sequence*]

annotations The number of elements in the cases component of the abstract representation of a case statement is equal to the number of case alternatives of the concrete representation.

Static Semantics

The case selector shall be of an ordinal type and shall be expression compatible with the types of the constant expressions of the case label list of each case alternative.

The values contained in the case label lists of different case alternatives shall be distinct from each other.

NOTE 1 — In addition to the requirement that the values contained in the case label lists s of different case alternatives shall be distinct from each other, it is also required that the values contained in each case label list shall be distinct from each other.

EXAMPLE — Duplication of case labels in case statements.

The alternatives illustrated in the following source code are illegal for the reasons given in the associated comments.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
CASE i OF
  (*a*)  1          : (* duplicate of label in list d *)
| (*b*)  11         : (* label 11 in b duplicated in range 11..13 in list i *)
| (*c*)  15..17     : (* range 15..17 in c overlaps with range 16..19 in list j *)
| (*d*)  1          : (* duplicate of label in list a *)
| (*e*)  2, 2       : (* label 2 is duplicated in list e *)
| (*f*)  3..5, 3..5 : (* range 3..5 is duplicated in list f *)
| (*g*)  6, 6..7    : (* label 6 duplicated in range 6..7 in list g *)
| (*h*)  8..9, 8..10 : (* range 8..9 overlaps range 8..10 in list h *)
| (*i*)  11..13     : (* label 11 in b duplicated in range 11..13 in list i *)
| (*j*)  16..19     : (* range 15..17 in c overlaps with range 16..19 in list j *)
END
```

wf-case-statement : *Case-statement* → *Environment* → $\mathbb{B}$

wf-case-statement (*mk-Case-statement* (*expr*, *cases*, *elsep*))ρ $\triangleq$

wf-expression (*expr*)ρ $\wedge$ 190

$\forall$ *case* $\in$ *cases* · *wf-case* (*case*)ρ $\wedge$ 161

(*elsep* = nil $\vee$ *wf-statement-sequence* (*elsep*)ρ) $\wedge$ 147

is-disjoint-case-labels (*cases*)ρ $\wedge$ 161

let *type* = *t-expression* (*expr*)ρ in *is-ordinal-type* (*type*)ρ $\wedge$ 190 322

is-compatible-case-labels (*type*, *variants*)ρ 161

annotations Check each of the case alternatives and the case else part if it is present. Check that the case selector is of an ordinal type and that it is expression-compatible with the type associated with each case label list.

Dynamic Semantics

The case selector shall be evaluated. The resulting value shall specify the execution of the statement sequence whose case label list contains that value. If the value of the case selector is not contained in any case label list, the statement sequence of the case else part (if present) shall be executed; otherwise an exception shall be raised.

CLARIFICATION — In *Programming in Modula-2*, it is not stated what should happen when the value of the case selector is not contained in any case label list and there is no case else part.

NOTE 2 — The evaluation of the case selector could produce a transformation of the program state, i.e. have side-effects.

m-case-statement : *Case-statement* → *Environment* $\xrightarrow{o}$ ()

m-case-statement (*mk-Case-statement* (*expr*, *cases*, *elsep*))ρ $\triangleq$

def i = *m-expression*(*expr*) ρ in 190

def $s = \bigcup \{$*m-set-definition*(c.*labels*) ρ | $c \in$ *cases*$\}$ in 244

if $i \in s$

then let $c \in$ *cases* be st

def *ls* = *m-set-definition*(c.*labels*) ρ in $i \in$ *ls* in 244

m-statement-sequence(c.*body*) ρ 148

elseif *elsep* $\neq$ nil

then *m-statement-sequence*(*elsep*) ρ 148

else let *handler* = *current-exception-handler* (ρ) in 318

mandatory-detection(CASE-RANGE) *handler* 396

annotations The expression is evaluated, and the result is used to determine which statement sequence of the case statement is to be executed.

6.6.9.1 Case Alternatives

A case alternative specifies the execution of a statement sequence if its case label list contains the value of the case selector.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Concrete Syntax

case alternative = [case label list, colon, statement sequence] ;

case label list = case label, { comma, case label } ;

case label = constant expression, [ellipsis, constant expression] ;

Abstract Syntax

types

Case :: *labels* : *Set-definition*
body : *Statement-sequence*

annotations The translation process (from concrete to abstract syntax) will ensure that the number of members in the *labels* component of a *Case* component of the abstract representation is equal to the number of case labels in the case label list of the concrete representation.

Static Semantics

Each case label shall be a constant expression, or a range indicated by a pair of constant expressions. The values contained in a case label list shall be distinct from each other. A value shall be contained in a case label list if it is equal to the value of one of the constant expressions of the case label list, or if the value lies within the range specified by a pair of constant expressions of the case label list.

$wf\text{-}case : Case \to Environment \to \mathbb{B}$

$wf\text{-}case\,(mk\text{-}Case\,(labels, body))\rho \triangleq$
$\quad wf\text{-}set\text{-}definition\,(labels)\rho \wedge$ 243
$\quad is\text{-}disj\,(labels)\rho \wedge$ 614
$\quad is\text{-}constant\text{-}set\text{-}definition\,(labels)\rho \wedge$ 256
$\quad wf\text{-}statement\text{-}sequence\,(body)\rho \wedge$ 147

Auxiliaries

$is\text{-}compatible\text{-}case\text{-}labels : Typed \times *\text{-set}Case \to Environment \to \mathbb{B}$

$is\text{-}compatible\text{-}case\text{-}labels\,(type, cases)\rho \triangleq$
$\quad \forall\, mk\text{-}Case\,(labels, \text{-}) \in cases \cdot is\text{-}expression\text{-}compatible\,(type, t\text{-}set\text{-}definition\,(labels)\rho)\rho$ 121 243

annotations Check that all of the variant labels are expression-compatible with the tag type.

$is\text{-}disjoint\text{-}case\text{-}labels : Case\text{-set} \to Environment \to \mathbb{B}$

$is\text{-}disjoint\text{-}case\text{-}labels\,(caseset)\rho \triangleq$
$\quad \forall\, case_1, case_2 \in caseset \cdot$
$\qquad case_1 = case_2 \vee$
$\qquad \text{let } mk\text{-}Case\,(l_1, \text{-}) = case_1 \text{ in}$
$\qquad \text{let } mk\text{-}Case\,(l_2, \text{-}) = case_2 \text{ in}$
$\qquad evaluate\text{-}constant\text{-}set\text{-}definition\,(l_1)\rho \cap evaluate\text{-}constant\text{-}set\text{-}definition\,(l_2)\rho = \{\}$ 257 257

annotations Check that the sets of values defined by each of the case label lists are disjoint.

6.6.10 While Statements

A while statement specifies the execution of a statement sequence zero or more times, depending on the value of a Boolean expression.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Concrete Syntax

while statement = "WHILE", boolean expression, "DO", statement sequence, "END" ;

Abstract Syntax

types

While-statement :: *expr* : *Expression*
body : *Statement-sequence*

Static Semantics

A while statement boolean expression shall be of the Boolean type.

wf-while-statement : *While-statement* → *Environment* → $\mathbb{B}$

wf-while-statement (*mk-While-statement* (*expr*, *body*))ρ $\triangleq$
  wf-expression (*expr*)ρ $\wedge$ 190
  t-expression (*expr*)ρ = BOOLEAN-TYPE $\wedge$ 190
  wf-statement-sequence (*body*)ρ 147

Dynamic Semantics

The while statement boolean expression shall be evaluated before each iteration of a while statement. If the expression has the value *true*, the statement sequence shall be executed, followed again by the execution of the while statement. If the expression has the value *false*, the statement sequence shall not be executed, and the execution of the while statement shall terminate.

NOTES

1 If the result of the first evaluation of the boolean expression is *false*, the statement sequence of the while statement is not executed.

2 The evaluation of the while statement boolean expression could produce a transformation of the program state, i.e. have side-effects.

3 The execution of a while statement may be prematurely terminated by the execution of a return statement, or by the execution of an exit statement (if the while statement is contained within a loop statement).

m-while-statement : *While-statement* → *Environment* $\xrightarrow{o}$ ()

m-while-statement (*mk-While-statement* (*expr*, *body*))ρ $\triangleq$
  def *value* = *m-expression*(*expr*) ρ in 190
  if *value*
  then (*m-statement-sequence*(*body*) ρ ; 148
    m-while-statement(*mk-While-statement* (*expr*, *body*)) ρ) 162
  else skip

6.6.11 Repeat Statements

A repeat statement specifies the execution of a statement sequence one or more times, depending on the value of a Boolean expression.

Concrete Syntax

repeat statement = "REPEAT", statement sequence, "UNTIL", boolean expression ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Abstract Syntax

types

Repeat-statement :: *body* : *Statement-sequence*
expr : *Expression*

Static Semantics

A repeat statement boolean expression shall be of the Boolean type.

wf-repeat-statement : *Repeat-statement* → *Environment* → $\mathbb{B}$

wf-repeat-statement (*mk-Repeat-Statement* (*body*, *expr*))ρ $\triangleq$
wf-statement-sequence (*body*)ρ $\wedge$ 147
wf-expression (*expr*)ρ $\wedge$ 190
t-expression (*expr*)ρ = BOOLEAN-TYPE 190

Dynamic Semantics

In the execution of a repeat statement, the statement sequence shall first be executed. The repeat statement boolean expression shall be evaluated; if the expression has the value *true*, the repeat statement shall be executed again. If the expression has the value *false*, the execution of the repeat statement shall terminate.

NOTES

1 The statement sequence of a repeat statement is executed at least once.

2 The evaluation of the repeat statement boolean expression could produce a transformation of the program state, i.e. have side-effects.

3 The execution of a repeat statement may be prematurely terminated by the execution of a return statement, or by the execution of an exit statement (if the repeat statement is contained within a loop statement).

m-repeat-statement : *Repeat-statement* → *Environment* $\xrightarrow{o}$ ()

m-repeat-statement (*mk-Repeat-statement* (*body*, *expr*))ρ $\triangleq$
(*m-statement-sequence*(*body*) ρ ; 148
(def *value* = *m-expression*(*expr*) ρ in 190
if $\neg$ *value*
then *m-repeat-statement*(*mk-Repeat-statement* (*body*, *expr*)) ρ 163
else skip))

6.6.12 Loop Statements

A loop statement specifies the repeated execution of a statement sequence. The statement sequence may contain zero or more exit statements, the execution of any one of which terminates the execution of the loop statement.

Concrete Syntax

loop statement = "LOOP", statement sequence, "END" ;

Abstract Syntax

types

Loop-statement :: *body* : *Statement-sequence*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Static Semantics

wf-loop-statement : *Loop-statement* → *Environment* → $\mathbb{B}$

wf-loop-statement (*mk-Loop-statement* (*body*))ρ $\triangleq$
  wf-statement-sequence (*body*)ρ 147

Dynamic Semantics

The statement sequence of a loop statement shall be repeatedly executed until an exit statement of the loop statement is executed.

NOTE — The execution of a loop statement may be prematurely terminated by the execution of a return statement.

m-loop-statement : *Loop-statement* → *Environment* $\xrightarrow{o}$ ()

m-loop-statement (*mk-Loop-statement* (*body*))ρ $\triangleq$
  def ρ_{cont} = *c-tixe*({LOOPEXIT ↦ skip}) ρ in 345
  (*m-statement-sequence*(*body*) ρ_{cont} ; 148
   m-loop-statement(*mk-Loop-statement* (*body*)) ρ) 164

6.6.13 · Exit Statements

An exit statement may occur only within a loop statement, where it specifies the termination of that loop statement.

Concrete Syntax

exit statement = "EXIT" ;

Abstract Syntax

types

  Exit-statement :: EXIT

Static Semantics

wf-exit-statement : *Exit-statement* → *Environment* → $\mathbb{B}$

wf-exit-statement (*mk-Exit-statement* (EXIT))ρ $\triangleq$
  true

annotations The check that the exit statement is contained inside a loop statement is part of the consistency check for the procedure/function/module that contains the exit statement.

Dynamic Semantics

The execution of an exit statement shall terminate the execution of a loop statement; the loop statement that is terminated is the loop statement that most immediately contains the exit statement.

m-exit-statement : *Exit-statement* → *Environment* $\xrightarrow{o}$ ()

m-exit-statement (*mk-Exit-statement* (EXIT))ρ $\triangleq$
  c-exit(LOOPEXIT) ρ 345

6.6.14 For Statements

A for statement specifies that a statement sequence is to be repeatedly executed as elements of a sequence of values are assigned, one by one, to a control variable.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Concrete Syntax

for statement =
"FOR", control variable identifier, assignment operator, initial value, "TO", final value, ["BY", step size], "DO", statement sequence, "END" ;

control variable identifier = identifier ;

initial value = ordinal expression ;

final value = ordinal expression ;

step size = constant expression ;

Abstract Syntax

types

For-statement :: *id* : *Identifier*
initial : *Expression*
final : *Expression*
step : *Expression*
body : *Statement-sequence*

annotations If the step size is not present in the concrete representation, the translation process (from concrete to abstract syntax) will supply a *step* component in the abstract representation equal to 1.

Static Semantics

The control variable identifier shall be the identifier of the control variable. The initial value and the final value shall be expressions. The type of the control variable shall be an ordinal type that is assignment-compatible with the type of the initial value. Either the type of the control variable or the host type of the type of the control variable and the type of the final value shall be expression-compatible. If a for statement does not contain a step size, the step size shall have the value 1. Otherwise the step size shall be a constant expression of a whole number type, shall be in an implementation-defined range, and shall not have the value 0.

NOTES

1 The requirement that the type of the control variable is assignment-compatible with the type of the initial value is made since this value might be assigned to the control variable.

2 Other constraints on the use of a control variable are given in 6.6.15.

CHANGE — In *Programming in Modula-2*, it is required that the type of the control variable is expression-compatible with the type of the initial value.

values

most-negative-step-value : $\mathbb{Z}$ = implementation-defined;

most-positive-step-value : $\mathbb{Z}$ = implementation-defined

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wf-for-statement : *For-statement* → *Environment* → $\mathbb{B}$

wf-for-statement (*mk-For-statement* (*id*, *initial*, *final*, *step*, *body*))ρ $\triangleq$
  is-simple-variable-identifier (*Id*)ρ $\wedge$ 315
  let *type* = *t-entire-designator* (*mk-Entire-designator* (*id*))ρ in 183
  is-ordinal-type (*type*)ρ $\wedge$ 322
  wf-expression (*initial*)ρ $\wedge$ 190
  wf-expression (*final*)ρ $\wedge$ 190
  wf-expression (*step*)ρ $\wedge$ 190
  is-assignment-compatible (*type*, *t-expression* (*initial*)ρ)ρ $\wedge$ 123 190
  is-expression-compatible (*host-type-of* (*type*)ρ, *t-expression* (*final*)ρ)ρ $\wedge$ 121 325 190
  is-constant-expression (*step*)ρ $\wedge$ 254
  let *stype* = *t-expression* (*step*)ρ in 190
  is-whole-number-type (*stype*)ρ $\wedge$ 323
  let *by* = *evaluate-constant-expression* (*step*)ρ in 257
  wf-step-range (*by*) $\wedge$ 166
  wf-statement-sequence (*body*)ρ $\wedge$ 147
  $\neg$ *is-threatened-in-statement-seq* (*id*, *body*)ρ 171

annotations Check each of the components of the for statement. The check that the control variable is declared in a variable declaration part of the block that contains the for statement is part of the consistency check for the procedure, function, or module that contains the for statement.

wf-step-range : *Value* → $\mathbb{B}$

wf-step-range (*by*) $\triangleq$
  by $\neq 0$ $\wedge$
  by $\geq$ *most-negative-step-value* $\wedge$
  by $\leq$ *most-positive-step-value*

annotations Check that the step value is between implementation-defined limits.

Dynamic Semantics

The expressions that specify the initial value and the final value shall be evaluated in an implementation-dependent order.

If the step size is positive and the initial value is greater than the final value, the statement sequence shall not be executed, and the execution of the for statement shall terminate. If the step size is negative and the initial value is less than the final value, the statement sequence shall not be executed, and the execution of the for statement shall terminate. Otherwise a sequence of values shall be established and for each element of the sequence in turn, the element shall be assigned to the control variable, and then the statement sequence shall be executed; the execution of the for statement shall terminate when the statement sequence has been executed for each value in the sequence.

The sequence of values for the control variable shall be the longest sequence that satisfies the conditions that the first element of the sequence shall be equal to the initial value, the last element of the sequence shall be less than or equal to the final value if the step size is positive, or greater than or equal to the final value if the step size is negative, and the difference between adjacent values in the sequence shall be equal to the step size.

After termination of the for statement, the value of the control variable shall be undefined.

NOTES

3 The expressions that are the initial value and the final value are evaluated only once at the start of the execution of a for statement. These evaluations could produce a transformation of the program state, i.e. have side-effects.

4 The statement sequence of a for statement is not executed at all if the initial value is greater than the final value (in the case that the step size is positive) or if the initial value is less than the final value (in the case that the step size is negative).

5 The execution of a for statement may be prematurely terminated by the execution of a return statement, or by the execution of an exit statement (if the for statement is contained within a loop statement).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

m-for-statement : *For-statement* → *Environment* $\xrightarrow{o}$ ()

m-for-statement (*mk-For-statement* (*id*, *initial*, *final*, *step*, *body*))ρ $\triangleq$
  def ρ_{cont} = *c-tixe*({LOOPEXIT ↦ 345
    (*assign*(*mk-Entire-designator* (*id*), UNDEFINED) ; *c-exit*(LOOPEXIT) ρ})) ρ in 150 345
  def *cvar* = *m-entire-designator*(*mk-Entire-designator* (*id*)) ρ in 183
  def *from* = *m-expression*(*initial*) ρ in 190
  def *to* = *m-expression*(*final*) ρ in 190
  def *by* = *m-expression*(*step*) ρ in 190
  let *control* = *iteration-values* (*from*, *to*, *by*) in 167
  execute-for-statement(*cvar*, *control*, *body*) ρ_{cont} 167

annotations The above evaluation order for the initial value and the final value is one of those permitted; the final value may instead be evaluated before the initial value, or both may be evaluated in parallel — see 4.1.

The iteration sequence for the loop is built from the initial value, step, and final value and then the loop is executed using the iteration sequence.

Auxiliaries

execute-for-statement : *Variable* × *Ordinal-value** × *Statement-sequence* → *Environment* $\xrightarrow{o}$ ()

execute-for-statement (*cvar*, *control*, *body*)ρ $\triangleq$
  let *handler* = *current-exception-handler* (ρ) in 318
  if *control* = [ ]
  then *assign*(*cvar*, UNDEFINED) *handler* 150
  else (*assign*(*cvar*, hd *control*) handler ; 150
    m-statement-sequence(*body*) ρ ; 148
    execute-for-statement(*cvar*, tl *control*, *body*) ρ) 167

annotations If the iteration sequence is empty, terminate the execution of the loop and assign the value undefined to the control variable. If the iteration sequence is not empty, assign the next iteration value to the control variable, execute the body of the loop and execute the for statement again with an iteration sequence having the value used for the current iteration removed.

iteration-values : *Ordinal-value* × *Ordinal-value* × *Ordinal-value* → *Ordinal-value**

iteration-values (*from*, *to*, *by*) $\triangleq$
  let *vals* = if *by* > 0
    then *iteration-up-sequence* (*from*, *to*) 167
    else *iteration-down-sequence* (*from*, *to*) in 167
  [*vals*(*i*) | *i* ∈ inds *vals* · (*i* − 1) mod abs $b\tilde{y}$ = 0]

annotations This function constructs the sequence of values that may be assigned to the control variable. The sequence of values starts with the initial value. The last value in the sequence is less than or equal to the final value if the step is positive, or greater than or equal to the final value if the step is negative. The difference between adjacent values in the sequence is equal to the step.

If the initial value is greater than the final value (in the case that the step is positive) or if the initial value is less than the final value (in the case that the step is negative) the sequence of values is empty.

iteration-up-sequence : *Ordinal-value* × *Ordinal-value* → *Ordinal-value**

iteration-up-sequence (*initial*, *final*) $\triangleq$
  if *initial* > *final*
  then [ ]
  elseif *initial* = *final*
  then [*initial*]
  else [*initial*] $\frown$ *iteration-up-sequence* (*succ* (*initial*), *final*) 167 168

annotations Construct the sequence of consecutive values from the initial and final values.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

iteration-down-sequence : *Ordinal-value* × *Ordinal-value* → *Ordinal-value**

iteration-down-sequence (*initial*, *final*) ≙
  if *initial* < *final*
  then [ ]
  elseif *initial* = *final*
  then [*initial*]
  else [*initial*] ↷ *iteration-down-sequence* (*pred* (*initial*), *final*) 167 168

annotations Construct a sequence of iteration values from the first and last values.

succ : *Ordinal-value* → *Ordinal-value*

succ (*val*) ≙
  if *is-Number*(*val*)
  then *val* + 1
  elseif *is-*𝔹(*val*)
  then true
  else let *mk-Enumeration-value* (*value*, *order*) = *val* in
    let *i* = *firstocc* (*value*, *order*) in 615
    mk-Enumeration-value (*order* (*i* + 1), *order*)

pred : *Ordinal-value* → *Ordinal-value*

pred (*val*) ≙
  if *is-Number*(*val*)
  then *val* − 1
  elseif *is-*𝔹(*val*)
  then false
  else let *mk-Enumeration-value* (*value*, *order*) = *val* in
    let *i* = *firstocc* (*value*, *order*) in 615
    mk-Enumeration-value (*order* (*i* − 1), *order*)

6.6.15 Well-formed Control Variables

A variable that is used as a control variable of a for statement must be a simple variable. It must be a local variable, and it must not be exported nor imported. It must not be threatened in any statement within the statement sequence of the for statement, or by any procedure in the declarations part of the block containing the for statement.

If a control variable is not threatened, no statement other than the for statement itself can attempt to alter the for loop control variable.

Static Semantics

The control variable of each for statement in the body of a block shall be declared as a simple variable in a variable declaration part of the block that immediately contains the for statement.

NOTE 1 — Since a control variable must be declared in the declaration part of a block in which the variable is used, it follows that it is not possible for the control variable to be:

— a formal parameter, or
— a variable that is imported into a program module or an implementation module, or
— a variable that is exported from a separate module.

A variable that is used as a control variable shall not be imported by a local module declaration of the block in which the variable is declared.

A variable that is used as a control variable in the block of a local module declaration shall not be exported from that module.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NOTE 2 — This restriction is checked by the well-formed condition for a module — see 6.2.12.

A control variable shall not be threatened in a statement of the statement sequence of the for statement.

NOTE 3 — This restriction is checked by the well-formed condition for a for statement — see 6.6.14.

A control variable shall not be threatened in a procedure declared in the declarations of the block that contains the for statement.

CLARIFICATION — In *Programming in Modula-2*:

- it is not stated that a control variable must be a local variable,
- it is not stated that a control variable cannot be exported,
- it is stated that the control variable should not be changed by the statement sequence of the for statement.

6.6.15.1 Threatening

A control variable shall be threatened in a proper procedure declaration if an identifier with the same spelling is not declared in the procedure heading, and the control variable is threatened in the block component of the procedure.

A control variable shall be threatened in a function procedure declaration if an identifier with the same spelling is not declared in the procedure heading, and the control variable is threatened in the block component of the procedure.

A control variable shall be threatened in a block if an identifier with the same spelling is not declared in a declaration of the block, and the control variable is threatened either in a procedure of a declaration of the block or in a statement of the statement sequence of the block.

A control variable shall be threatened in an assignment statement if it is the variable designator of the assignment, if it is threatened in an expression of the variable designator component, or if it is threatened in the expression component of the assignment statement.

A control variable shall be threatened in a procedure call if it is threatened in an expression component of the procedure designator component of the procedure call or it is threatened in an actual parameter of the procedure call.

A control variable shall be threatened in a return statement if the return statement is a function return statement and the control variable is threatened in the expression component of the return statement.

A control variable shall be threatened in a with statement if an identifier with the same spelling is not a field of the variable designated by the designator component of the with statement and the control variable is threatened in a statement of the statement sequence component of the with statement.

A control variable shall be threatened in an if statement if it is threatened in any Boolean expression components or if it is threatened in a statement of the statement sequence component.

A control variable shall be threatened in a case statement if it is threatened in the case selector or if it is threatened in a statement of the statement sequence component.

A control variable shall be threatened in a while statement if it is threatened in the Boolean expression component or if it is threatened in a statement of the statement sequence component.

A control variable shall be threatened in a repeat statement if it is threatened in a statement of the statement sequence component or if it is threatened in the Boolean expression component.

A control variable shall be threatened in a loop statement if it is threatened in a statement of the statement sequence component.

A control variable shall be threatened in a for statement if it is the control variable of the for statement, if it is

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

threatened in the initial value, if it is threatened in the final value, or if it is threatened in a statement of the statement sequence component.

NOTE 1 — This rule is concerned with the threatening of the control variable of a for statement in another for statement nested within the statement sequence of the first for statement.

A control variable shall be threatened in a variable designator if it is threatened in an expression component of the variable designator.

A control variable shall be threatened in an expression if it is threatened as an element of the expression.

A control variable shall be threatened in a function call if it is threatened in the value designator component or if it is threatened in an actual parameter of the function call.

A control variable shall be threatened in an actual parameter to a procedure or function call if it is an actual parameter corresponding to a variable formal parameter or if it is threatened in an expression component of the actual parameter.

NOTES

2 The predefined procedures **DEC** and **INC**, and the function **ADR** of the module **SYSTEM**, are considered to take a variable parameter, and therefore cannot be passed a control variable.

3 A control variable is not threatened in an empty statement or in an exit statement or in a simple return statement or in a retry statement.

control-variables : *Module-block* → *Identifier*-set

control-variables (*block*) ≙
  let *mk-Module-block* (-, *actions*) = *block* in
  let *for-sts* = *x-statements-of-statement-seq* (*actions*, *For-statement*) in 179
  {*st.id* | *st* ∈ *for-sts*}

annotations Return the set of identifiers that are used (in a block) as the identifiers of for statement control variables.

wf-control-variables : *Declarations* × *Statement** → *Environment* → 𝔹

wf-control-variables (*decls*, *sts*)ρ ≙
  let *for-sts* = *x-statements-of-statement-seq* (*sts*, *For-statement*) in 179
  let *for-ids* = {*st.id* | *st* ∈ *for-sts*} in
  ∀ *id* ∈ *for-ids* ·
    is-simple-variable (*id*, *decls*) ∧ 170
    ¬ *is-threatened-in-procedure* (*id*, *decls*)ρ ∧ 170
    ¬ *is-imported-into-module* (*id*, *decls*)ρ 176

annotations The result is true if and only if all the variables that are used as control variables are declared in the block, are not threatened in the statements or expressions contained in the block, are not threatened in procedures declared in the block, and are not imported by a module declared in the block.

is-simple-variable : *Identifier* × *Declarations* → 𝔹

is-simple-variable (*id*, *decls*) ≙
  ∃ *mk-Variable-declaration* (*ids*, -) ∈ elems *decls* · *id* ∈ elems *ids*

annotations The result is true if and only if there exists a variable declaration that contains a declaration of the variable.

annotations The identifier parameter in the following operations is a control variable identifier and it designates a control variable.

is-threatened-in-procedure : *Identifier* × *Declarations* → *Environment* → 𝔹

is-threatened-in-procedure (*id*, *decls*)ρ ≙
  ∃ *proc* ∈ elems *decls* ∩ *Proper-procedure-declaration* ·
    is-threatened-in-proper-procedure (*id*, *proc*)ρ ∨ 171
  ∃ *proc* ∈ elems *decls* ∩ *Function-procedure-declaration* ·
    is-threatened-in-function-procedure (*id*, *proc*)ρ 171

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations A control variable is threatened in the procedures declared in a block if there exists a procedure in the declarations component that threatens the control variable.

is-threatened-in-proper-procedure : *Identifier* × *Proper-procedure-declaration* → *Environment* → $\mathbb{B}$

is-threatened-in-proper-procedure $(id, procd)\rho \triangleq$
  let *mk-Proper-procedure-declaration* $(head, block) = procd$ in
  let *mk-Proper-procedure-heading* $(\text{-}, parms) = head$ in
  $id \notin$ *identifiers-of-formal-parameter-list* $(parms) \wedge$ 76
  is-threatened-in-proper-procedure-block $(id, block)\rho$ 171

annotations A control variable is threatened in a proper procedure if an identifier with the same spelling is not declared in the procedure heading, and the control variable is threatened in the block component of the procedure.

is-threatened-in-function-procedure : *Identifier* × *Function-procedure-declaration* → *Environment* → $\mathbb{B}$

is-threatened-in-function-procedure $(id, fund)\rho \triangleq$
  let *mk-Function-procedure-declaration* $(head, block) = fund$ in
  let *mk-Function-procedure-heading* $(\text{-}, parms, \text{-}) = head$ in
  $id \notin$ *identifiers-of-formal-parameter-list* $(parms) \wedge$ 76
  is-threatened-in-function-procedure-block $(id, block)\rho$ 171

annotations A control variable is threatened in a function procedure if an identifier with the same spelling is not declared in the procedure heading, and the control variable is threatened in the block component of the procedure.

is-threatened-in-proper-procedure-block : *Identifier* × *Proper-procedure-block* → *Environment* → $\mathbb{B}$

is-threatened-in-proper-procedure-block $(id, block)\rho \triangleq$
  let *mk-Proper-procedure-block* $(decls, actions) = block$ in
  $id \notin$ *identifiers-declared-in* $(decls) \wedge$ 82
  (*is-threatened-in-procedure* $(id, decls)\rho \vee$ *is-threatened-in-statement-seq* $(id, actions)\rho$) 170 171

annotations A control variable is threatened in a proper procedure block if an identifier with the same spelling is not declared in the declarations component of the block, and the control variable is threatened either in a procedure of the declarations component of the block or in the statement sequence of the block.

is-threatened-in-function-procedure-block : *Identifier* × *Function-procedure-block* → *Environment* → $\mathbb{B}$

is-threatened-in-function-procedure-block $(id, block)\rho \triangleq$
  let *mk-Function-procedure-block* $(decls, actions) = block$ in
  $id \notin$ *identifiers-declared-in* $(decls) \wedge$ 82
  (*is-threatened-in-procedure* $(id, decls)\rho \vee$ *is-threatened-in-statement-seq* $(id, actions)\rho$) 170 171

annotations A control variable is threatened in a function procedure block if an identifier with the same spelling is not declared in the declarations component of the block, and the control variable is threatened either in a procedure of the declarations component of the block or in the statement sequence of the block.

is-threatened-in-statement-seq : *Identifier* × *Statement** → *Environment* → $\mathbb{B}$

is-threatened-in-statement-seq $(id, sts)\rho \triangleq$
  $\exists\, st \in$ elems $sts \cdot$
    is-threatened-in-statement $(id, st)\rho$ 171

annotations A control variable is threatened in a statement sequence if at least one of the statements of the statement sequence threatens the control variable.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

is-threatened-in-statement : *Identifier* × *Statement* → *Environment* → 𝔹

is-threatened-in-statement (*id*, *st*)*ρ* ≜
  if *is-Empty-statement*(*st*)
  then false
  elseif *is-Assignment-statement*(*st*)
  then *is-threatened-in-assignment-statement* (*id*, *st*) 172
  elseif *is-Procedure-call*(*st*)
  then *is-threatened-in-procedure-call* (*id*, *st*) 172
  elseif *is-Return-statement*(*st*)
  then *is-threatened-in-return-statement* (*id*, *st*) 172
  elseif *is-Retry-statement*(*st*)
  then false
  elseif *is-With-statement*(*st*)
  then *is-threatened-in-with-statement* (*id*, *st*)*ρ* 173
  elseif *is-If-statement*(*st*)
  then *is-threatened-in-if-statement* (*id*, *st*)*ρ* 173
  elseif *is-Case-statement*(*st*)
  then *is-threatened-in-case-statement* (*id*, *st*)*ρ* 173
  elseif *is-While-statement*(*st*)
  then *is-threatened-in-while-statement* (*id*, *st*)*ρ* 173
  elseif *is-Repeat-statement*(*st*)
  then *is-threatened-in-repeat-statement* (*id*, *st*)*ρ* 173
  elseif *is-Loop-statement*(*st*)
  then *is-threatened-in-loop-statement* (*id*, *st*)*ρ* 174
  elseif *is-Exit-statement*(*st*)
  then false
  elseif *is-For-statement*(*st*)
  then *is-threatened-in-for-statement* (*id*, *st*)*ρ* 174
  else undefined

annotations Check if control variable threatened by a statement.

is-threatened-in-assignment-statement : *Identifier* × *Assignment-statement* → 𝔹

is-threatened-in-assignment-statement (*id*, *st*) ≜
  let *mk-Assignment-statement* (*desig*, *expr*) = *st* in
  desig = *mk-Entire-designator* (*id*) ∨
  is-threatened-in-variable-designator (*id*, *desig*) ∨ 174
  is-threatened-in-expression (*id*, *expr*) 174

annotations A control variable is threatened in an assignment statement if it is the variable designator of the assignment statement, or if it is threatened in any expression component of the assignment statement.

is-threatened-in-procedure-call : *Identifier* × *Procedure-call* → 𝔹

is-threatened-in-procedure-call (*id*, *st*) ≜
  let *mk-Procedure-call* (*desig*, *aps*) = *st* in
  is-threatened-in-value-designator (*id*, *desig*) ∨ 175
  is-threatened-as-argument (*id*, *aps*) 176

annotations A control variable is threatened in a procedure call if it is threatened as the procedure designator component of the procedure call or it is threatened in an actual parameter of the procedure call.

is-threatened-in-return-statement : *Identifier* × *Return-statement* → 𝔹

is-threatened-in-return-statement (*id*, *st*) ≜
  cases *st* :
    mk-Simple-return-statement (-) → false,
    mk-Function-return-statement (-, *expr*) → *is-threatened-in-expression* (*id*, *expr*) 174
  end

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations A control variable is threatened in a return statement if the return statement is a function return statement and the control variable is threatened in the expression component of the return statement.

is-threatened-in-with-statement : *Identifier* × *With-statement* → *Environment* → 𝔹

is-threatened-in-with-statement $(id, st)\rho \triangleq$
  let *mk-With-statement* $(desig, body) = st$ in
  if *is-Variable-designator*$(desig)$
  then let *type* = *t-variable-designator* $(desig)\rho$ in 182
    let *renv* = *field-types-of-record* $(type)\rho$ in 326
    $id \notin$ dom *renv* ∧
    is-threatened-in-variable-designator $(id, desig)$ ∧ 174
    is-threatened-in-statement-seq $(id, body)\rho$ 171
  elseif *is-Value-designator*$(desig)$
  then let *type* = *t-value-designator* $(desig)\rho$ in 223
    let *renv* = *field-types-of-record* $(type)\rho$ in 326
    $id \notin$ dom *renv* ∧
    is-threatened-in-statement-seq $(id, body)\rho$ 171
  else undefined

annotations A control variable is threatened in a with statement if an identifier with the same spelling is not a field of the variable or value designated by the designator component of the with statement and the control variable is threatened in the statement sequence that is the body of the with statement.

renv is the addition to the environment caused by the with statement. If the control variable identifier belongs to the domain of the mapping then within the with statement, it is to a field of the record that is being threatened, not the control variable.

is-threatened-in-if-statement : *Identifier* × *If-statement* → *Environment* → 𝔹

is-threatened-in-if-statement $(id, st)\rho \triangleq$
  let *mk-If-statement* $(thens, elsep) = st$ in
  $(\exists$ *mk-Guarded-statement* $(guard, body) \in$ elems *thens* ·
    is-threatened-in-expression $(id, guard)$ ∨ 174
    is-threatened-in-statement-seq $(id, body)\rho)$ ∨ 171
  $(elsep \neq$ nil ∧ *is-threatened-in-statement-seq* $(id, elsep)\rho)$ 171

annotations A control variable is threatened in an if statement if it is threatened in the Boolean expression component of a guarded statement, if it is threatened in the statement sequence component of a guarded statement, or if it is threatened in the statement sequence component of the if else part.

is-threatened-in-case-statement : *Identifier* × *Case-statement* → *Environment* → 𝔹

is-threatened-in-case-statement $(id, st)\rho \triangleq$
  let *mk-Case-statement* $(expr, cases, elsep) = st$ in
  is-threatened-in-expression $(id, expr)$ ∨ 174
  $(\exists$ *mk-Case* $(\text{-}, body) \in cases$ · *is-threatened-in-statement-seq* $(id, body)\rho)$ ∨ 171
  $(elsep \neq$ nil ∧ *is-threatened-in-statement-seq* $(id, elsep)\rho)$ 171

annotations A control variable is threatened in a case statement if it is threatened in the case selector, if it is threatened in one of the statement sequences of the case alternatives, or if it is threatened in the statement sequence of the if else part.

is-threatened-in-while-statement : *Identifier* × *While-statement* → *Environment* → 𝔹

is-threatened-in-while-statement $(id, st)\rho \triangleq$
  let *mk-While-statement* $(expr, body) = st$ in
  is-threatened-in-expression $(id, expr)$ ∨ 174
  is-threatened-in-statement-seq $(id, body)\rho$ 171

annotations A control variable is threatened in a while statement if it is threatened in the expression or if it is threatened in the statement sequence.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

is-threatened-in-repeat-statement : *Identifier* × *Repeat-statement* → *Environment* → 𝔹

is-threatened-in-repeat-statement (*id*, *st*)ρ ≜
  let *mk-Repeat-statement* (*body*, *expr*) = *st* in
  is-threatened-in-statement-seq (*id*, *body*)ρ ∨ 171
  is-threatened-in-expression (*id*, *expr*) 174

annotations A control variable is threatened in a repeat statement if it is threatened in the statement sequence or if it is threatened in the expression.

is-threatened-in-loop-statement : *Identifier* × *Loop-statement* → *Environment* → 𝔹

is-threatened-in-loop-statement (*id*, *st*)ρ ≜
  let *mk-Loop-statement* (*body*) = *st* in
  is-threatened-in-statement-seq (*id*, *body*)ρ 171

annotations A control variable is threatened in a loop statement if it is threatened in the statement sequence.

is-threatened-in-for-statement : *Identifier* × *For-statement* → *Environment* → 𝔹

is-threatened-in-for-statement (*cid*, *st*)ρ ≜
  let *mk-For-statement* (*id*, *initial*, *final*, -, *body*) = *st* in
  cid = *id* ∨
  is-threatened-in-expression (*cid*, *initial*) ∨ 174
  is-threatened-in-expression (*cid*, *final*) ∨ 174
  is-threatened-in-statement-seq (*cid*, *body*)ρ 171

annotations A control variable is threatened in a for statement if it is the control variable of the for statement, if it is threatened in the initial expression, if it is threatened in the final expression, or if it is threatened in the statement sequence that is the body of the for statement.

is-threatened-in-variable-designator : *Identifier* × *Variable-designator* → 𝔹

is-threatened-in-variable-designator (*id*, *vdesig*) ≜
  cases *vdesig* :
    mk-Entire-designator (-) → false,
    mk-Indexed-designator (*desig*, *expr*) → *is-threatened-in-variable-designator* (*id*, *desig*) ∨ 174
      is-threatened-in-expression (*id*, *expr*), 174
    mk-Selected-designator (*desig*, -) → *is-threatened-in-variable-designator* (*id*, *desig*), 174
    mk-Dereferenced-designator (*desig*) → *is-threatened-in-variable-designator* (*id*, *desig*) 174
  end

annotations A control variable is threatened in a variable designator if it is threatened in any expression component of the variable designator.

is-threatened-in-expression : *Identifier* × *Expression* → 𝔹

is-threatened-in-expression (*id*, *expr*) ≜
  if *is-Infix-expression*(*expr*)
  then *is-threatened-in-infix-expression* (*id*, *expr*) 175
  elseif *is-Prefix-expression*(*expr*)
  then *is-threatened-in-prefix-expression* (*id*, *expr*) 175
  elseif *is-Value-designator*(*expr*)
  then *is-threatened-in-value-designator* (*id*, *expr*) 175
  elseif *is-Function-call*(*expr*)
  then *is-threatened-in-function-call* (*id*, *expr*) 175
  elseif *is-Value-constructor*(*expr*)
  then *is-threatened-in-value-constructor* (*id*, *expr*) 175
  elseif *is-Constant-literal*(*expr*)
  then false
  else undefined

annotations A control variable is threatened in an expression if it is threatened as an element of the expression.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

is-threatened-in-infix-expression : *Identifier* × *Infix-expression* → 𝔹

is-threatened-in-infix-expression (*id*, *expr*) ≜
  let *mk-Infix-expression* (*left*, -, *right*) = *expr* in
  is-threatened-in-expression (*id*, *left*) ∨ 174
  is-threatened-in-expression (*id*, *right*) 174

annotations A control variable is threatened in an infix expression if it is threatened in an expression component of the infix expression.

is-threatened-in-prefix-expression : *Identifier* × *Prefix-expression* → 𝔹

is-threatened-in-prefix-expression (*id*, *pexpr*) ≜
  let *mk-Prefix-expression* (-, *expr*) = *pexpr* in
  is-threatened-in-expression (*id*, *expr*) 174

annotations A control variable is threatened in an prefix expression if it is threatened in the expression component of the prefix expression.

is-threatened-in-value-designator : *Identifier* × *Value-designator* → 𝔹

is-threatened-in-value-designator (*id*, *vdesig*) ≜
  cases *vdesig* :
    mk-Entire-value (-) → false,
    mk-Indexed-value (*desig*, *expr*) → *is-threatened-in-value-designator* (*id*, *desig*) ∨ 175
      is-threatened-in-expression (*id*, *expr*), 174
    mk-Selected-value (*desig*, -) → *is-threatened-in-value-designator* (*id*, *desig*), 175
    mk-Dereferenced-value (*desig*) → *is-threatened-in-value-designator* (*id*, *desig*) 175
  end

annotations A control variable is threatened in a value designator if it is threatened in any expression component of the value designator.

is-threatened-in-function-call : *Identifier* × *Function-call* → 𝔹

is-threatened-in-function-call (*id*, *expr*) ≜
  let *mk-Function-call* (*desig*, *args*) = *expr* in
  is-threatened-in-value-designator (*id*, *desig*) ∨ 175
  is-threatened-as-argument (*id*, *args*) 176

annotations A control variable is threatened in a function call if it is threatened as the value designator component or if it is threatened in an actual parameter of the function call.

is-threatened-in-value-constructor : *Identifier* × *Value-constructor* → 𝔹

is-threatened-in-value-constructor (*id*, *expr*) ≜
  cases *expr* :
    mk-Array-constructor (-, *sval*) → *is-threatened-in-structured-value* (*id*, *sval*), 175
    mk-Record-constructor (-, *sval*) → *is-threatened-in-structured-value* (*id*, *sval*), 175
    mk-Set-constructor (-, *def*) → *is-threatened-in-set-constructor* (*id*, *def*) 175
  end

annotations A control variable is threatened in a value constructor if it is threatened in any expression component of the value constructor.

is-threatened-in-structured-value : *Identifier* × *Structured-value* → 𝔹

is-threatened-in-structured-value (*id*, *sval*) ≜
  ∃ *mk-Element* (*val*, -) ∈ elems *sval* · *is-threatened-in-expression* (*id*, *val*) 174

annotations A control variable is threatened in a structured value if it is threatened in any expression component of the structured value. Note that a control variable cannot be threatened in the replication component as it is a constant expression.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

is-threatened-in-set-constructor : *Identifier* × *Set-definition* → 𝔹

is-threatened-in-set-constructor (*id*, *set*) ≜
  ∃ *mem* ∈ *set* ·
    cases *mem* :
      mk-Singleton (*expr*) → *is-threatened-in-expression* (*id*, *expr*), 174
      mk-Interval (*min*, *max*) → *is-threatened-in-expression* (*id*, *min*) ∨ 174
                                    is-threatened-in-expression (*id*, *max*) 174
    end

annotations A control variable is threatened in a set constructor if it is threatened in any expression component of the set constructor.

is-threatened-as-argument : *Identifier* × *Actual-parameters* → 𝔹

is-threatened-as-argument (*id*, *aps*) ≜
  ∃ *ap* ∈ elems *aps* ·
    if *is-Variable-designator*(*ap*)
    then *mk-Entire-designator* (*id*) = *ap* ∨
      is-threatened-in-variable-designator (*id*, *ap*) 174
    elseif *is-Expression*(*ap*)
    then *is-threatened-in-expression* (*id*, *ap*) 174
    elseif *is-Type-parameter*(*ap*)
    then false
  else false

annotations A control variable is threatened as an actual parameter to a procedure or function call if it is an actual parameter corresponding to a variable formal parameter or if it is threatened in an expression component of the actual parameter.

is-imported-into-module : *Identifier* × *Declarations* → *Environment* → 𝔹

is-imported-into-module (*id*, *decls*)ρ ≜
  ∃ *mk-Module-declaration* (-, *imports*, -, -, -) ∈ elems *decls* ·
    id ∈ *imports-of* (*imports*)ρ 42

annotations Check if a control variable is imported by a module declaration of the block.

6.6.16 Auxiliary Functions

6.6.16.1 Well-formed Return Statements

wf-simple-return-statements : *Statement** → 𝔹

wf-simple-return-statements (*sts*) ≜
  let *returns* = *return-statements-of* (*sts*) in 177
  returns ⊆ *Simple-return-Statement*

wf-function-return-statements : *Typed* × *Block-body* → *Environment* → 𝔹

wf-function-return-statements (*rtype*, *sts*)ρ ≜
  let *returns* = *return-statements-of* (*sts*) in 177
  card *returns* ≥ 1 ∧
  returns ⊆ *Function-return-Statement* ∧
  ∀ *ret* ∈ *returns* ·
    let *etype* = *t-expression* (*ret.expr*)ρ in 190
    is-assignment-compatible (*rtype*, *etype*)ρ 123

annotations Check that the return statement components of the statement sequence are all function return statements and that for each return statement, the return type of the function procedure is assignment-compatible with the type of the expression component of the return statement.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

return-statements-of : *Statement** → *Return-statement*-set

return-statements-of (*sts*) ≙
  let *rsts* = {*st* | *is-Return-statement*(*st*)} in
  x-statements-of-statement-seq (*sts*, *rsts*) 179

annotations The return-statements of a statement sequence is the set containing all the return statements contained in the statements of the statement sequence.

6.6.16.2 Well-formed Retry Statements

no-retry-statements : *Statement** → 𝔹

no-retry-statements (*sts*) ≙
  retry-statements-of (*sts*) = { } 177

annotations Check that there are no **RETRY** statements in a block body.

retry-statements-of : *Statement** → *Retry-statement*-set

retry-statements-of (*sts*) ≙
  let *rsts* = {*st* | *is-Retry-statement*(*st*)} in
  x-statements-of-statement-seq (*sts*, *rsts*) 179

annotations The retry statements of a statement sequence is the set containing all the retry statements contained in the statements of the statement sequence.

6.6.16.3 Well-formed Exit Statements

wf-exit-statements : *Statement** → 𝔹

wf-exit-statements (*sts*) ≙
  free-exit-statements-of-statement-seq (*sts*) = { } 177

annotations This function returns true if and only if each exit statement of the statement sequence is textually contained within a loop statement.

A free exit statement is one that is not contained within a loop statement, thus the loop statements of a statement sequence are well-formed if the statement sequence contains no free exit statements.

free-exit-statements-of-statement-seq : *Statement** → *Exit-statement*-set

free-exit-statements-of-statement-seq (*sts*) ≙
  ⋃ {*free-exit-statements-of-statement* (*st*) | *st* ∈ elems *sts*} 177

annotations The free-exit-statements of a statement sequence is the set containing all the exit statements contained in the statements of the statement sequence that are not contained in loop statements.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

free-exit-statements-of-statement : *Statement* → *Exit-statement*-set

free-exit-statements-of-statement (*st*) ≜
  if *is-Empty-statement*(*st*)
  then { }
  elseif *is-Assignment-statement*(*st*)
  then { }
  elseif *is-Procedure-call*(*st*)
  then { }
  elseif *is-Return-statement*(*st*)
  then { }
  elseif *is-With-statement*(*st*)
  then *free-exit-statements-of-with-statement* (*st*) 178
  elseif *is-If-statement*(*st*)
  then *free-exit-statements-of-if-statement* (*st*) 178
  elseif *is-Case-statement*(*st*)
  then *free-exit-statements-of-case-statement* (*st*) 178
  elseif *is-While-statement*(*st*)
  then *free-exit-statements-of-while-statement* (*st*) 179
  elseif *is-Repeat-statement*(*st*)
  then *free-exit-statements-of-repeat-statement* (*st*) 179
  elseif *is-Loop-statement*(*st*)
  then { }
  elseif *is-Exit-statement*(*st*)
  then {*st*}
  elseif *is-For-statement*(*st*)
  then *free-exit-statements-of-for-statement* (*st*) 179
  else undefined

annotations The free exit statements of an exit statement is a set containing that statement. The free exit statements of any other statement that is not composed of any parts that are themselves statements is the empty set. The free exit statements of a loop statement is the empty set. The free exit statements of any other statement that is composed of parts that are themselves statements is the free exit statements contained in the statement sequence component parts.

free-exit-statements-of-with-statement : *With-statement* → *Exit-statement*-set

free-exit-statements-of-with-statement (*st*) ≜
  let *mk-With-statement* (-, *body*) = *st* in
  free-exit-statements-of-statement-seq (*body*) 177

annotations The free exit statements contained in a with statement are the free exit statements contained in the statement sequence of the with statement.

free-exit-statements-of-if-statement : *If-statement* → *Exit-statement*-set

free-exit-statements-of-if-statement (*st*) ≜
  let *mk-If-statement* (*thens*, *elsep*) = *st* in
  let *thens-set* = ⋃ {*free-exit-statements-of-statement-seq* (*gs.body*) | *gs* ∈ elems *thens*} in 177
  let *elsep-set* = if *elsep* = nil
                    then { }
                    else *free-exit-statements-of-statement-seq* (*elsep*) in 177
  thens-set ∪ *elsep-set*

annotations The free exit statements contained in an if statement are the free exit statements contained in the statement sequence compo ent of the if statement.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

free-exit-statements-of-case-statement : *Case-statement* → *Exit-statement*-set

free-exit-statements-of-case-statement (*st*) ≙
  let *mk-Case-statement* (-, *cases*, *elsep*) = *st* in
  let *cases-set* = ⋃ {*free-exit-statements-of-statement-seq* (*c.body*) | *c* ∈ *cases*} in 177
  let *elsep-set* = if *elsep* = nil
                    then { }
                    else *free-exit-statements-of-statement-seq* (*elsep*) in 177
  cases-set ∪ *elsep-set*

annotations The free exit statements contained in a case statement are the free exit statements contained in the statement sequence components of the case statement.

free-exit-statements-of-while-statement : *While-statement* → *Exit-statement*-set

free-exit-statements-of-while-statement (*st*) ≙
  let *mk-While-statement* (-, *body*) = *st* in
  free-exit-statements-of-statement-seq (*body*) 177

annotations The free exit statements contained in a while statement are the free exit statements contained in the statement sequence component of the while statement.

free-exit-statements-of-repeat-statement : *Repeat-statement* → *Exit-statement*-set

free-exit-statements-of-repeat-statement (*st*) ≙
  let *mk-Repeat-statement* (*body*, -) = *st* in
  free-exit-statements-of-statement-seq (*body*) 177

annotations The free exit statements contained in a repeat statement are the free exit statements contained in the statement sequence component of the repeat statement.

free-exit-statements-of-for-statement : *For-statement* → *Exit-statement*-set

free-exit-statements-of-for-statement (*st*) ≙
  let *mk-For-statement* (-, -, -, -, *body*) = *st* in
  free-exit-statements-of-statement-seq (*body*) 177

annotations The free exit statements contained in a for statement are the free exit statements contained in the statement sequence component of the for statement.

6.6.16.4 Statement Classes of Statement Sequences

x-statements-of-statement-seq : *Statement** × *Statement*-set → *Statement*-set

x-statements-of-statement-seq (*sts*, *stats*) ≙
  ⋃ {*x-statements-of-statement* (*st*, *stats*) | *st* ∈ elems *sts*} 179

annotations This function returns a set containing all the statements contained in the statement sequence that are members of the set defined by the second parameter.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

x-statements-of-statement : *Statement* × *Statement*-set → *Statement*-set

x-statements-of-statement (*st*, *stats*) ≜
  let *ast* = if *st* ∈ *stats*
    then {*st*}
    else { } in
  let *rst* = if *is-With-statement*(*st*)
    then *x-statements-of-with-statement* (*st*, *stats*) 180
    elseif *is-If-statement*(*st*)
    then *x-statements-of-if-statement* (*st*, *stats*) 180
    elseif *is-Case-statement*(*st*)
    then *x-statements-of-case-statement* (*st*, *stats*) 180
    elseif *is-While-statement*(*st*)
    then *x-statements-of-while-statement* (*st*, *stats*) 180
    elseif *is-Repeat-statement*(*st*)
    then *x-statements-of-repeat-statement* (*st*, *stats*) 181
    elseif *is-Loop-statement*(*st*)
    then *x-statements-of-loop-statement* (*st*, *stats*) 181
    elseif *is-For-statement*(*st*)
    then *x-statements-of-for-statement* (*st*, *stats*) 181
    else { } in
  ast ∪ *bst*

x-statements-of-with-statement : *With-statement* × *Statement*-set → *Statement*-set

x-statements-of-with-statement (*st*, *stats*) ≜
  let *mk-With-statement* (-, *body*) = *st* in
  x-statements-of-statement-seq (*body*, *stats*) 179

annotations The statements of a with statement that are members of the set defined by the second parameter are obtained from the statement sequence of the with statement.

x-statements-of-if-statement : *If-statement* × *Statement*-set → *Statement*-set

x-statements-of-if-statement (*st*, *stats*) ≜
  let *mk-If-statement* (*thens*, *elsep*) = *st* in
  let *thens-set* = ⋃{*x-statements-of-statement-seq* (*gs.body*, *stats*) | *gs* ∈ elems *thens*} in 179
  let *elsep-set* = if *elsep* = nil
    then { }
    else *x-statements-of-statement-seq* (*elsep*, *stats*) in 179
  thens-set ∪ *elsep-set*

annotations The statements of an if statement that are members of the set defined by the second parameter are obtained from the statement sequences of the if statement.

x-statements-of-case-statement : *Case-statement* × *Statement*-set → *Statement*-set

x-statements-of-case-statement (*st*, *stats*) ≜
  let *mk-Case-statement* (-, *cases*, *elsep*) = *st* in
  let *cases-set* = ⋃{*x-statements-of-statement-seq* (*c.body*, *stats*) | *c* ∈ *cases*} in 179
  let *elsep-set* = if *elsep* = nil
    then { }
    else *x-statements-of-statement-seq* (*elsep*, *stats*) in 179
  cases-set ∪ *elsep-set*

annotations The statements of a case statement that are members of the set defined by the second parameter are obtained from the statement sequences of the case statement.

x-statements-of-while-statement : *While-statement* × *Statement*-set → *Statement*-set

x-statements-of-while-statement (*st*, *stats*) ≜
  let *mk-While-statement* (-, *body*) = *st* in
  x-statements-of-statement-seq (*body*, *stats*) 179

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations The statements of a while statement that are members of the set defined by the second parameter are obtained from the statement sequence of the while statement.

x-statements-of-repeat-statement : *Repeat-statement* × *Statement*-set → *Statement*-set

x-statements-of-repeat-statement (*st*, *stats*) ≙
  let *mk-Repeat-statement* (*body*, -) = *st* in
  x-statements-of-statement-seq (*body*, *stats*) 179

annotations The statements of a repeat statement that are members of the set defined by the second parameter are obtained from the statement sequence of the repeat statement.

x-statements-of-loop-statement : *Loop-statement* × *Statement*-set → *Statement*-set

x-statements-of-loop-statement (*st*, *stats*) ≙
  let *mk-Loop-statement* (*body*) = *st* in
  x-statements-of-statement-seq (*body*, *stats*) 179

annotations The statements of a loop statement that are members of the set defined by the second parameter are obtained from the statement sequence of the loop statement.

x-statements-of-for-statement : *For-statement* × *Statement*-set → *Statement*-set

x-statements-of-for-statement (*st*, *stats*) ≙
  let *mk-For-statement* (-, -, -, -, *body*) = *st* in
  x-statements-of-statement-seq (*body*, *stats*) 179

annotations The statements of a for statement that are members of the set defined by the second parameter are obtained from the statement sequence of the for statement.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.7 Variable Designators

A variable designator denotes a variable. The denoted variable may be a variable introduced in a variable definition or variable declaration, a formal parameter of a procedure, or a component of such a variable.

Concrete Syntax

variable designator = entire designator | indexed designator | selected designator | dereferenced designator ;

Abstract Syntax

types

Variable-designator = *Entire-designator*
| *Indexed-designator*
| *Selected-designator*
| *Dereferenced-designator*

Static Semantics

wf-variable-designator : *Variable-designator* → *Environment* → $\mathbb{B}$

wf-variable-designator (*vdesig*)ρ $\triangleq$
cases *vdesig* :
mk-Entire-designator(-) → *wf-entire-designator* (*vdesig*)ρ, 183
mk-Indexed-designator(-, -) → *wf-indexed-designator* (*vdesig*)ρ, 184
mk-Selected-designator(-, -) → *wf-selected-designator* (*vdesig*)ρ, 185
mk-Dereferenced-designator(-) → *wf-dereferenced-designator* (*vdesig*)ρ 187
end

t-variable-designator : *Variable-designator* → *Environment* → *Variable-typed*

t-variable-designator (*vdesig*)ρ $\triangleq$
cases *vdesig* :
mk-Entire-designator(-) → *t-entire-designator* (*vdesig*)ρ, 183
mk-Indexed-designator(-, -) → *t-indexed-designator* (*vdesig*)ρ, 184
mk-Selected-designator(-, -) → *t-selected-designator* (*vdesig*)ρ, 186
mk-Dereferenced-designator(-) → *t-dereferenced-designator* (*vdesig*)ρ 187
end

Dynamic Semantics

m-variable-designator : *Variable-designator* → *Environment* $\xrightarrow{o}$ *Variable*

m-variable-designator (*vdesig*)ρ $\triangleq$
cases *vdesig*:
mk-Entire-designator(-) → *m-entire-designator*(*vdesig*) ρ, 183
mk-Indexed-designator(-, -) → *m-indexed-designator*(*vdesig*) ρ, 185
mk-Selected-designator(-, -) → *m-selected-designator*(*vdesig*) ρ, 186
mk-Dereferenced-designator(-) → *m-dereferenced-designator*(*vdesig*) ρ 188
end

6.7.1 Entire Designators

An entire designator is a variable designator that is a qualified identifier; it denotes an entire elementary or structured (composite) variable (which may be a formal parameter), or a field of a record variable within a with statement that applies to that variable.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Concrete Syntax

entire designator = qualified identifier ;

Abstract Syntax

types

Entire-designator :: *qid* : *Qualident*

Static Semantics

The qualified identifier of an entire designator shall denote an entire variable (which may be a formal parameter), or a field of a record variable within a with statement that applies to that variable.

The type of an entire designator shall be the type of the entity that is denoted by the qualified identifier.

wf-entire-designator : *Entire-designator* → *Environment* → 𝔹

wf-entire-designator (*mk-Entire-designator*(*qid*))ρ ≙
  wf-qualident (*qid*)ρ ∧ *is-variable-or-field* (*qid*)ρ 53 315

annotations Check that the qualified identifier component is defined or declared as a variable, or is a component of a record if the context is the statement sequence of a with statement, and that any qualification is by a module name.

t-entire-designator : *Entire-designator* → *Environment* → *Variable-typed*

t-entire-designator (*mk-Entire-designator*(*qid*))ρ ≙
  type-of-variable-or-field (*qid*)ρ 315

annotations Return the type of the variable or, if the context is a with statement and the entire designator designates a field of a record, the type of the field.

Dynamic Semantics

If the variable designated by an entire designator is a component of a group of variant fields which has an identified tag field, and the value of the tag field is such that the variant containing that component is not active, an exception shall occur (but need not be raised); otherwise, the variable designated by the entire designator shall be the variable or formal parameter associated with the qualified identifier.

If the designated variable belongs to one or more variants without an identified tag field, the variants shall become active.

m-entire-designator : *Entire-designator* → *Environment* $\xrightarrow{o}$ *Variable*

m-entire-designator (*mk-Entire-designator*(*qid*))ρ ≙
  let *var* = *associated-variable* (*qid*)ρ in 315
  (*is-Variant-variable*(*var*) → def *is-active* = *is-active-component*(*var*) in 186
      if *is-active*
      then return *var*
      else let *handler* = *current-exception-handler* (ρ) in 318
        non-mandatory-detection(INACTIVE-VARIANT) *handler* , 397
  is-Tagged-variable(*var*) → (*make-component-active*(*var*) ; 187
      return *var*),
  others → return *var*)

annotations Return the variable denoted by the qualified identifier. If the variable denoted by the identifier is a field of a variant of a group of variant fields and the tag for that variant fields component exists, then check that the value of the tag is such that it selects that field. If there is no explicit tag, set the implicit tag to select the field.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.7.2 Indexed Designators

An indexed designator is a variable designator that is an array variable designator followed by one or more index expressions, and it denotes a component variable of the indexed array variable, or a component of a component of the indexed array variable.

Concrete Syntax

indexed designator = array variable designator, left bracket, index expression, { comma, index expression }, right bracket ;

array variable designator = variable designator ;

index expression = ordinal expression ;

The concrete syntax allows for a full or abbreviated representation of the indexing of an array variable that is itself designated by an indexed designator. In the abbreviated form, a single comma shall replace the sequence '] [' that occurs in the full form. The abbreviated form and the full form shall be equivalent.

EXAMPLE — Indexing of an indexed designator

A variable designator of the form:
`A[E1][E2][E3]`

may also be written in the following ways:
`A[E1,E2,E3]` `A[E1][E2,E3]` `A[E1,E2][E3]`

Abstract Syntax

types

Indexed-designator :: *desig* : *Variable-designator*
expr : *Expression*

annotations The abstract syntax only uses the full form.

Static Semantics

The array variable designator of an indexed designator shall be a variable designator that is of an array type. The type of the index expression shall be assignment-compatible with the index type of the array type.

The type of an indexed designator shall be the component type of the array type.

wf-indexed-designator : *Indexed-designator* → *Environment* → $\mathbb{B}$

wf-indexed-designator (*mk-Indexed-designator*(*desig*, *expr*))ρ $\triangleq$
wf-variable-designator (*desig*)ρ ∧ 182
wf-expression (*expr*)ρ ∧ 190
let *type* = *t-variable-designator* (*desig*)ρ in 182
(*is-array-type* (*type*)ρ ∨ *is-open-array* (*type*)ρ) ∧ 321 328
let *itype* = *index-type-of* (*type*)ρ in 326
is-assignment-compatible (*itype*, *t-expression* (*expr*)ρ)ρ 123 190

annotations Check the designator and expression components of the indexed designator. Check that the variable denoted by the designator is of an array type, and that the index type given in the definition or declaration of the variable is assignment-compatible with the index expression.

t-indexed-designator : *Indexed-designator* → *Environment* → *Variable-typed*

t-indexed-designator (*mk-Indexed-designator*(*desig*, -))ρ $\triangleq$
let *type* = *t-variable-designator* (*desig*)ρ in 182
component-type-of (*type*)ρ 325

annotations Return the type of the component of the array as given in the definition or declaration of the variable.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Dynamic Semantics

The variable designator shall be evaluated to give an array variable. The index expression shall be evaluated to give an ordinal value. Provided that this ordinal value is a value of the index type of the array variable, the variable designated by the indexed designator shall be the component of the array variable indexed by the ordinal value; otherwise an exception shall be raised.

The order of evaluation of the variable designator and the index expression of an indexed designator shall be implementation-dependent.

NOTE — If a function call appears in the index expression, the evaluation of the indexed designator could produce a transformation of the program state, i.e. have side-effects.

m-indexed-designator : *Indexed-designator* → *Environment* $\xrightarrow{o}$ *Variable*

m-indexed-designator (*mk-Indexed-designator*(*desig*, *expr*))ρ $\triangleq$
  def *mk-Array-variable*(*array*) = *m-variable-designator*(*desig*) ρ in 182
  def *index* = *m-expression*(*expr*) ρ in 190
  if *index* $\in$ dom *array*
  then return *array* (*index*)
  else let *handler* = *current-exception-handler* (ρ) in 318
    mandatory-detection(INDEX-RANGE) *handler* 396

annotations The above evaluation order for the array variable and the index expression is one of those permitted; the index expression may instead be evaluated before the array variable, or both may be evaluated in parallel — see 4.1.

6.7.3 Selected Designators

A selected designator is a variable designator that is a record variable designator followed by a field identifier, and it denotes a component variable of a variable of a record type.

Concrete Syntax

selected designator = record variable designator, period, field identifier ;

record variable designator = variable designator ;

field identifier = identifier ;

Abstract Syntax

types

Selected-designator :: *desig* : *Variable-designator*
  id : *Identifier*

Static Semantics

The record variable designator of a selected designator shall be a variable designator that is of a record type. The field identifier shall be one of the identifiers or tag identifiers of the field list of that record type.

The type of a selected designator shall be the type of the field identifier.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wf-selected-designator : *Selected-designator* → *Environment* → 𝔹

wf-selected-designator (*mk-Selected-designator*(*desig*, *id*))ρ ≜
  wf-variable-designator (*desig*)ρ ∧ 182
  let *type* = *t-variable-designator* (*desig*)ρ in 182
  is-record-type (*type*)ρ ∧ 321
  id ∈ dom *field-types-of-record* (*type*)ρ

annotations Check the designator component and that it designates a record structure. Check that identifier component is one of the field list identifiers of the record.

t-selected-designator : *Selected-designator* → *Environment* → *Typed*

t-selected-designator (*mk-Selected-designator*(*desig*, *id*))ρ ≜
  let *type* = *t-variable-designator* (*desig*)ρ in 182
  let *types* = *field-types-of-record* (*type*)ρ in 326
  types (*id*)

annotations Return the type of the field corresponding to the field list identifier.

Dynamic Semantics

The variable designator shall be evaluated to give a record variable. If the field identifier denotes a component of the record variable's type that belongs to a variant of a group of variant fields that has a tag field, and the variant is not active, an exception shall occur (but need not be raised); otherwise the variable designated by the selected designator shall be the component of the record variable selected by the field identifier.

If the designated variable belongs to one or more variants without an identified tag field, the variants shall become active.

m-selected-designator : *Selected-designator* → *Environment* $\xrightarrow{o}$ *Variable*

m-selected-designator (*mk-Selected-designator*(*desig*, *id*))ρ ≜
  def *mk-Record-variable*(*fields*) = *m-variable-designator*(*desig*) ρ in 182
  let *component* = *fields* (*id*) in
  (*is-Variant-variable*(*component*) → def *is-active* = *is-active-component*(*component*) in 186
      if *is-active*
      then return *component*
      else let *handler* = *current-exception-handler* (ρ) in 318
        non-mandatory-detection(INACTIVE-VARIANT) *handler* , 397
   is-Tagged-variable(*component*) → (*make-component-active*(*component*) ; 187
      return *component*),
   others → return *component*)

annotations Evaluate the designator and select the appropriate field of the record. If the field is a variant component of a variant fields and the tag for that variant fields component exists, then check that the value of the tag is such that it selects that field. If there is no explicit tag, set the implicit tag to select the field.

Auxiliaries

is-active-component : *Variable* $\xrightarrow{o}$ 𝔹

is-active-component (*component*) ≜
  if *is-Variant-variable*(*component*)
  then let *mk-Variant-variable*(*var*, *tagloc*, *tagvals*) = *component* in
    def *tagval* = *variable-value*(*tagloc*) , 224
      active-var = *is-active-component*(*var*) in 186
    return *tagval* ∈ *tagvals* ∧ *active-var*
  else return true

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations A field of a record may occur in a variant that is nested in several variants. First check that the outer variant is an active variant; this is done by checking that the value of the tag field is consistent with the field component that is referenced. The function is called recursively to check the next (inner) variant.

make-component-active : *Variable* $\xrightarrow{o}$ ()

make-component-active (*component*) $\triangleq$
  if *is-Tagged-variable*(*component*)
  then let *mk-Tagged-variable*(*var*, *tagvar*, *tagvals*) = *component* in
    let *val* $\in$ *tagvals* in
    (*assign*(*tagvar*, *val*) ; 150
     make-component-active(*var*)) 187
  else skip

annotations A field of a record may occur in a variant that is nested in several variants. The outermost variant is made active by assigning an appropriate value to the tag field associated with this variant. The function is called recursively to make the next innermost variant active.

6.7.4 Dereferenced Designators

A dereferenced designator is a variable designator that is a pointer variable designator followed by a dereferencing operator, and it denotes the variable (if any) that is referenced by the value of the pointer variable designator.

Concrete Syntax

dereferenced designator = pointer variable designator, dereferencing operator ;

pointer variable designator = variable designator ;

Abstract Syntax

types

Dereferenced-designator :: *desig* : *Variable-designator*

Static Semantics

The pointer variable designator of a dereferenced designator shall be a variable designator that is of a pointer type.

The type of a dereferenced designator shall be the bound type specified in the declaration of the pointer type.

wf-dereferenced-designator : *Dereferenced-designator* $\rightarrow$ *Environment* $\rightarrow$ $\mathbb{B}$

wf-dereferenced-designator (*mk-Dereferenced-designator*(*desig*))ρ $\triangleq$
  wf-variable-designator (*desig*)ρ $\wedge$ 182
  let *type* = *t-variable-designator* (*desig*)ρ in 182
  is-pointer-type (*type*)ρ 322

annotations Check the designator and that it designates a variable of a pointer type. Outside of the implementation module in which it is defined, an opaque type is not a pointer type.

t-dereferenced-designator : *Dereferenced-designator* $\times$ *Environment* $\rightarrow$ *Typed*

t-dereferenced-designator (*mk-Dereferenced-designator*(*desig*))ρ $\triangleq$
  let *type* = *t-variable-designator* (*desig*)ρ in 182
  bound-type-of (*type*)ρ 326

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Dynamic Semantics

The variable designator shall be evaluated to give a variable P. Provided that the value of the variable P denotes a variable, the variable designated by the dereferenced designator shall be the value of the variable P.

An exception shall be raised if the value of the variable P is the value of the pervasive identifier `NIL`.

An exception shall occur (but need not be raised) if the value of the variable P is a value associated with non-existing storage.

An exception shall occur (but need not be raised) if the value of the variable P is a pointer to a variable of the wrong type.

NOTE — Since the address type is assignment compatible with all pointer types, the value of the variable P may be a pointer to a variable of the wrong type if it has been assigned through a variable or parameter of the address type.

m-dereferenced-designator : *Dereferenced-designator* → *Environment* $\xrightarrow{o}$ *Variable*

m-dereferenced-designator (*mk-Dereferenced-designator*(*desig*))ρ $\triangleq$
  def *mk-Pointer-variable* (*loc*, *vtype*) = *m-variable-designator*(*desig*) ρ in 182
  def *address* = *value-of-a-variable*(*loc*) in 224
  let *handler* = *current-exception-handler* (ρ) in 318
   cases *address*:
    mk-Address-value (*addr*, *atype*, -) →
      def *valid* = *exists-loc*(*addr*) in 335
      (*valid* ∧ *matching-pointer-type* (*vtype*, *atype*)ρ → 188
          return *addr*,
       ¬ *valid* →
          non-mandatory-detection(NONEXISTENT) *handler* , 397
       ¬ *matching-pointer-type* (*vtype*, *atype*)ρ → 188
          non-mandatory-detection(WRONG-TYPE) *handler*), 397
    NIL-VALUE →
      mandatory-detection(NIL-DEREFERENCE) *handler* 396
  end

annotations Check that the value of the pointer is such that it denotes a variable that exists, and that the type of the variable it denotes matches the type of the pointer.

matching-pointer-type : *Typed* × *Typed* → *Environment* → $\mathbb{B}$

matching-pointer-type (*ptype*, *atype*)ρ $\triangleq$
  ptype = *atype*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.8 Expressions

An expression is a formula involving operators and operands whose computation produces a value. However, an expression does not define a value if the computation does not terminate, or if an exception is raised during the computation, such as might arise from using an undefined value, or from dividing by zero.

CHANGES

1 This part of ISO/IEC 10514 does not allow arithmetic on values of the address type (except via functions imported from `SYSTEM`).

2 In *Programming in Modula-2* there are no predefined types for complex values, nor is any provision made for arithmetic on complex values.

Concrete Syntax

expression = simple expression, [relational operator, simple expression] ;

simple expression = [sign], term, { term operator, term } ;

term = factor, { factor operator, factor } ;

factor =
    left parenthesis, expression, right parenthesis | logical negation operator, factor |
    value designator | function call |
    value constructor | constant literal ;

Within one expression, operators shall be applied in decreasing order of precedence. If there are two or more operators of the same precedence within a single expression, simple expression, or term, these operators shall be applied from left to right.

NOTES

1 The concrete syntax does not permit an expression to have two operators juxtaposed.

2 The concrete syntax defines four levels of operator precedence. From highest to lowest these are: logical negation operator, factor operators, term operators and signs, and relational operators.

3 Parentheses can be used in an expression to nest another expression, and they can be used to override the operator precedence.

Abstract Syntax

types

Expression = *Infix-expression*
| *Prefix-expression*
| *Value-designator*
| *Function-call*
| *Value-constructor*
| *Constant-literal*

annotations An expression in the concrete representation may include simple expressions and/or terms containing more than one operator. Such an expression is an abbreviation of a fully-parenthesized expression whose simple expressions contain at most one operator (a sign or a term operator) and whose terms contain at most one factor operator. The abstract syntax for expressions models fully-parenthesized expressions.

For example, the expression `a*b+c` is an abbreviation of the fully-parenthesized expression `(a*b)+c`. It is modelled in the abstract syntax by an *Expression* that is an *InfixExpression*.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Static Semantics

wf-expression : *Expression* → *Environment* → $\mathbb{B}$

wf-expression (*expr*)ρ $\triangleq$
(*is-Infix-expression*(*expr*) → *wf-infix-expression* (*expr*)ρ, 191
is-Prefix-expression(*expr*) → *wf-prefix-expression* (*expr*)ρ, 219
is-Value-designator(*expr*) → *wf-value-designator* (*expr*)ρ, 223
is-Function-call(*expr*) → *wf-function-call* (*expr*)ρ, 229
is-Value-constructor(*expr*) → *wf-value-constructor* (*expr*)ρ, 231
is-Constant-literal(*expr*) → *wf-constant-literal* (*expr*)ρ) 248

t-expression : *Expression* → *Environment* → *Expression-typed*

t-expression (*expr*)ρ $\triangleq$
(*is-Infix-expression*(*expr*) → *t-infix-expression* (*expr*), 191
is-Prefix-expression(*expr*) → *t-prefix-expression* (*expr*), 219
is-Value-designator(*expr*) → *t-value-designator* (*expr*)ρ, 223
is-Function-call(*expr*) → *t-function-call* (*expr*)ρ, 230
is-Value-constructor(*expr*) → *t-value-constructor* (*expr*)ρ, 231
is-Constant-literal(*expr*) → *t-constant-literal* (*expr*)ρ) 248

Dynamic Semantics

m-expression : *Expression* → *Environment* $\xrightarrow{o}$ *Value*

m-expression (*expr*)ρ $\triangleq$
(*is-Infix-expression*(*expr*) → *m-infix-expression*(*expr*) ρ, 191
is-Prefix-expression(*expr*) → *m-prefix-expression*(*expr*) ρ, 219
is-Value-designator(*expr*) → *m-value-designator*(*expr*) ρ, 223
is-Function-call(*expr*) → *m-function-call*(*expr*) ρ, 230
is-Value-constructor(*expr*) → *m-value-constructor*(*expr*) ρ, 231
is-Constant-literal(*expr*) → *m-constant-literal*(*expr*) ρ) 248

6.8.1 Ordinal Expressions

An ordinal expression is an expression that produces an ordinal result. There are numerous places where this restriction is needed, and the concrete syntax has been designed to make this readily apparent when this occurs.

Concrete Syntax

ordinal expression = expression ;

6.8.2 Infix Expressions and Operations

An infix expression is an expression whose value is obtained by applying an infix operator to the values of two operands.

Concrete Syntax

relational operator =
equals operator | inequality operator | less than operator |
greater than operator | less than or equal operator | subset operator |
greater than or equal operator | superset operator | set membership operator ;

term operator =
plus operator | set union operator | minus operator |

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

set difference operator | logical disjunction operator | string catenate symbol ;

factor operator =
multiplication operator | set intersection operator | division operator |
symmetric set difference operator | rem operator | div operator |
mod operator | logical conjunction operator ;

NOTE 1 — The syntax for the various forms of infix operators is given in the Lexis — see section 5.

Abstract Syntax

types

Infix-expression :: *left* : *Expression*
op : *Infix-operation*
right : *Expression*

Static Semantics

For any operand (of an infix expression) that is of the $\mathbb{Z}$-type, the $\mathbb{R}$-type or the $\mathbb{C}$-type, the value of the operand shall be within the range of values defined by the type of that infix expression. The type of an infix expression shall be a scalar type, a set type, a packedset type, or a string literal type.

wf-infix-expression : *Infix-expression* → *Environment* → $\mathbb{B}$

wf-infix-expression (*mk-Infix-expression* (*left*, *op*, *right*))ρ $\triangleq$
wf-expression (*left*)ρ ∧ 190
wf-expression (*right*)ρ ∧ 190
let *ltype* = *t-expression* (*left*)ρ in 190
let *rtype* = *t-expression* (*right*)ρ in 190
wf-infix-operation (*op*, *ltype*, *rtype*)ρ ∧ 192
wf-constants-in-expression (*left*, *right*)ρ 192

t-infix-expression : *Infix-expression* → *Expression-typed*

t-infix-expression (*mk-Infix-expression* (-, *op*, -)) $\triangleq$
t-infix-operation (*op*) 193

Dynamic Semantics

If the operator of an infix expression is not a Boolean operator, the two expressions shall be evaluated, and then the infix operator shall be applied.

NOTE 2 — The evaluation order that applies when the operator is a Boolean operator is given in 6.8.2.4.

m-infix-expression : *Infix-expression* → *Environment* $\xrightarrow{o}$ *Value*

m-infix-expression (*mk-Infix-expression* (*left*, *op*, *right*))ρ $\triangleq$
if *is-Boolean-operation*(*op*)
then def *lvalue* = *m-expression*(*left*) ρ in 190
m-infix-operation(*op*, *lvalue*, *right*) ρ 193
else def *lvalue* = *m-expression*(*left*) ρ in 190
def *rvalue* = *m-expression*(*right*) ρ in 190
m-infix-operation(*op*, *lvalue*, *rvalue*) ρ 193

annotations In the case of an infix operator other than a Boolean operator, the above evaluation order for the left operand and the right operand is one of those permitted; the right operand may instead be evaluated before the left operand, or both may be evaluated in parallel — see 4.1.

In the case of a Boolean operator, whether or not the right operand is evaluated depends upon the value of the left operand: the left operand is evaluated first, and, if necessary, the right operand is evaluated and the infix operation performed.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Auxiliaries

wf-constants-in-expression : *Expression* × *Expression* → *Environment* → 𝔹

wf-constants-in-expression (*left*, *right*)ρ ≙
  let *ltype* = *t-expression* (*left*)ρ in 190
  let *rtype* = *t-expression* (*right*)ρ in 190
  if *ltype* = ℤ-TYPE ∨ *ltype* = ℝ-TYPE ∨ *ltype* = ℂ-TYPE
  then let *lvalue* = *evaluate-constant-expression* (*left*)ρ in 257
    in-range (*rtype*, *lvalue*) 192
  elseif *rtype* = ℤ-TYPE ∨ *rtype* = ℝ-TYPE ∨ *rtype* = ℂ-TYPE
  then let *rvalue* = *evaluate-constant-expression* (*right*)ρ in 257
    in-range (*ltype*, *rvalue*) 192
  else true

annotations For any operand (of an infix expression) that is of the ℤ-type, the ℝ-type or the ℂ-type, check that its value is within the range of values defined by the type of that infix expression.

in-range : *Typed* × *Value* → 𝔹

in-range (*type*, *value*) ≙
  if *type* = COMPLEX-TYPE
  then *re-min-complex-value* ≤ re*value* ∧ re*value* ≤ *re-max-complex-value* ∧
    im-min-complex-value ≤ im*value* ∧ im*value* ≤ *im-max-complex-value*
  elseif *type* = LONG-COMPLEX-TYPE
  then *re-min-long-complex-value* ≤ re*value* ∧ re*value* ≤ *re-max-long-complex-value* ∧
    im-min-long-complex-value ≤ im*value* ∧ im*value* ≤ *im-max-long-complex-value*
  elseif *type* = ℂ-TYPE
  then *re-min-cc-value* ≤ re*value* ∧ re*value* ≤ *re-max-cc-value* ∧
    im-min-cc-value ≤ im*value* ∧ im*value* ≤ *im-max-cc-value*
  else *minimum-of-basic-type* (*type*) ≤ *value* ∧ *value* ≤ *maximum-of-basic-type* (*type*) 276 276

Infix Operations

Abstract Syntax

types

Infix-operation = *Complex-number-operation*
  | *Real-number-operation*
  | *Whole-number-operation*
  | *Boolean-operation*
  | *Set-operation*
  | *String-literal-operation*
  | *Relational-operation*

annotations For convenience of the description, infix operations are divided into seven classes.

Static Semantics

wf-infix-operation : *Infix-operation* × *Expression-typed* × *Expression-typed* → *Environment* → 𝔹

wf-infix-operation (*op*, *ltype*, *rtype*)ρ ≙
  (*is-Complex-number-operation*(*op*) → *wf-complex-number-operation* (*op*, *ltype*, *rtype*)ρ, 193
  is-Real-number-operation(*op*) → *wf-real-number-operation* (*op*, *ltype*, *rtype*)ρ, 196
  is-Whole-number-operation(*op*) → *wf-whole-number-operation* (*op*, *ltype*, *rtype*)ρ, 199
  is-Boolean-operation(*op*) → *wf-Boolean-operation* (*op*, *ltype*, *rtype*)ρ, 202
  is-Set-operation(*op*) → *wf-set-operation* (*op*, *ltype*, *rtype*)ρ, 203
  is-String-literal-operation(*op*) → *wf-string-literal-operation* (*op*, *ltype*, *rtype*)ρ, 205
  is-Relational-operation(*op*) → *wf-relational-operation* (*op*, *ltype*, *rtype*)ρ) 206

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

t-infix-operation : *Infix-operation* → *Expression-typed*

t-infix-operation (*op*) ≙
  (*is-Complex-number-operation*(*op*) → *t-complex-number-operation* (*op*), 193
  is-Real-number-operation(*op*) → *t-real-number-operation* (*op*), 196
  is-Whole-number-operation(*op*) → *t-whole-number-operation* (*op*), 199
  is-Boolean-operation(*op*) → *t-Boolean-operation* (*op*), 202
  is-Set-operation(*op*) → *t-set-operation* (*op*), 203
  is-String-literal-operation(*op*) → *t-string-literal-operation* (*op*), 205
  is-Relational-operation(*op*) → *t-relational-operation* (*op*)) 206

Dynamic Semantics

m-infix-operation : *Infix-operation* × *Value* × (*Value* | *Expression*) → *Environment* $\xrightarrow{o}$ *Value*

m-infix-operation (*op*, *lval*, *rval*)ρ ≙
  let *handler* = *current-exception-handler* (ρ) in 318
  (*is-Complex-number-operation*(*op*) → *m-complex-number-operation*(*op*, *lval*, *rval*) *handler*, 194
  is-Real-number-operation(*op*) → *m-real-number-operation*(*op*, *lval*, *rval*) *handler*, 196
  is-Whole-number-operation(*op*) → *m-whole-number-operation*(*op*, *lval*, *rval*) *handler*, 199
  is-Boolean-operation(*op*) → *m-Boolean-operation*(*op*, *lval*, *rval*) ρ, 202
  is-Set-operation(*op*) → *m-set-operation*(*op*, *lval*, *rval*) , 203
  is-String-literal-operation(*op*) → *m-string-literal-operation*(*op*, *lval*, *rval*) , 206
  is-Relational-operation(*op*) → *m-relational-operation*(*op*, *lval*, *rval*)) 207

6.8.2.1 Complex Number Infix Operations

For operands of a complex number type, there are four arithmetic infix operations: addition, subtraction, multiplication, and division. These operations are denoted lexically by the symbols '+', '-', '*', and '/' respectively. The result of a complex number infix operation is an approximation to the result of the corresponding mathematical operation.

Abstract Syntax

types

Complex-number-operation :: *op* : *Complex-number-operator*
  otype : *Complex-number-type*

Static Semantics

A complex number infix operation whose operands are both of the ℂ-type has a result that shall be of the ℂ-type; a complex number infix operation with one operand that is of the ℂ-type has a result that shall be of the same type as the other operand, which shall be of a complex number type. Otherwise both operands shall be of the identical complex number type, and the result shall also be of this type.

NOTE 1 — Given the types of two operands, the following table indicates the type of the result for those combinations that are permitted.

	complex type	long complex type	ℂ-type
complex type	complex type	error	complex type
long complex type	error	long complex type	long complex type
ℂ-type	complex type	long complex type	ℂ-type

wf-complex-number-operation :
  Complex-number-operation × *Expression-typed* × *Expression-typed* → *Environment* → 𝔹

wf-complex-number-operation (*mk-Complex-number-operation* (-, *otype*), *ltype*, *rtype*)ρ ≙
  is-expression-compatible (*ltype*, *rtype*)ρ ∧ 121
  otype = *complex-result-type* (*ltype*, *rtype*) 194

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

t-complex-number-operation : *Complex-number-operation* → *Expression-typed*

t-complex-number-operation (*mk-Complex-number-operation* (-, *otype*)) ≜
otype

Dynamic Semantics

A complex number infix operation shall be performed with an accuracy that is determined by the types of its operands. This accuracy shall not decrease from the complex type to the long complex type, or from the long complex type to the ℂ-type.

An exception shall be raised if a complex number infix operation leads to overflow. The value of a complex number infix operation that leads to underflow shall be an approximation to the mathematically correct result of the operation.

NOTES

2 Underflow does not cause an exception to be raised.

3 'Gradual underflow' is different from 'underflow'. It is defined in the Glossary, and discussed in 8.2.

m-complex-number-operation : *Complex-number-operation* × *Value* × *Value* → *Exception-handler* $\xrightarrow{o}$ *Value*

m-complex-number-operation (*mk-Complex-number-operation* (*op*, *otype*), *lvalue*, *rvalue*)*handler* ≜
def *c* = *m-complex-number-operator*(*op*, *lvalue*, *rvalue*) *handler* in 195
get-complex-result(*otype*, *c*) *handler* 194

Auxiliaries

complex-result-type : *Complex-number-type* × *Complex-number-type* → *Complex-number-type*

complex-result-type (*ltype*, *rtype*) ≜
(*ltype* = *rtype* → *ltype*,
ltype = ℂ-TYPE → *rtype*,
rtype = ℂ-TYPE → *ltype*)

annotations The result of the function is undefined if the two parameters are not the same complex number type (unless at least one of the parameters is the ℂ-type).

get-complex-result : *Complex-number-type* × ℂ → *Exception-handler* $\xrightarrow{o}$ ℂ

get-complex-result (*type*, *r*)*handler* ≜
if *in-range* (*type*, *r*) 192
then cases *type*:
COMPLEX-TYPE → return *complex-approximation* (*r*), 194
LONG-COMPLEX-TYPE → return *long-complex-approximation* (*r*) , 194
ℂ-TYPE → return *complex-constant-approximation* (*r*) 194
end
else *mandatory-detection*(COMPLEX-OVERFLOW) *handler* 396

annotations The functions *complex-approximation*, *long-complex-approximation*, and *complex-constant-approximation* are implementation-defined and give approximations to the corresponding mathematical result. The order of non-decreasing accuracy of these functions is: *complex-approximation*, *long-complex-approximation*, *complex-constant-approximation*.

Complex Number Infix Operators

Abstract Syntax

types

Complex-number-operator = ADD | SUBTRACT | MULTIPLY | DIVIDE

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations The above operators correspond to '+', '-', '*', and '/' respectively.

Dynamic Semantics

m-complex-number-operator : *Complex-number-operator* × *Value* × *Value* → *Exception-handler* $\xrightarrow{o}$ *Value*

m-complex-number-operator (*op*, *lvalue*, *rvalue*)*handler* $\triangleq$
  cases *op*:
    ADD → *m-complex-add*(*lvalue*, *rvalue*) , 195
    SUBTRACT→ *m-complex-subtract*(*lvalue*, *rvalue*) , 195
    MULTIPLY → *m-complex-multiply*(*lvalue*, *rvalue*) , 195
    DIVIDE → *m-complex-divide*(*lvalue*, *rvalue*) *handler* 195
  end

The evaluation of the left and right operands of a complex number infix operation shall result in two values, denoted by `lval` and `rval`.

The value of the complex number infix operation '`lval + rval`' shall be the value that results from the addition of the two values `lval` and `rval`.

The value of the complex number infix operation '`lval - rval`' shall be the value that results from subtracting the value `rval` from the value `lval`.

The value of the complex number infix operation '`lval * rval`' shall be the value that results from the multiplication of the two values `lval` and `rval`.

Provided that the magnitude of `rval` is non-zero, the value of the complex number infix operation '`lval / rval`' shall be the value that results from the division of the value `lval` by the value `rval`; otherwise an exception shall be raised.

m-complex-add : *Value* × *Value* $\xrightarrow{o}$ *Value*

m-complex-add (*lvalue*, *rvalue*) $\triangleq$
  return *lvalue* + *rvalue*

m-complex-subtract : *Value* × *Value* $\xrightarrow{o}$ *Value*

m-complex-subtract (*lvalue*, *rvalue*) $\triangleq$
  return *lvalue* − *rvalue*

m-complex-multiply : *Value* × *Value* $\xrightarrow{o}$ *Value*

m-complex-multiply (*lvalue*, *rvalue*) $\triangleq$
  return *lvalue* × *rvalue*

m-complex-divide : *Value* × *Value* → *Exception-handler* $\xrightarrow{o}$ *Value*

m-complex-divide (*lvalue*, *rvalue*)*handler* $\triangleq$
  if *rvalue* ≠ 0
  then return *lvalue*/*rvalue*
  else *mandatory-detection*(COMPLEX-ZERO-DIVISION) *handler* 396

annotations The operations +, −, ×, and / that are used in the above definitions are the mathematical ones, which are of infinite precision. Hence the finite precision of actual implementations is mirrored by infinite precision followed by an approximation operation (performed by *get-complex-result*).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.8.2.2 Real Number Infix Operations

For operands of a real number type, there are four arithmetic infix operations: addition, subtraction, multiplication, and division. These operations are denoted lexically by the symbols '+', '-', '*', and '/' respectively. The result of a real number infix operation is an approximation to the result of the corresponding mathematical operation.

Abstract Syntax

types

Real-number-operation :: *op* : *Real-number-operator*
otype : *Real-number-type*

Static Semantics

A real number infix operation whose operands are both of the $\mathbb{R}$-type has a result that shall be of the $\mathbb{R}$-type; a real number infix operation with one operand that is of the $\mathbb{R}$-type has a result that shall be of the same type as the other operand, which shall be of a real number type. Otherwise both operands shall be of the identical real number type, and the result shall also be of this type.

NOTE 1 — Given the types of two operands, the following table indicates the type of the result for those combinations that are permitted.

	real type	long real type	$\mathbb{R}$-type
real type	real type	error	real type
long real type	error	long real type	long real type
$\mathbb{R}$-type	real type	long real type	$\mathbb{R}$-type

wf-real-number-operation : *Real-number-operation* × *Expression-typed* × *Expression-typed* → *Environment* → $\mathbb{B}$

wf-real-number-operation (*mk-Real-number-operation* (-, *otype*), *ltype*, *rtype*)ρ $\triangleq$
is-expression-compatible (*ltype*, *rtype*)ρ ∧ 121
otype = *real-result-type* (*ltype*, *rtype*) 197

t-real-number-operation : *Real-number-operation* → *Expression-typed*

t-real-number-operation (*mk-Real-number-operation* (-, *otype*)) $\triangleq$
otype

Dynamic Semantics

A real number infix operation shall be performed with an accuracy that is determined by the types of its operands. This accuracy shall not decrease from the real type to the long real type, or from the long real type to the $\mathbb{R}$-type.

An exception shall be raised if a real number infix operation leads to overflow. The value of a real number infix operation that leads to underflow shall be an approximation to the mathematically correct result of the operation.

NOTES

2 Underflow does not cause an exception to be raised.

3 'Gradual underflow' is different from 'underflow'. It is defined in the Glossary, and discussed in 8.2.

m-real-number-operation : *Real-number-operation* × *Value* × *Value* → *Exception-handler* $\xrightarrow{o}$ *Value*

m-real-number-operation (*mk-Real-number-operation* (*op*, *otype*), *lvalue*, *rvalue*)*handler* $\triangleq$
def *r* = *m-real-number-operator*(*op*, *lvalue*, *rvalue*) *handler* in 197
get-real-result(*otype*, *r*) *handler* 197

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Auxiliaries

real-result-type : *Real-number-type* × *Real-number-type* → *Real-number-type*

real-result-type (*ltype*, *rtype*) ≙
  (*ltype* = *rtype* → *ltype*,
   ltype = ℝ-TYPE → *rtype*,
   rtype = ℝ-TYPE → *ltype*)

annotations The result of the function is undefined if the two parameters are not the same real number type (unless at least one of the parameters is the ℝ-type).

get-real-result : *Real-number-type* × ℝ → *Exception-handler* $\xrightarrow{o}$ ℝ

get-real-result (*type*, *r*)*handler* ≙
  if *in-range* (*type*, *r*) 192
  then cases *type*:
    REAL-TYPE → return *real-approximation* (*r*), 197
    LONG-REAL-TYPE → return *long-real-approximation* (*r*), 197
    ℝ-TYPE → return *real-constant-approximation* (*r*) 197
  end
  else *mandatory-detection*(REAL-OVERFLOW) *handler* 396

annotations The functions *real-approximation*, *long-real-approximation*, and *real-constant-approximation* are implementation-defined and give approximations to the corresponding mathematical result. The order of non-decreasing accuracy of these functions is: *real-approximation*, *long-real-approximation*, *real-constant-approximation*.

Real Number Infix Operators

Abstract Syntax

types

Real-number-operator = ADD | SUBTRACT | MULTIPLY | DIVIDE

annotations The above operators correspond to '+', '-', '*', and '/' respectively.

Dynamic Semantics

m-real-number-operator : *Real-number-operator* × *Value* × *Value* → *Exception-handler* $\xrightarrow{o}$ *Value*

m-real-number-operator (*op*, *lvalue*, *rvalue*)*handler* ≙
  cases *op*:
    ADD → *m-real-add*(*lvalue*, *rvalue*) , 198
    SUBTRACT → *m-real-subtract*(*lvalue*, *rvalue*) , 198
    MULTIPLY → *m-real-multiply*(*lvalue*, *rvalue*) , 198
    DIVIDE → *m-real-divide*(*lvalue*, *rvalue*) *handler* 198
  end

The evaluation of the left and right operands of a real number infix operation shall result in two values, denoted by `lval` and `rval`.

The value of the real number infix operation '`lval + rval`' shall be the value that results from the addition of the two values `lval` and `lval`.

The value of the real number infix operation '`lval - rval`' shall be the value that results from subtracting the value `rval` from the value `lval`.

The value of the real number infix operation '`lval * rval`' shall be the value that results from the multiplication of the two values `lval` and `rval`.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Provided that the value `rval` is non-zero, the value of the real number infix operation '`lval / rval`' shall be the value that results from the division of the value `lval` by the value `rval`; otherwise an exception shall be raised.

m-real-add : *Value* × *Value* $\xrightarrow{o}$ *Value*

m-real-add (*lvalue*, *rvalue*) $\triangleq$
  return *lvalue* + *rvalue*

m-real-subtract : *Value* × *Value* $\xrightarrow{o}$ *Value*

m-real-subtract (*lvalue*, *rvalue*) $\triangleq$
  return *lvalue* − *rvalue*

m-real-multiply : *Value* × *Value* $\xrightarrow{o}$ *Value*

m-real-multiply (*lvalue*, *rvalue*) $\triangleq$
  return *lvalue* × *rvalue*

m-real-divide : *Value* × *Value* → *Exception-handler* $\xrightarrow{o}$ *Value*

m-real-divide (*lvalue*, *rvalue*)*handler* $\triangleq$
  if *rvalue* ≠ 0
  then return *lvalue*/*rvalue*
  else *mandatory-detection*(REAL-ZERO-DIVISION) *handler* 396

annotations The operations +, −, ×, and / that are used in the above definitions are the mathematical ones, which are of infinite precision. Hence the finite precision of actual implementations is mirrored by infinite precision followed by an approximation operation (performed by *get-real-result*).

6.8.2.3 Whole Number Infix Operations

For operands of a whole number type, there are seven arithmetic infix operations: addition, subtraction, multiplication, two forms of division, and two forms of remaindering. These operations are denoted lexically by the symbols '`+`', '`-`', '`*`', '`/`', '`DIV`', '`REM`', and '`MOD`' respectively. The result of a whole number infix operation is mathematically correct if it is within the range of values defined by the whole number type.

CHANGE — The operators '`/`' and '`REM`' are not described in *Programming in Modula-2*.

Abstract Syntax

types

Whole-number-operation :: *op* : *Whole-number-operator*
  otype : *Whole-number-type*

Static Semantics

A whole number infix operation whose operands are both of the ℤ-type has a result that shall be of the ℤ-type; a whole number infix operation with one operand that is of the ℤ-type has a result that shall be of the same type as the other operand, which shall be of a whole number type. Otherwise both operands shall be of the identical whole number type, and the result shall also be of this type.

NOTE 1 — Given the types of two operands, the following table indicates the type of the result for those combinations that are permitted.

	signed type	unsigned type	ℤ-type
signed type	signed type	error	signed type
unsigned type	error	unsigned type	unsigned type
ℤ-type	signed type	unsigned type	ℤ-type

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wf-whole-number-operation :
Whole-number-operation × *Expression-typed* × *Expression-typed* → *Environment* → 𝔹

wf-whole-number-operation (*mk-Whole-number-operation* (-, *otype*), *ltype*, *rtype*)ρ ≙
is-expression-compatible (*ltype*, *rtype*)ρ ∧ 121
otype = *whole-number-result-type* (*ltype*, *rtype*) 199

t-whole-number-operation : *Whole-number-operation* → *Expression-typed*

t-whole-number-operation (*mk-Whole-number-operation* (-, *otype*)) ≙
otype

Dynamic Semantics

The value of a whole number infix operation shall be the mathematically correct result of the operation if the result is within the range of values defined by the result type.

An exception shall occur, and may be raised, if the result is not within the range of values defined by the result type.

NOTE 2 — If the result of a whole number infix operation is not within the range of values defined by the result type, an implementation for which the result type conforms to the requirements of ISO/IEC 10967-1 will raise the whole number overflow exception.

m-whole-number-operation : *Whole-number-operation* × *Value* × *Value* → *Exception-handler* $\xrightarrow{o}$ *Value*

m-whole-number-operation (*mk-Whole-number-operation* (*op*, *otype*), *lvalue*, *rvalue*)*handler* ≙
def *n* = *m-whole-number-operator*(*op*, *lvalue*, *rvalue*) *handler* in 200
get-whole-result(*otype*, *n*) *handler* 199

Auxiliaries

whole-number-result-type : *Whole-number-type* × *Whole-number-type* → *Whole-number-type*

whole-number-result-type (*ltype*, *rtype*) ≙
(*ltype* = *rtype* → *ltype*,
ltype = ℤ-TYPE → *rtype*,
rtype = ℤ-TYPE → *ltype*)

annotations The result of the function is undefined if the two parameters are not the same whole number type (unless at least one of the parameters is the ℤ-type).

get-whole-result : *Whole-number-type* × ℤ → *Exception-handler* $\xrightarrow{o}$ ℤ

get-whole-result (*type*, *n*)*handler* ≙
if *in-range* (*type*, *n*) 192
then return *n*
else *non-mandatory-detection*(WHOLE-OVERFLOW) *handler* 397

Whole Number Infix Operators

Abstract Syntax

types

Whole-number-operator = ADD | SUBTRACT | MULTIPLY | DIVIDE | REM | DIV | MOD

annotations The above operators correspond to '+', '-', '*', '/', 'REM', 'DIV', and 'MOD' respectively.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Dynamic Semantics

m-whole-number-operator : *Whole-number-operator* × *Value* × *Value* → *Exception-handler* $\xrightarrow{o}$ *Value*

m-whole-number-operator (*op*, *lvalue*, *rvalue*)*handler* $\triangleq$

```
cases op:
  ADD       → m-whole-add(lvalue, rvalue) ,                          201
  SUBTRACT→ m-whole-subtract(lvalue, rvalue) ,                       201
  MULTIPLY → m-whole-multiply(lvalue, rvalue) ,                      201
  DIVIDE    → m-whole-divide(lvalue, rvalue) handler,                201
  REM       → m-whole-rem(lvalue, rvalue) handler,                   201
  DIV       → m-whole-div(lvalue, rvalue) handler,                   201
  MOD       → m-whole-mod(lvalue, rvalue) handler                    201
end
```

The evaluation of the left and right operands of a whole number infix operation shall result in two values, denoted by `lval` and `rval`.

The value of the whole number infix operation '`lval + rval`' shall be the value that results from the addition of the two values `lval` and `rval`.

The value of the whole number infix operation '`lval - rval`' shall be the value that results from subtracting the value `rval` from the value `lval`.

The value of the whole number infix operation '`lval * rval`' shall be the value that results from the multiplication of the two values `lval` and `rval`.

The evaluation of the whole number infix operation '`lval / rval`' shall raise an exception if `rval` is zero; otherwise the value of the whole number infix operation shall be the value `quotient` which is such that the identity

```
lval = rval * quotient + remainder
```

is satisfied for a value of `remainder` that is either zero, or an integer of the same sign as `lval` and of smaller absolute value than `rval`.

NOTE 3 — If `x` and `y` are whole numbers and `y` is not equal to zero, then `(-x)/y = x/(-y) = - (x/y)`

The evaluation of the whole number infix operation '`lval REM rval`' shall raise an exception if `rval` is zero; otherwise the value of the whole number infix operation shall be the value `remainder` which is such that the identity

```
lval = rval * quotient + remainder
```

is satisfied for an integral value of `quotient`, and a value of `remainder` that is either zero, or an integer of the same sign as `lval` and of smaller absolute value than `rval`.

The evaluation of the whole number infix operation '`lval DIV rval`' shall raise an exception if `rval` is zero or negative; otherwise the value of the whole number infix operation shall be the value `quotient` which is such that the identity

```
lval = rval * quotient + modulus
```

is satisfied for a value of `modulus` that is a non-negative integer less than `rval`.

The evaluation of the whole number infix operation '`lval MOD rval`' shall raise an exception if `rval` is zero or negative; otherwise the value of the whole number infix operation shall be the value `modulus` which is such that the identity

```
lval = rval * quotient + modulus
```

is satisfied for an integral value of `quotient`, and a value of `modulus` that is a non-negative integer less than `rval`.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NOTE 4 — These operations are exemplified by the following table.

op	$31 op 10$	$31 op (-10)$	$(-31) op 10$	$(-31) op (-10)$
/	3	−3	−3	3
REM	1	1	−1	−1
DIV	3	Exception	−4	Exception
MOD	1	Exception	9	Exception

m-whole-add : $Value \times Value \xrightarrow{o} Value$

m-whole-add $(lvalue, rvalue) \triangleq$
  return $lvalue + rvalue$

m-whole-subtract : $Value \times Value \xrightarrow{o} Value$

m-whole-subtract $(lvalue, rvalue) \triangleq$
  return $lvalue - rvalue$

m-whole-multiply : $Value \times Value \xrightarrow{o} Value$

m-whole-multiply $(lvalue, rvalue) \triangleq$
  return $lvalue \times rvalue$

m-whole-divide : $Value \times Value \rightarrow Exception\text{-}handler \xrightarrow{o} Value$

m-whole-divide $(lvalue, rvalue) handler \triangleq$
  if $rvalue \neq 0$
  then let $quotient : \mathbb{Z}$ be st $\exists\, remainder : \mathbb{Z} \cdot \text{abs}\ remainder < \text{abs}\ rvalue \wedge$
    $sign\,(remainder) = sign\,(lvalue) \wedge$ 615 615
    $lvalue = quotient \times rvalue + remainder$ in
    return $quotient$
  else *mandatory-detection*(WHOLE-ZERO-DIVISION) *handler* 396

m-whole-rem : $Value \times Value \rightarrow Exception\text{-}handler \xrightarrow{o} Value$

m-whole-rem $(lvalue, rvalue) handler \triangleq$
  if $rvalue \neq 0$
  then let $remainder : \mathbb{Z}$ be st $\exists\, quotient : \mathbb{Z} \cdot \text{abs}\ remainder < \text{abs}\ rvalue \wedge$
    $sign\,(remainder) = sign\,(lvalue) \wedge$ 615 615
    $lvalue = quotient \times rvalue + remainder$ in
    return $remainder$
  else *mandatory-detection*(WHOLE-ZERO-REMAINDER) *handler* 396

m-whole-div : $Value \times Value \rightarrow Exception\text{-}handler \xrightarrow{o} Value$

m-whole-div $(lvalue, rvalue) handler \triangleq$
  if $rvalue > 0$
  then let $quotient : \mathbb{Z}$ be st $\exists\, modulus : \mathbb{N} \cdot lvalue = quotient \times rvalue + modulus \wedge$
    $modulus < rvalue$ in
    return $quotient$
  else *mandatory-detection*(WHOLE-NONPOS-DIV) *handler* 396

m-whole-mod : $Value \times Value \rightarrow Exception\text{-}handler \xrightarrow{o} Value$

m-whole-mod $(lvalue, rvalue) handler \triangleq$
  if $rvalue > 0$
  then let $modulus : \mathbb{N}$ be st $\exists\, quotient : \mathbb{Z} \cdot lvalue = quotient \times rvalue + modulus \wedge$
    $modulus < rvalue$ in
    return $modulus$
  else *mandatory-detection*(WHOLE-NONPOS-MOD) *handler* 396

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.8.2.4 Boolean Infix Operations

For operands of the Boolean type, there are two Boolean infix operations: logical conjunction and logical disjunction. These operations are denoted lexically by the symbols '**AND**' and '**OR**' respectively.

NOTE — The symbol '**&**' may be used as a synonym for '**AND**'.

Abstract Syntax

types

Boolean-operation :: *op* : *Boolean-operator*

Static Semantics

Both operands of a Boolean infix operation shall be of the Boolean type, and the result shall also be of the Boolean type.

wf-Boolean-operation : *Boolean-operation* × *Expression-typed* × *Expression-typed* → *Environment* → $\mathbb{B}$

wf-Boolean-operation (*mk-Boolean-operation* (-), *ltype*, *rtype*)ρ $\triangleq$
  ltype = BOOLEAN-TYPE ∧ *rtype* = BOOLEAN-TYPE

t-Boolean-operation : *Boolean-operation* → *Expression-typed*

t-Boolean-operation (*mk-Boolean-operation* (-)) $\triangleq$
  BOOLEAN-TYPE

Dynamic Semantics

m-Boolean-operation : *Boolean-operation* × *Value* × *Expression* → *Environment* $\xrightarrow{o}$ $\mathbb{B}$

m-Boolean-operation (*mk-Boolean-operation* (*op*), *lvalue*, *right*)ρ $\triangleq$
  m-Boolean-operator(*op*, *lvalue*, *right*) ρ 202

Boolean Infix Operators

Abstract Syntax

types

Boolean-operator = AND | OR

annotations The above operators correspond to '**AND**' (or '**&**') and '**OR**' respectively.

Dynamic Semantics

m-Boolean-operator : *Boolean-operator* × *Value* × *Expression* → *Environment* $\xrightarrow{o}$ $\mathbb{B}$

m-Boolean-operator (*op*, *lvalue*, *right*)ρ $\triangleq$
  cases *op*:
    AND → *m-Boolean-and*(*lvalue*, *right*) ρ, 203
    OR → *m-Boolean-or*(*lvalue*, *right*) ρ 203
  end

If the left operand to a logical conjunction operator has the value *false*, the value of the Boolean infix operation shall be the value *false*, and the right operand shall not be evaluated. If the left operand to a logical conjunction operator has the value *true*, the value of the Boolean infix operation shall be the value of the right operand.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

If the left operand to a logical disjunction operator has the value *true*, the value of the Boolean infix operation shall be the value *true*, and the right operand shall not be evaluated. If the left operand to a logical disjunction operator has the value *false*, the value of the Boolean infix operation shall be the value of the right operand.

m-Boolean-and : *Value* × *Expression* → *Environment* $\xrightarrow{o}$ $\mathbb{B}$

m-Boolean-and (*lvalue*, *right*)ρ $\triangleq$
  if *lvalue*
  then *m-expression*(*right*) ρ 190
  else return false

m-Boolean-or : *Value* × *Expression* → *Environment* $\xrightarrow{o}$ $\mathbb{B}$

m-Boolean-or (*lvalue*, *right*)ρ $\triangleq$
  if *lvalue*
  then return true
  else *m-expression*(*right*) ρ 190

6.8.2.5 Set Infix Operations

For operands of a set type or of a packedset type, there are four set infix operations: set union, set difference, set intersection, and symmetric set difference. These operations are denoted lexically by the symbols '+', '-', '*', and '/' respectively.

NOTE — Set infix operations apply to operands of a set type and to operands of a packedset type.

Abstract Syntax

types

Set-operation :: *op* : *Set-operator*
  otype : *Typed*

Static Semantics

Both operands of a set infix operation shall be of the identical set type, and the result shall also be of this type.

wf-set-operation : *Set-operation* × *Expression-typed* × *Expression-typed* → *Environment* → $\mathbb{B}$

wf-set-operation (*mk-Set-operation* (-, *otype*), *ltype*, *rtype*)ρ $\triangleq$
  is-set-type (*ltype*)ρ ∧ 322
  ltype = *otype* ∧ *rtype* = *otype*

t-set-operation : *Set-operation* → *Expression-typed*

t-set-operation (*mk-Set-operation* (-, *otype*)) $\triangleq$
  otype

Dynamic Semantics

m-set-operation : *Set-operation* × *Value* × *Value* $\xrightarrow{o}$ *Value*

m-set-operation (*mk-Set-operation* (*op*, -), *lvalue*, *rvalue*) $\triangleq$
  let *mk-Set-value* (*lvalues*) = *lvalue* in
  let *mk-Set-value* (*rvalues*) = *rvalue* in
  def *res* = *m-set-operator*(*op*, *lvalues*, *rvalues*) in 204
  return *mk-Set-value* (*res*)

annotations For each set, the actual set of values is contained in a composite object; the sets must be extracted in order to apply the operator, and then the result converted back to the appropriate composite type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Set Infix Operators

Abstract Syntax

types

Set-operator = UNION | DIFFERENCE | INTERSECTION | SYMMETRIC-DIFFERENCE

annotations The above operators correspond to '+', '-', '*', and '/' respectively.

Dynamic Semantics

m-set-operator : *Set-operator* × *Value* × *Value* $\xrightarrow{o}$ *Value*

m-set-operator (*op*, *lvalues*, *rvalues*) $\triangleq$
  cases *op*:
    UNION → *m-set-union*(*lvalues*, *rvalues*) , 204
    DIFFERENCE → *m-set-difference*(*lvalues*, *rvalues*) , 204
    INTERSECTION → *m-set-intersection*(*lvalues*, *rvalues*) , 204
    SYMMETRIC-DIFFERENCE → *m-set-symmetric-difference*(*lvalues*, *rvalues*) 204
  end

The evaluation of the left and right operands of a set infix operation shall result in two values, called the 'left value' and the 'right value'.

The value of a set union operation shall be the set whose elements are elements of either the left value, or of the right value, or of both values.

The value of a set difference operation shall be the set whose elements are elements of the left value, but are not elements of the right value.

The value of a set intersection operation shall be the set whose elements are elements of both the left value and the right value.

The value of a symmetric set difference operation shall be the set whose elements are elements found in only one of the two values, left value and right value.

m-set-union : *Value* × *Value* $\xrightarrow{o}$ *Value*

m-set-union (*lvalues*, *rvalues*) $\triangleq$
  return *lvalues* ∪ *rvalues*

m-set-difference : *Value* × *Value* $\xrightarrow{o}$ *Value*

m-set-difference (*lvalues*, *rvalues*) $\triangleq$
  return *lvalues* − *rvalues*

m-set-intersection : *Value* × *Value* $\xrightarrow{o}$ *Value*

m-set-intersection (*lvalues*, *rvalues*) $\triangleq$
  return *lvalues* ∩ *rvalues*

m-set-symmetric-difference : *Value* × *Value* $\xrightarrow{o}$ *Value*

m-set-symmetric-difference (*lvalues*, *rvalues*) $\triangleq$
  return (*lvalues* − *rvalues*) ∪ (*rvalues* − *lvalues*)

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.8.2.6 String Literal Infix Operations

For operands of the string literal type, there is one string literal infix operation: string catenation. This operation is denoted lexically by the symbol '+'.

CHANGE — String catenation is not described in *Programming in Modula-2*.

NOTE — String catenation is not applicable to variables of a concrete string type, or to constants of the character type. The examples below show both valid and invalid attempts to perform string catenation.

EXAMPLE — String catenation.

```
CONST
  CR = 15C;
  LF = 12C;
  LineEnd = CR + LF;                    (* valid *)
  DoubleLineEnd = LineEnd + LineEnd;    (* valid *)
  Greeting = "hello " + "world";        (* valid, same as "hello world" *)
  Reply = "hello " + "" + "world";      (* valid, same as "hello world" *)
  String1 = "I said " + Greeting;       (* valid, same as "I said hello world" *)
  String2 = Greeting + CR;              (* valid *)

CONST
  Linefeed = CHR(10);
  String3 = "hello" + Linefeed;         (* invalid, Linefeed is of type CHAR *)

VAR
  who: ARRAY [0 .. 79] OF CHAR;

BEGIN
  who := Greeting + CR;                 (* valid *)
  who := "I said " + who;               (* invalid, who is of a concrete string type *)
```

Abstract Syntax

types

String-literal-operation :: *otype* : *String-type*

Static Semantics

Both operands of a string literal infix operation shall be of the string literal type, and the result shall also be of this type.

wf-string-literal-operation : *String-literal-operation* × *Expression-typed* × *Expression-typed* → *Environment* → $\mathbb{B}$

wf-string-literal-operation (*mk-String-literal-operation* (*otype*), *ltype*, *rtype*)ρ $\triangleq$
  is-String-type(*ltype*) ∧
  is-String-type(*rtype*) ∧
  let *mk-String-type* (*llength*) = *ltype* in
  let *mk-String-type* (*rlength*) = *rtype* in
  otype = *mk-String-type* (*llength* + *rlength*)

t-string-literal-operation : *String-literal-operation* → *Expression-typed*

t-string-literal-operation (*mk-String-literal-operation* (*otype*)) $\triangleq$
  otype

Dynamic Semantics

The evaluation of the left and right operands of a string literal infix operation shall result in two values, called the 'left value' and the 'right value'.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The value of a string literal infix operation shall be the string value formed by the catenation of the right value to the end of the left value.

CLARIFICATION — This part of ISO/IEC 10514 has not adopted a change to Modula-2 (described in the fourth edition of *Programming in Modula-2*) of always requiring a string terminator for a string value.

m-string-literal-operation : *String-literal-operation* × *Value* × *Value* $\xrightarrow{o}$ *Value*

m-string-literal-operation (*mk-String-literal-operation* (-), *lvalue*, *rvalue*) $\triangleq$
return *lvalue* $\curvearrowright$ *rvalue*

6.8.2.7 Relational Operations

For operands of an elementary type, there are relational operations that provide tests for equality and inequality. These operations are denoted lexically by the symbols '=' and '<>' respectively. For operands of a scalar type, there are further relational operations that provide tests for ordering. These operations are denoted lexically by the symbols '<', '>', '<=' and '>='. For operands of a set type or a packedset type, there are relational operations that provide set relationships, and an operation that provides a test for set membership. These operations are denoted lexically by the symbols '<=', '>=' and 'IN' respectively.

NOTE — The symbol '#' may be used as a synonym for '<>'.

Abstract Syntax

types

Relational-operation = *Complex-comparison-operation*
| *Scalar-relational-operation*
| *Set-relational-operation*
| *Procedure-relational-operation*
| *Pointer-relational-operation*
| *Protection-relational-operation*
| *System-basic-type-relational-operator*

Static Semantics

wf-relational-operation : *Infix-operation* × *Expression-typed* × *Expression-typed* → *Environment* → $\mathbb{B}$

wf-relational-operation (*op*, *ltype*, *rtype*)ρ $\triangleq$

(*is-Complex-comparison-operation*(*op*)	→ *wf-complex-comparison-operation* (*op*, *ltype*, *rtype*)ρ,	207
is-Scalar-relational-operation(*op*)	→ *wf-scalar-relational-operation* (*op*, *ltype*, *rtype*)ρ,	208
is-Set-relational-operation(*op*)	→ *wf-set-relational-operation* (*op*, *ltype*, *rtype*)ρ,	210
is-Procedure-relational-operation(*op*)	→ *wf-procedure-relational-operation* (*op*, *ltype*, *rtype*)ρ,	213
is-Pointer-relational-operation(*op*)	→ *wf-pointer-relational-operation* (*op*, *ltype*, *rtype*)ρ,	215
is-Protection-relational-operation(*op*)	→ *wf-protection-relational-operation* (*op*, *ltype*, *rtype*)ρ,	216
is-System-basic-type-relational-operation(*op*)	→ *wf-system-basic-type-relational-operation* (*op*, *ltype*, *rtype*)ρ)	217

t-relational-operation : *Infix-operation* → *Expression-typed*

t-relational-operation (*expr*) $\triangleq$

(*is-Complex-comparison-operation*(*op*)	→ *t-complex-comparison-operation* (*op*),	207
is-Scalar-relational-operation(*op*)	→ *t-scalar-relational-operation* (*op*),	208
is-Set-relational-operation(*op*)	→ *t-set-relational-operation* (*op*),	211
is-Procedure-relational-operation(*op*)	→ *t-procedure-relational-operation* (*op*),	213
is-Pointer-relational-operation(*op*)	→ *t-pointer-relational-operation* (*op*),	215
is-Protection-relational-operation(*op*)	→ *t-protection-relational-operation* (*op*),	216
is-System-basic-type-relational-operation(*op*)	→ *t-system-basic-type-relational-operation* (*op*))	218

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Dynamic Semantics

m-relational-operation : *Infix-operation* × *Value* × (*Value* | *Expression*) $\xrightarrow{o}$ *Value*

m-relational-operation (*op*, *lval*, *rval*) $\triangleq$
(*is-Complex-comparison-operation*(*op*) → *m-complex-comparison-operation*(*op*, *ltype*, *rtype*) , 207
is-Scalar-relational-operation(*op*) → *m-scalar-relational-operation*(*op*, *ltype*, *rtype*) , 209
is-Set-relational-operation(*op*) → *m-set-relational-operation*(*op*, *ltype*, *rtype*) , 211
is-Procedure-relational-operation(*op*) → *m-procedure-relational-operation*(*op*, *ltype*, *rtype*) , 213
is-Pointer-relational-operation(*op*) → *m-pointer-relational-operation*(*op*, *ltype*, *rtype*) , 215
is-Protection-relational-operation(*op*) → *m-protection-relational-operation*(*op*, *ltype*, *rtype*) , 216
is-System-basic-type-relational-operation(*op*) → *m-system-basic-type-relational-operation*(*op*, *ltype*, *rtype*)) 218

6.8.2.7.1 Complex Comparison Operations

Two complex comparison operators are provided for operands of a complex number type. These operators are denoted lexically by the symbols '=' and '<>' (or '#').

Abstract Syntax

types

Complex-comparison-operation :: *op* : *Complex-comparison-operator*

Static Semantics

Both operands of a complex comparison operation shall be of types that are complex number types, and these types shall be expression-compatible. The result of a complex comparison operation shall be of the Boolean type.

wf-complex-comparison-operation :
Complex-comparison-operation × *Expression-typed* × *Expression-typed* → *Environment* → $\mathbb{B}$

wf-complex-comparison-operation (*mk-Complex-comparison-operation* (*op*), *ltype*, *rtype*)ρ $\triangleq$
is-Complex-number-type(*ltype*) ∧
is-Complex-number-type(*rtype*) ∧
is-expression-compatible (*ltype*, *rtype*)ρ 121

t-complex-comparison-operation : *Complex-comparison-operation* → *Expression-typed*

t-complex-comparison-operation (*mk-Complex-comparison-operation* (-)) $\triangleq$
BOOLEAN-TYPE

Dynamic Semantics

m-complex-comparison-operation : *Complex-comparison-operation* × *Value* × *Value* $\xrightarrow{o}$ *Value*

m-complex-comparison-operation (*mk-Complex-comparison-operation* (*op*), *lvalue*, *rvalue*) $\triangleq$
m-complex-comparison-operator(*op*, *lvalue*, *rvalue*) 208

Complex Comparison Operators

Abstract Syntax

types

Complex-comparison-operator = EQ | NE

annotations The above operators correspond to '=' and '<>' (or '#') respectively.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Dynamic Semantics

m-complex-comparison-operator : *Complex-comparison-operator* × *Value* × *Value* $\xrightarrow{o}$ *Value*

m-complex-comparison-operator (*op*, *lvalue*, *rvalue*) $\triangleq$
  cases *op*:
    EQ → *m-complex-eq*(*lvalue*, *rvalue*) , 208
    NE → *m-complex-ne*(*lvalue*, *rvalue*) 208
  end

The evaluation of the left and right operands of a complex comparison operation shall result in two values.

The value of a complex equality operation shall be the value *true* if and only if the two values are the same.

The value of a complex inequality operation shall be the value *true* if and only if the two values are different.

NOTE — A consequence of the approximation inherent in complex number types is that the result of a complex comparison operation will depend upon the nature of the approximation.

m-complex-eq : *Value* × *Value* $\xrightarrow{o}$ *Value*

m-complex-eq (*lvalue*, *rvalue*) $\triangleq$
  return *lvalue* = *rvalue*

m-complex-ne : *Value* × *Value* $\xrightarrow{o}$ *Value*

m-complex-ne (*lvalue*, *rvalue*) $\triangleq$
  return *lvalue* ≠ *rvalue*

6.8.2.7.2 Scalar Relational Operations

Six scalar relational operators are provided for operands of a scalar type. These operators are denoted lexically by the symbols '=', '<>' (or '#'), '<', '>', '<=', and '>='.

Abstract Syntax

types

Scalar-relational-operation :: *op* : *Scalar-relational-operator*

Static Semantics

Both operands of a scalar relational operation shall be of types that are scalar types, and these types shall be expression-compatible. The result of a scalar relational operation shall be of the Boolean type.

NOTE 1 — The Boolean type is an ordinal type with the value *false* being less than the value *true*. So, if p and q are operands of the Boolean type, 'p = q' denotes their equivalence and 'p <= q' means 'p implies q'.

wf-scalar-relational-operation :
  Scalar-relational-operation × *Expression-typed* × *Expression-typed* → *Environment* → $\mathbb{B}$

wf-scalar-relational-operation (*mk-Scalar-relational-operation* (*op*), *ltype*, *rtype*)ρ $\triangleq$
  is-scalar-type (*ltype*)ρ ∧ 322
  is-scalar-type (*rtype*)ρ ∧ 322
  is-expression-compatible (*ltype*, *rtype*)ρ 121

t-scalar-relational-operation : *Scalar-relational-operation* → *Expression-typed*

t-scalar-relational-operation (*mk-Scalar-relational-operation* (-)) $\triangleq$
  BOOLEAN-TYPE

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Dynamic Semantics

m-scalar-relational-operation : *Scalar-relational-operation* × *Value* × *Value* $\xrightarrow{o}$ *Value*

m-scalar-relational-operation (*mk-Scalar-relational-operation* (*op*), *lvalue*, *rvalue*) $\triangleq$
  m-scalar-relational-operator(*op*, *lvalue*, *rvalue*) 209

Scalar Relational Operators

Abstract Syntax

types

Scalar-relational-operator = EQ | NE | LT | GT | LE | GE

annotations The above operators correspond to '=', '<>' (or '#'), '<', '>', '<=' and '>=' respectively.

Dynamic Semantics

m-scalar-relational-operator : *Scalar-relational-operator* × *Value* × *Value* $\xrightarrow{o}$ *Value*

m-scalar-relational-operator (*op*, *lvalue*, *rvalue*) $\triangleq$
  cases *op*:
    EQ → *m-scalar-eq*(*lvalue*, *rvalue*), 209
    NE → *m-scalar-ne*(*lvalue*, *rvalue*), 209
    LT → *m-scalar-lt*(*lvalue*, *rvalue*), 210
    GT → *m-scalar-gt*(*lvalue*, *rvalue*), 210
    LE → *m-scalar-le*(*lvalue*, *rvalue*), 210
    GE → *m-scalar-ge*(*lvalue*, *rvalue*) 210
  end

The evaluation of the left and right operands of a scalar relational operation shall result in two values, called the 'left value' and the 'right value'.

The value of a scalar equality operation shall be the value *true* if and only if the two values are the same.

The value of a scalar inequality operation shall be the value *true* if and only if the two values are different.

The value of a scalar less than operation shall be the value *true* if and only if the left value is less than the right value.

The value of a scalar greater than operation shall be the value *true* if and only if the left value is greater than the right value.

The value of a scalar less than or equal operation shall be the value *true* if and only if the left value is less than or equal to the right value.

The value of a scalar greater than or equal operation shall be the value *true* if and only if the left value is greater than or equal to the right value.

NOTE 2 — A consequence of the approximation inherent in real number types is that, for operands of these types, the result of a scalar relational operation will depend upon the nature of the approximation.

m-scalar-eq : *Value* × *Value* $\xrightarrow{o}$ *Value*

m-scalar-eq (*lvalue*, *rvalue*) $\triangleq$
  return *scalar-value* (*lvalue*) = *scalar-value* (*rvalue*) 210 210

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

m-scalar-ne : *Value* × *Value* $\xrightarrow{o}$ *Value*

m-scalar-ne (*lvalue*, *rvalue*) $\triangleq$
  return *scalar-value* (*lvalue*) $\neq$ *scalar-value* (*rvalue*) 210 210

m-scalar-lt : *Value* × *Value* $\xrightarrow{o}$ *Value*

m-scalar-lt (*lvalue*, *rvalue*) $\triangleq$
  return *scalar-value* (*lvalue*) $<$ *scalar-value* (*rvalue*) 210 210

m-scalar-gt : *Value* × *Value* $\xrightarrow{o}$ *Value*

m-scalar-gt (*lvalue*, *rvalue*) $\triangleq$
  return *scalar-value* (*lvalue*) $>$ *scalar-value* (*rvalue*) 210 210

m-scalar-le : *Value* × *Value* $\xrightarrow{o}$ *Value*

m-scalar-le (*lvalue*, *rvalue*) $\triangleq$
  return *scalar-value* (*lvalue*) $\leq$ *scalar-value* (*rvalue*) 210 210

m-scalar-ge : *Value* × *Value* $\xrightarrow{o}$ *Value*

m-scalar-ge (*lvalue*, *rvalue*) $\triangleq$
  return *scalar-value* (*lvalue*) $\geq$ *scalar-value* (*rvalue*) 210 210

Auxiliaries

scalar-value : *Value* → *Number*

scalar-value (*value*) $\triangleq$
  (*is-Enumeration-value*(*value*) → let *mk-Enumeration-value* (*id*, *ids*) = *value* in
    firstocc (*id*, *ids*) − 1, 615
   value : *Char* → *collation-map* (*value*),
   value : *String-value* → *collation-map* (*to-char* (*value*)), 253
   value : $\mathbb{B}$ → if *value* then 1 else 0,
   others → *value*)

6.8.2.7.3 Set Relational Operations

Five set relational operators are provided. These operators are denoted lexically by the symbols '=', '<>' (or '#'), '<=', '>=', and 'IN'.

Abstract Syntax

types

Set-relational-operation :: *op* : *Set-relational-operator*

Static Semantics

Both operands of a set equality, set inequality, subset, or superset operation shall be of types that are set types, and these types shall be identical.

The right operand of a set membership operation shall be of a set type. The type of the left operand and the base type of the set type shall be assignment-compatible.

The result of a set relational operation shall be of the Boolean type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wf-set-relational-operation : *Set-relational-operation* × *Expression-typed* × *Expression-typed* → *Environment* → 𝔹

wf-set-relational-operation (*mk-Set-relational-operation* (*op*), *ltype*, *rtype*)ρ ≜
  (*op* ∈ {EQ, NE, SUBSET, SUPERSET} → *is-set-type* (*ltype*)ρ ∧ 322
    ltype = *rtype*,
  op = IN → *wf-membership-expression* (*ltype*, *rtype*)ρ) 211

t-set-relational-operation : *Set-relational-operation* → *Expression-typed*

t-set-relational-operation (*mk-Set-relational-operation* (-)) ≜
  BOOLEAN-TYPE

Dynamic Semantics

m-set-relational-operation : *Set-relational-operation* × *Value* × *Value* $\xrightarrow{o}$ *Value*

m-set-relational-operation (*mk-Set-relational-operation* (*op*), *lvalue*, *rvalue*) ≜
  (*op* ∈ {EQ, NE, SUBSET, SUPERSET} → let *mk-Set-value* (*lvalues*) = *lvalue* in
    let *mk-Set-value* (*rvalues*) = *rvalue* in
    def *res* = *m-set-relational-operator*(*op*, *lvalues*, *rvalues*) in 211
    return *res* ,
  op = IN → let *mk-Set-value* (*rvalues*) = *rvalue* in
    def *res* = *m-set-relational-operator*(*op*, *lvalue*, *rvalues*) in 211
    return *res*)

annotations The actual set of values is contained in a composite object, so the sets must be extracted in order to apply the operator.

Auxiliaries

wf-membership-expression : *Expression-typed* × *Expression-typed* → *Environment* → 𝔹

wf-membership-expression (*ltype*, *rtype*)ρ ≜
  is-ordinal-type (*ltype*)ρ ∧ 322
  is-set-type (*rtype*)ρ ∧ 322
  let *htype* = *host-type-of* (*base-type-of* (*rtype*)ρ)ρ in 325 325
  is-assignment-compatible (*ltype*, *rtype*)ρ 123

Set Relational Operators

Abstract Syntax

types

Set-relational-operator = EQ | NE | SUBSET | SUPERSET | IN

annotations The above operators correspond to '=', '<>' (or '#'), '<=', '>=' and 'IN' respectively.

Dynamic Semantics

m-set-relational-operator : *Set-relational-operator* × *Ordinal-value*-set × *Ordinal-value*-set $\xrightarrow{o}$ *Value*

m-set-relational-operator (*op*, *lvalue*, *rvalue*) ≜
  cases *op*:
    EQ → *m-set-eq*(*lvalue*, *rvalue*) , 212
    NE → *m-set-ne*(*lvalue*, *rvalue*) , 212
    SUBSET → *m-subset*(*lvalue*, *rvalue*) , 212
    SUPERSET → *m-superset*(*lvalue*, *rvalue*) , 212
    IN → *m-in*(*lvalue*, *rvalue*) 212
  end

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The evaluation of the left and right operands of a set relational operation shall result in two values, called the 'left value' and the 'right value'.

The value of a set equality operation shall be the value *true* if and only if the two values have the same elements.

The value of a set inequality operation shall be the value *true* if and only if the two values do not have the same elements.

The value of a subset operation shall be the value *true* if and only if all the elements of the left value are also in the right value.

The value of a superset operation shall be the value *true* if and only if all the elements of the right value are also in the left value.

The value of a set membership operation shall be the value *true* if and only if the left value is an element of the right value.

NOTE — A consequence of the above rules is that '`1000 IN DigitSet{0..9}`' produces a result of *false*, and no exception is raised.

EXAMPLE — Set complement.

Given the declarations:

```
TYPE
  BaseType = ...
  SetType = SET OF BaseType;
VAR
  s, complement : SetType;
```

the complement of the set `s` can be calculated by:

```
complement := SetType{MIN(BaseType)..MAX(BaseType)} - s;
```

m-set-eq : *Ordinal-value*-set × *Ordinal-value*-set $\xrightarrow{o}$ *Value*

m-set-eq (*lvalues*, *rvalues*) $\triangleq$
  return *rvalues* = *lvalues*

m-set-ne : *Ordinal-value*-set × *Ordinal-value*-set $\xrightarrow{o}$ *Value*

m-set-ne (*lvalues*, *rvalues*) $\triangleq$
  return *rvalues* $\neq$ *lvalues*

m-subset : *Ordinal-value*-set × *Ordinal-value*-set $\xrightarrow{o}$ *Value*

m-subset (*lvalues*, *rvalues*) $\triangleq$
  return *lvalues* $\subseteq$ *rvalues*

m-superset : *Ordinal-value*-set × *Ordinal-value*-set $\xrightarrow{o}$ *Value*

m-superset (*lvalues*, *rvalues*) $\triangleq$
  return *rvalues* $\subseteq$ *lvalues*

m-in : *Ordinal-value*-set × *Ordinal-value*-set $\xrightarrow{o}$ *Value*

m-in (*lvalue*,) $\triangleq$
  return *lvalue* $\in$ *rvalues*

6.8.2.7.4 Procedure Relational Operations

Two procedure relational operators are provided for operands of a procedure type or a procedure structure. These operators are denoted lexically by the symbols '=' and '<>' (or '#').

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Abstract Syntax

types

Procedure-relational-operation :: *op* : *Procedure-relational-operator*

Static Semantics

If both operands of a procedure relational operation are variables of a procedure type, those variables shall be of identical type.

If at least one of the operands of a procedure relational operation is a declared procedure, or a procedure constant, the operands shall be of identical structure.

NOTES

1 The structure of a procedure, procedure constant, or procedure type is given by the number, formal types, and binding of the formal parameters, and by the function return type (in the case of a function procedure, function procedure constant, or function procedure type).

2 Standard procedures and system procedures may not be operands in a procedure relational operation.

EXAMPLE — Procedure relational operations.

```
PROCEDURE Relations(fp: INTEGER);
  PROCEDURE A(fp: CARDINAL); BEGIN END A;
  PROCEDURE B(fp: CARDINAL); BEGIN END B;
  PROCEDURE C(fp: INTEGER); BEGIN END C;
  TYPE TC1 = PROCEDURE(CARDINAL); TC2 = PROCEDURE(CARDINAL);
  CONST AA = A; untrue = (A = B);   (*     valid *)
  VAR VC1a, VC1b: TC1; VC2: TC2;
BEGIN
  VC1a := A; VC1b := AA; VC2 := B;
  IF VC1a = VC1b    THEN END ;      (*   valid *)
  IF VC1a = VC2     THEN END ;      (* invalid - different types *)
  IF VC1a = B       THEN END ;      (*   valid *)
  IF VC1a = C       THEN END ;      (* invalid - different structure *)
  IF Relations = C  THEN END ;      (*   valid *)
END Relations;
```

The result of a procedure relational operation shall be of the Boolean type.

wf-procedure-relational-operation :
Procedure-relational-operation × *Expression-typed* × *Expression-typed* → *Environment* → $\mathbb{B}$

wf-procedure-relational-operation (*mk-Procedure-relational-operation* (-), *ltype*, *rtype*)ρ $\triangleq$
(*is-Type-name*(*ltype*) ∧ *is-Type-name*(*rtype*) → *ltype* = *rtype* ∧
is-procedure-type (*ltype*)ρ, 321
is-Type-name(*ltype*) ∧ *is-Procedure-typed*(*rtype*) → *structure-of* (*ltype*)ρ = *procedure-structure* (*rtype*), 315 316
is-Procedure-typed(*ltype*) ∧ *is-Type-name*(*rtype*) → *procedure-structure* (*ltype*) = *structure-of* (*rtype*)ρ, 316 315
is-Procedure-typed(*ltype*) ∧ *is-Procedure-type*(*rtype*) → *procedure-structure* (*ltype*) = *procedure-structure* (*rtype*)) 316

t-procedure-relational-operation : *Procedure-relational-operation* → *Expression-typed*

t-procedure-relational-operation (*mk-Procedure-relational-operation* (-)) $\triangleq$
BOOLEAN-TYPE

Dynamic Semantics

m-procedure-relational-operation : *Procedure-relational-operation* × *Value* × *Value* $\xrightarrow{o}$ *Value*

m-procedure-relational-operation (*mk-Procedure-relational-operation* (*op*), *lvalue*, *rvalue*) $\triangleq$
m-procedure-relational-operator(*op*, *lden*, *rden*) 214

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Procedure Relational Operators

Abstract Syntax

types

Procedure-relational-operator = EQ | NE

annotations The above operators correspond to '=' and '<>' (or '#') respectively.

CLARIFICATION — *Programming in Modula-2* does not state whether the equality and inequality operators are applicable to values that are procedures.

Dynamic Semantics

m-procedure-relational-operator : *Procedure-relational-operator* × *Value* × *Value* $\xrightarrow{o}$ *Value*

m-procedure-relational-operator (*op*, *lvalue*, *rvalue*) $\triangleq$
  cases *op*:
    EQ → *m-proc-eq*(*lvalue*, *rvalue*) , 214
    NE → *m-proc-ne*(*lvalue*, *rvalue*) 214
  end

The value of a procedure equality operation shall be the value *true* if and only if the values of both operands denote the same procedure text.

The value of a procedure inequality operation shall be the value *true* if and only if the values of both operands do not denote the same procedure text.

m-proc-eq : *Value* × *Value* $\xrightarrow{o}$ *Value*

m-proc-eq (*lvalue*, *rvalue*) $\triangleq$
  return *lvalue* = *rvalue*

m-proc-ne : *Value* × *Value* $\xrightarrow{o}$ *Value*

m-proc-ne (*lvalue*, *rvalue*) $\triangleq$
  return *lvalue* ≠ *rvalue*

6.8.2.7.5 Pointer Relational Operations

Two pointer relational operators are provided for operands of a pointer type or an opaque type. These operators are denoted lexically by the symbols '=' and '<>' (or '#').

Abstract Syntax

types

Pointer-relational-operation :: *op* : *Pointer-relational-operator*

Static Semantics

One of the following shall old for the operands of a pointer relational operation:

a) both operands are of identical opaque type,

b) both operands are of identical pointer type,

c) one operand is of a pointer type and the value of the other is the nil value.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

d) both operands have the nil value.

NOTE 1 — The nil value is that denoted by the pervasive identifier `NIL` — see 6.10.2.8.

The result of a pointer relational operation shall be of the Boolean type.

CLARIFICATION — This part of ISO/IEC 10514 agrees with the third edition of *Programming in Modula-2*, which allows the equality and inequality operators to be applied to values of an opaque type.

wf-pointer-relational-operation :
Pointer-relational-operation × *Expression-typed* × *Expression-typed* → *Environment* → 𝔹

wf-pointer-relational-operation (*mk-Pointer-relational-operation* (-), *ltype*, *rtype*)ρ ≙
(*is-Type-name*(*ltype*) ∧ *is-Type-name*(*rtype*) → *ltype* = *rtype* ∧
(*is-pointer-type* (*ltype*)ρ ∨ *is-opaque-type* (*ltype*)ρ), 322 322
is-Type-name(*ltype*) ∧ *rtype* = NIL-TYPE → *is-pointer-type* (*ltype*)ρ, 322
ltype = NIL-TYPE ∧ *is-Type-name*(*rtype*) → *is-pointer-type* (*rtype*)ρ, 322
ltype = NIL-TYPE ∧ *rtype* = NIL-TYPE → true)

t-pointer-relational-operation : *Pointer-relational-operation* → *Expression-typed*

t-pointer-relational-operation (*mk-Pointer-relational-operation* (-)) ≙
BOOLEAN-TYPE

annotations Opaque types are indistinguishable from pointer types with respect to the meaning function. The well-formed conditions forbid the dereferencing of a value of an opaque type outside of the implementation module in which the opaque type is declared — see 6.7.4.

Dynamic Semantics

m-pointer-relational-operation : *Pointer-relational-operation* × *Value* × *Value* $\xrightarrow{o}$ *Value*

m-pointer-relational-operation (*mk-Pointer-relational-operation* (*op*), *lvalue*, *rvalue*) ≙
let *mk-Pointer-value* (*plvalue*) = *lvalue* in
let *mk-Pointer-value* (*prvalue*) = *rvalue* in
m-pointer-relational-operator(*op*, *plvalue*, *prvalue*) 215

Pointer Relational Operators

Abstract Syntax

types

Pointer-relational-operator = EQ | NE

annotations The above operators correspond to '=' and '<>' (or '#') respectively.

Dynamic Semantics

m-pointer-relational-operator : *Pointer-relational-operator* × *Value* × *Value* $\xrightarrow{o}$ 𝔹

m-pointer-relational-operator (*op*, *lvalue*, *rvalue*) ≙
cases *op*:
EQ → *m-pointer-eq*(*lvalue*, *rvalue*) , 216
NE → *m-pointer-ne*(*lvalue*, *rvalue*) 216
end

The value of a pointer equality operation shall be the value *true* if and only if the values of both operands denote the same variable, or if they both have the nil value.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The value of a pointer inequality operation shall be the value *true* if and only if the values of both operands do not denote the same variable, and if they do not both have the nil value.

NOTE 2 — Since pointer values can be constructed via **SYSTEM.MAKEADR**, these semantics are defined in terms of the underlying model rather than the objects held in the store.

m-pointer-eq : *Value* × *Value* $\xrightarrow{o}$ *Value*

m-pointer-eq (*plvalue*, *prvalue*) $\triangleq$
return *plvalue* = *prvalue*

m-pointer-ne : *Value* × *Value* $\xrightarrow{o}$ *Value*

m-pointer-ne (*plvalue*, *prvalue*) $\triangleq$
return *plvalue* $\neq$ *prvalue*

6.8.2.7.6 Protection Relational Operations

Two protection relational operators are provided for operands of the implementation-defined protection type. These operators are denoted lexically by the symbols '<=' and '>='. The protection relat onal operators are those that are common both to ordinal types and to set types — see 6.10.2.11.

Abstract Syntax

types

Protection-relational-operation :: *op* : *Protection-relational-operator*

Static Semantics

Both operands of a protection relational operation shall be of the protection type.

The result of a protection relational operation shall be of the Boolean type.

wf-protection-relational-operation :
Protection-relational-operation × *Expression-typed* × *Expression-typed*) → *Environment* → $\mathbb{B}$

wf-protection-relational-operation (*mk-Protection-relational-operation* (-), *ltype*, *rtype*)ρ $\triangleq$
ltype = PROTECTION-TYPE $\wedge$ *rtype* = PROTECTION-TYPE

t-protection-relational-operation : *Protection-relational-operation* → *Expression-typed*

t-protection-relational-operation (*mk-Protection-relational-operation* (-)) $\triangleq$
BOOLEAN-TYPE

Dynamic Semantics

m-protection-relational-operation : *Protection-relational-operation* × *Value* × *Value* $\xrightarrow{o}$ *Value*

m-protection-relational-operation (*mk-Protection-relational-operation* (*op*), *lvalue*, *rvalue*) $\triangleq$
m-protection-relational-operator(*op*, *lvalue*, *rvalue*)

217

Protection Relational Operators

Abstract Syntax

types

Protection-relational-operator = LE | GE

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations The above operators correspond to '<=' and '>=' respectively.

Dynamic Semantics

m-protection-relational-operator : *Protection-relational-operator* × *Value* × *Value* $\xrightarrow{o}$ *Value*

m-protection-relational-operator (*op*, *lvalue*, *rvalue*) $\triangleq$
  cases *op*:
    LE → *m-prot-le*(*lvalue*, *rvalue*) , 217
    GE → *m-prot-ge*(*lvalue*, *rvalue*) 217
  end

The value of a protection less than or equal operation, or a protection greater than or equal operation, shall depend on the implementation-defined order of the values of the protection type. The ordering shall be such that a value P1 of the protection type shall compare less than or equal to another value P2 if and only if P1 does not prevent any of the interrupts not prevented by P2.

m-prot-le : *Value* × *Value* $\xrightarrow{o}$ *Value*

m-prot-le (*lvalue*, *rvalue*) $\triangleq$
  is-protection-less-than-or-equal(*lvalue*, *rvalue*) 217

m-prot-ge : *Value* × *Value* $\xrightarrow{o}$ *Value*

m-prot-ge (*lvalue*, *rvalue*) $\triangleq$
  is-protection-less-than-or-equal(*rvalue*, *lvalue*) 217

is-protection-less-than-or-equal : *Value* × *Value* $\xrightarrow{o}$ $\mathbb{B}$

is-protection-less-than-or-equal (*lvalue*, *rvalue*) $\triangleq$
  return implementation-defined

6.8.2.7.7 System Basic Types Relational Operations

Two relational operators are provided for operands of a system basic type, other than the location type. These operators are denoted lexically by the symbols '=' and '<>' (or '#').

The system basic type whose values may be compared for equality and inequality is the coroutine type — see 7.2. The interrupt source type, which is also defined by the system module COROUTINES, is an implementation-defined type that may be identical to a basic type or may be a predefined constructed type; corresponding operations will therefore be applicable to that type.

The only operation directly defined on the location type is assignment — see 7.1. Values of the location type may only be compared if they are first interpreted as a value of another type using SYSTEM.CAST.

Abstract Syntax

types

S stem-basic-type-relational-o eration :: *op* : *System-basic-type-relational-operator*

Static Semantics

The operands of a system basic type relational operation shall be of identical type.

The result of a system basic type relational operation shall be of the Boolean type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wf-system-basic-type-relational-operation :
system-basic-type-relational-operation → (*Expression-typed* × *Expression-typed*) → *Environment* → 𝔹
wf-system-basic-type-relational-operation (*mk-System-basic-type-relational-operation* (-))(*ltype*, *rtype*)ρ ≜
ltype = *rtype* ∧
ltype = COROUTINE-TYPE

t-system-basic-type-relational-operation : *System-basic-type-relational-operation* → *Expression-typed*
t-system-basic-type-relational-operation (*mk-System-basic-type-relational-operation* (-)) ≜
BOOLEAN-TYPE

Dynamic Semantics

m-system-basic-type-relational-operation : *System-basic-type-relational-operation* → (*Value* × *Value*) $\xrightarrow{o}$ *Value*
m-system-basic-type-relational-operation (*mk-System-basic-type-relational-operation* (*op*))(*lvalue*, *rvalue*) ≜
let *mk-System-basic-type-value* (*slvalue*) = *lvalue* in
let *mk-System-basic-type-value* (*srvalue*) = *rvalue* in
m-system-basic-type-relational-operator(*op*) (*slvalue*, *srvalue*) 218

System Basic Types Relational Operators

Abstract Syntax

types

System-basic-type-relational-operator = EQ | NE

annotations The above operators correspond to '=' and '<>' (or '#') respectively.

Dynamic Semantics

m-system-basic-type-relational-operator : *System-basic-type-relational-operator* → (*Value* × *Value*) $\xrightarrow{o}$ 𝔹
m-system-basic-type-relational-operator (*op*)(*lvalue*, *rvalue*) ≜
cases *op*:
EQ → *m-system-basic-type-eq*(*lvalue*, *rvalue*) , 218
NE → *m-system-basic-type-ne*(*lvalue*, *rvalue*) 218
end

The value of a system basic type equality operation shall be the value *true* if and only if the values of both operands are the same, and hence identify the same coroutine.

The value of a system basic type inequality operation shall be the value *true* if and only if the values of both operands are not the same, and hence do not identify the same coroutine.

m-system-basic-type-eq : *Value* × *Value* $\xrightarrow{o}$ *Value*
m-system-basic-type-eq (*slvalue*, *srvalue*) ≜
return *slvalue* = *srvalue*

m-system-basic-type-ne : *Value* × *Value* $\xrightarrow{o}$ *Value*
m-system-basic-type-ne (*slvalue*, *srvalue*) ≜
return *slvalue* ≠ *srvalue*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.8.3 Prefix Expressions

A prefix expression is an expression whose value is obtained by applying a prefix operator to the value of a single operand.

Concrete Syntax

NOTE — The syntax for the various forms of prefix operators is given in the Lexis — see section 5.

Abstract Syntax

types

Prefix-expression :: *op* : *Prefix-operation*
expr : *Expression*

Static Semantics

The type of a prefix expression shall be the same as the type of its expression.

wf-prefix-expression : *Prefix-expression* → *Environment* → $\mathbb{B}$

wf-prefix-expression (*mk-Prefix-expression* (*op*, *expr*))ρ $\triangleq$
wf-expression (*expr*)ρ ∧ 190
let *etype* = *t-expression* (*expr*)ρ in 190
wf-prefix-operation (*op*, *etype*) 219

t-prefix-expression : *Prefix-expression* → *Expression-typed*

t-prefix-expression (*mk-Prefix-expression* (*op*, -)) $\triangleq$
t-prefix-operation (*op*) 220

Dynamic Semantics

The expression shall be evaluated first, and then the prefix operator shall be applied.

m-prefix-expression : *Prefix-expression* → *Environment* $\xrightarrow{o}$ *Value*

m-prefix-expression (*mk-Prefix-expression* (*op*, *expr*))ρ $\triangleq$
def *value* = *m-expression*(*expr*) ρ in 190
let *handler* = *current-exception-handler* (ρ) in 318
m-prefix-operation(*op*, *value*) *handler* 220

Prefix Operations

Abstract Syntax

types

Prefix-operation = *Arithmetic-prefix-operation* | *Boolean-prefix-operation*

annotations For convenience of the description, prefix operations are divided into two classes.

Static Semantics

wf-prefix-operation : *Prefix-operation* × *Expression-typed* → $\mathbb{B}$

wf-prefix-operation (*op*, *etype*) $\triangleq$
(*is-Arithmetic-prefix-operation*(*op*) → *wf-arithmetic-prefix-operation* (*op*, *etype*), 220
is-Boolean-prefix-operation(*op*) → *wf-Boolean-prefix-operation* (*op*, *etype*)) 222

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

t-prefix-operation : *Prefix-operation* → *Expression-typed*

t-prefix-operation (*op*) ≙
  (*is-Arithmetic-prefix-operation*(*op*) → *t-arithmetic-prefix-operation* (*op*), 220
   is-Boolean-prefix-operation(*op*) → *t-Boolean-prefix-operation* (*op*)) 222

Dynamic Semantics

m-prefix-operation : *Prefix-operation* × *Value* → *Exception-handler* $\xrightarrow{o}$ *Value*

m-prefix-operation (*op*, *value*)*handler* ≙
  (*is-Arithmetic-prefix-operation*(*op*) → *m-arithmetic-prefix-operation*(*op*, *value*) handler, 221
   is-Boolean-prefix-operation(*op*) → *m-Boolean-prefix-operation*(*op*, *value*)) 222

6.8.3.1 Arithmetic Prefix Operations

There are two arithmetic prefix operations: identity and arithmetic negation. These operations are denoted lexically by the symbols '+' and '-' respectively. If the operand of an arithmetic prefix operation is of a complex number type or a real number type, the result of the operation is an approximation to the result of the corresponding mathematical operation. If the operand is of a whole number type, the result of the operation is mathematically correct if it is within the range of values defined by the whole number type.

Abstract Syntax

types

Arithmetic-prefix-operation :: *op* : *Arithmetic-prefix-operator*
  otype : *Number-type*

Static Semantics

The operand of an arithmetic prefix operation shall be of a complex number type, a real number type or a whole number type, except that the arithmetic negation operator shall not be applied to an operand of an unsigned type.

NOTE 1 — The semantics disallow the use of the arithmetic negation operator in an expression such as `(-a+b)` when the type of the expression is the unsigned type.

wf-arithmetic-prefix-operation : *Arithmetic-prefix-operation* × *Expression-typed* → $\mathbb{B}$

wf-arithmetic-prefix-operation (*mk-Arithmetic-prefix-operation* (*op*, *otype*), *etype*) ≙
  otype = *arithmetic-prefix-operation-result-type* (*op*, *etype*) 221

t-arithmetic-prefix-operation : *Arithmetic-prefix-operation* → *Expression-typed*

t-arithmetic-prefix-operation (*mk-Arithmetic-prefix-operation* (-, *otype*)) ≙
  otype

Dynamic Semantics

The value of an arithmetic prefix operation shall be the mathematically correct result of the operation if the result is within the range of values defined by the type of the operand.

An exception shall be raised if the result is not within the range of values defined by the type of the operand.

NOTE 2 — An exception may be raised if the arithmetic negation operator is applied to an operand of a numeric type that has a non-symmetric range.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

m-arithmetic-prefix-operation : *Arithmetic-prefix-operation* × *Value* → *Exception-handler* $\xrightarrow{o}$ *Value*

m-arithmetic-prefix-operation (*mk-Arithmetic-prefix-operation* (*op*, *otype*), *value*)*handler* $\triangleq$
  def *val* = *m-arithmetic-prefix-operator*(*op*, *value*) in
  (*is-Whole-number-type*(*otype*) → *get-whole-result*(*otype*, *val*) *handler*, 199
   is-Real-number-type(*otype*) → *get-real-result*(*otype*, *val*) *handler*, 197
   is-Complex-number-type(*otype*) → *get-complex-result*(*otype*, *val*) *handler*) 194

Auxiliaries

arithmetic-prefix-operation-result-type : *Arithmetic-prefix-operator* × *Typed* → *Typed*

arithmetic-prefix-operation-result-type (*op*, *type*) $\triangleq$
  if *is-Number-type*(*type*)
  then if *type* = UNSIGNED-TYPE ∧ *op* = MINUS
      then undefined
      else *type*
  else undefined

annotations Return the type of an arithmetic prefix operation. The result is undefined if the arithmetic negation operator is applied to a value of an unsigned type or if it is applied to a value of a non-numeric type.

Arithmetic Prefix Operators

Abstract Syntax

types

Arithmetic-prefix-operator = PLUS | MINUS

annotations The above operators correspond to '+' and '-' respectively.

Dynamic Semantics

m-arithmetic-prefix-operator : *Arithmetic-prefix-operator* × *Value* $\xrightarrow{o}$ *Value*

m-arithmetic-prefix-operator (*op*, *value*) $\triangleq$
  cases *op*:
    PLUS → *m-plus*(*value*), 221
    MINUS → *m-minus*(*value*) 221
  end

The value of an identity operation shall be the value of its operand.

The value of an arithmetic negation operation shall be the additive inverse of the value of its operand.

m-plus : *Value* $\xrightarrow{o}$ *Value*

m-plus (*value*) $\triangleq$
  return *value*

m-minus : *Value* $\xrightarrow{o}$ *Value*

m-minus (*value*) $\triangleq$
  return − *value*

6.8.3.2 Boolean Prefix Operations

There is one Boolean prefix operation: logical negation. This operation is denoted lexically by the symbol '**NOT**'.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NOTE — The symbol '~' may be used as a synonym for 'NOT'.

Abstract Syntax

types

Boolean-prefix-operation :: *op* : *Boolean-prefix-operator*

Static Semantics

The operand of a Boolean prefix operation shall be of the Boolean type.

wf-Boolean-prefix-operation : *Boolean-prefix-operation* × *Type* → $\mathbb{B}$

wf-Boolean-prefix-operation (*mk-Boolean-prefix-operation* (-), *etype*) $\triangleq$
  etype = BOOLEAN-TYPE

t-Boolean-prefix-operation : *Boolean-prefix-operation* → *Expression-typed*

t-Boolean-prefix-operation (*mk-Boolean-prefix-operation* (-)) $\triangleq$
  BOOLEAN-TYPE

Dynamic Semantics

m-Boolean-prefix-operation : *Boolean-prefix-operation* × $\mathbb{B} \xrightarrow{o} \mathbb{B}$

m-Boolean-prefix-operation (*mk-Boolean-prefix-operation* (-), *value*) $\triangleq$
  m-not(*value*)

Boolean Prefix Operators

Abstract Syntax

types

Boolean-prefix-operator = NOT

annotations The above operator corresponds to 'NOT' or '~'.

Dynamic Semantics

The value of a Boolean prefix operation shall be the value *true* if its operand has the value *false*, or *false* if its operand has the value *true*.

m-not : *Value* $\xrightarrow{o}$ *Value*

m-not (*value*) $\triangleq$
  return ¬ *value*

6.8.4 Value Designators

A value designator is an expression denoting a constant, a variable (including a formal parameter of a procedure), a component of a constant or variable, or a procedure.

Concrete Syntax

value designator = entire value | indexed value | selected value | dereferenced value ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Abstract Syntax

types

Value-designator = *Entire-value*
| *Indexed-value*
| *Selected-value*
| *Dereferenced-value*

Static Semantics

wf-value-designator : *Value-designator* → *Environment* → $\mathbb{B}$

wf-value-designator $(vd)\rho \triangleq$
cases *vd* :
mk-Entire-value (-) → *wf-entire-value* $(vd)\rho$, 224
mk-Indexed-value (-, -) → *wf-indexed-value* $(vd)\rho$, 226
mk-Selected-value (-, -) → *wf-selected-value* $(vd)\rho$, 227
mk-Dereferenced-value (-, -) → *wf-dereferenced-value* $(vd)\rho$ 228
end

t-value-designator : *Value-designator* → *Environment* → *Typed*

t-value-designator $(vd)\rho \triangleq$
cases *vd* :
mk-Entire-value (-) → *t-entire-value* $(vd)\rho$, 224
mk-Indexed-value (-, -) → *t-indexed-value* $(vd)\rho$, 226
mk-Selected-value (-, -) → *t-selected-value* $(vd)\rho$, 227
mk-Dereferenced-value (-, -) → *t-dereferenced-value* $(vd)\rho$ 228
end

Dynamic Semantics

m-value-designator : *Value-designator* → *Environment* $\xrightarrow{o}$ *Value*

m-value-designator $(vd)\rho \triangleq$
cases *vd*:
mk-Entire-value (-) → *m-entire-value*(vd) ρ, 224
mk-Indexed-value (-, -) → *m-indexed-value*(vd) ρ, 226
mk-Selected-value (-, -) → *m-selected-value*(vd) ρ, 227
mk-Dereferenced-value (-, -) → *m-dereferenced-value*(vd) ρ 228
end

6.8.4.1 Entire Values

An entire value is a value designator that is a qualified identifier; it denotes a constant, a variable (which may be a formal parameter), a field of a record within a with statement that applies to that record, or a declared procedure.

Concrete Syntax

entire value = qualified identifier ;

Abstract Syntax

types

Entire-value :: *qid* : *Qualident*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations Within the body of a with statement, an entire value may also be the field identifier of a record that has been specified in the designator component of the with statement.

Static Semantics

The qualified identifier of an entire value shall denote a constant, a variable (which may be a formal parameter), a field of a record within a with statement that applies to that record, or a procedure.

If the type of the entity denoted by the qualified identifier is a subrange type, the type of the entire value shall be the host type of that subrange type; otherwise it shall be the type of the entity denoted by the qualified identifier.

wf-entire-value : *Entire-value* → *Environment* → 𝔹

wf-entire-value (*mk-Entire-value* (*qid*))ρ ≙
  (*is-constant* (*qid*)ρ ∨ *is-variable-or-field* (*qid*)ρ ∨ *is-procedure* (*qid*)ρ) ∧ 313 315 316
  wf-qualident (*qid*)ρ 53

t-entire-value : *Entire-value* → *Environment* → *Expression-typed*

t-entire-value (*mk-Entire-value* (*qid*))ρ ≙
  let *entity* = *access-environment* (*qid*)ρ in 319
  let *rtype* = (*is-Constant-value*(*entity*) → *type-of-constant* (*qid*)ρ, 314
  is-Variable-typed(*entity*) → *type-of-variable-or-field* (*qid*)ρ, 315
  is-Procedure-value(*entity*) → *type-of-procedure* (*qid*)ρ) in 316
  host-type-of (*rtype*)ρ 325

annotations The type information for a procedure or procedure constant includes its structure and its declaration level.

Dynamic Semantics

The value of an entire value shall be the value of the entity denoted by the qualified identifier.

An exception shall occur (but need not be raised) if the entity denoted by the qualified identifier is a variable of an elementary type whose value is undefined.

NOTE — The value of a variable might be undefined because no value has been assigned to that variable, or because the variable is a component of a variant and either that variant is not active or no value has been assigned to that component since the variant was made active.

m-entire-value : *Entire-value* → *Environment* $\xrightarrow{+}_{o}$ *Value*

m-entire-value (*mk-Entire-value* (*qid*))ρ ≙
  let *entity* = *access-environment* (*qid*)ρ in 319
  let *handler* = *current-exception-handler* (ρ) in 318
  (*is-Constant-value*(*entity*) → return *entity.value*,
  is-Variable(*entity*) → *value-of-a-variable*(*entity*) *handler*, 224
  is-Procedure-value(*entity*) → return *entity.value*)

Auxiliaries

value-of-a-variable : *Variable* → *Exception-handler* $\xrightarrow{o}$ *Value*

value-of-a-variable (*var*)*handler* ≙
  def *val* = *variable-value*(*var*) in 224
  if *val* = UNDEFINED
  then *non-mandatory-detection*(UNDEFINED-VALUE) *handler* 397
  else return *val*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

variable-value : *Variable* $\xrightarrow{o}$ [*Value*]

variable-value (*v*) $\triangleq$
  cases *v*:
    mk-Elementary-variable (*loc*, -) → *contents*(*loc*), 335
    mk-Pointer-variable (*loc*, *type*) → *contents*(*loc*), 335
    mk-Array-variable (*vars*) → dcl *mp* : *Value* $\xrightarrow{m}$ *Value* := {↦};
      (for all *i* ∈ inds *vars*
       do def *ival* = *variable-value*(*vars* (*i*)) in 224
          mp := *mp* † {*i* ↦ *ival*};
       return *mp*),
    mk-Record-variable (*fields*) → dcl *mp* : *Value* $\xrightarrow{m}$ *Value* := {↦};
      (for all *id* ∈ dom *fields*
       do def *ival* = *variable-value*(*fields* (*id*)) in 224
          if *ival* ≠ nil
          then *mp* := *mp* † {*id* ↦ *ival*}
          else skip;
       return *mp*),
    mk-Tag-variable (*var*, -) → *variable-value*(*var*), 224
    mk-Variant-variable (*var*, *tagloc*, *tagvals*) → def *tval* = *variable-value*(*tagloc*) in 224
      if *tval* ∈ *tagvals*
      then *variable-value*(*var*) 224
      else return UNDEFINED ,
    mk-Tagged-variable (*var*, *tagvar*, *tagvals*) → def *tval* = *variable-value*(*tagvar*) in 224
      if *tval* ∈ *tagvals*
      then *variable-value*(*var*) 224
      else return UNDEFINED
  end

6.8.4.2 Indexed Values

An indexed value is an expression that is an array value followed by one or more index expressions, and it denotes a component of the indexed array value, or a component of a component of the indexed array value.

Concrete Syntax

indexed value = array value, left bracket, index expression, { comma, index expression }, right bracket ;

array value = value designator ;

The concrete syntax allows for a full or abbreviated representation of the indexing of an array value that is itself an indexed value. In the abbreviated form, a single comma shall replace the sequence '] [' that occurs in the full form. The abbreviated form and the full form shall be equivalent.

EXAMPLE — Indexing of an indexed value.

A value designator of the form:
`A[E1][E2][E3]`

may also be written in the following ways:
`A[E1,E2,E3]` `A[E1][E2,E3]` `A[E1,E2][E3]`

Abstract Syntax

types

Indexed-value :: *desig* : *Value-designator*
expr : *Expression*

annotations The abstract syntax only uses the full form.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Static Semantics

The array value of an indexed value shall be a value designator that is of an array type. The type of the index expression shall be assignment-compatible with the index type of the array type.

If the component type of the array type is a subrange type, the type of an indexed value shall be the host type of that component type; otherwise it shall be the component type of the array type.

NOTE 1 — An indexed value cannot be formed by indexing a string constant.

wf-indexed-value : *Indexed-value* → *Environment* → $\mathbb{B}$

wf-indexed-value (*mk-Indexed-value* (*desig*, *expr*))ρ $\triangleq$
  wf-value-designator (*desig*)ρ $\wedge$ 223
  wf-expression (*expr*)ρ $\wedge$ 190
  let *type* = *t-value-designator* (*desig*)ρ in 223
  is-array-type (*type*)ρ $\wedge$ 321
  let *itype* = *index-type-of* (*type*)ρ in 326
  is-assignment-compatible (*itype*, *t-expression* (*expr*)ρ)ρ 123 190

t-indexed-value : *Indexed-value* → *Environment* → *Expression-typed*

t-indexed-value (*mk-Indexed-value* (*desig*, -))ρ $\triangleq$
  let *type* = *t-value-designator* (*desig*)ρ in 223
  let *ctype* = *component-type-of* (*type*)ρ in 325
  host-type-of (*ctype*)ρ 325

Dynamic Semantics

The value designator shall be evaluated to give an array value. The index expression shall be evaluated to give an ordinal value. Provided that this ordinal value is a value of the index type of the array value, the value of the indexed value shall be the value of the component of the array value indexed by the ordinal value; otherwise an exception shall be raised.

The order of evaluation of the value designator and the index expression of an indexed value shall be implementation-dependent.

NOTE 2 — If a function call appears in the index expression, the evaluation of the indexed value could produce a transformation of the program state, i.e. have side-effects.

m-indexed-value : *Indexed-value* → *Environment* $\xrightarrow{o}$ *Value*

m-indexed-value (*mk-Indexed-value* (*desig*, *expr*))ρ $\triangleq$
  def *array* = *m-value-designator*(*desig*) ρ in 223
  def *index* = *m-expression*(*expr*) ρ in 190
  if *index* $\in$ dom *array*
  then return *array* (*index*)
  else let *handler* = *current-exception-handler* (ρ) in 318
    mandatory-detection(INDEX-RANGE) *handler* 396

annotations The above evaluation order for the array value and the index expression is one of those permitted; the index expression may instead be evaluated before the array value, or both may be evaluated in parallel — see 4.1.

6.8.4.3 Selected Values

A selected value is an expression that is a record value followed by a field identifier, and it denotes the value of the identified field of the record value.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Concrete Syntax

selected value = record value, period, field identifier ;

record value = value designator ;

Abstract Syntax

types

Selected-value :: *desig* : *Value-designator*
id : *Identifier*

Static Semantics

The record value of a selected value shall be a value designator that is of a record type. The field identifier shall be one of the identifiers or tag identifiers of the field list of that record type.

If the the type of the field denoted by the field identifier is a subrange type, the type of a selected value shall be the host type of the type of the denoted field; otherwise it shall be the type of that field.

wf-selected-value : *Selected-value* → *Environment* → $\mathbb{B}$

wf-selected-value (*mk-Selected-value* (*desig*, *id*))ρ $\triangleq$
wf-value-designator (*desig*)ρ ∧ 223
let *type* = *t-value-designator* (*desig*)ρ in 223
is-record-type (*type*)ρ ∧ 321
id ∈ dom *field-types-of-record* (*type*)ρ

t-selected-value : *Selected-value* → *Environment* → *Expression-typed*

t-selected-value (*mk-Selected-value* (*desig*, *id*))ρ $\triangleq$
let *type* = *t-value-designator* (*desig*)ρ in 223
let *types* = *field-types-of-record* (*type*)ρ in 326
let *ftype* = *types* (*id*) in
host-type-of (*ftype*)ρ 325

Dynamic Semantics

The value designator shall be evaluated to give a record value. Provided that the field identifier denotes a component of a variant of the record type that is a component of the record value, the value of the selected value shall be the value of the component of the record value selected by the field identifier; otherwise an exception shall occur (but need not be raised).

m-selected-value : *Selected-value* → *Environment* $\xrightarrow{o}$ *Value*

m-selected-value (*mk-Selected-value* (*desig*, *id*))ρ $\triangleq$
def *recvar* = *m-value-designator*(*desig*) ρ in 223
if *id* ∈ dom *recvar*
then return *recvar* (*id*)
else let *handler* = *current-exception-handler* (ρ) in 318
non-mandatory-detection(INACTIVE-VARIANT) *handler* 397

6.8.4.4 Dereferenced Values

A dereferenced value is an expression that is a pointer value followed by a dereferencing operator, and it denotes the value of the variable (if any) that is referenced by the pointer value.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Concrete Syntax

dereferenced value = pointer value, dereferencing operator ;

pointer value = value designator ;

Abstract Syntax

types

Dereferenced-value :: *type* : *Typed*
desig : *Value-designator*

Static Semantics

The pointer value of a dereferenced value shall be a value designator that is of a pointer type.

If the the type of the bound type specified in the declaration of the pointer type is a subrange type, the type of a dereferenced value shall be the host type of the bound type; otherwise it shall be the bound type.

wf-dereferenced-value : *Dereferenced-value* → *Environment* → $\mathbb{B}$

wf-dereferenced-value (*mk-Dereferenced-value* (*ptype*, *desig*))ρ $\triangleq$
wf-value-designator (*desig*)ρ ∧ 223
let *type* = *t-value-designator* (*desig*)ρ in 223
is-pointer-type (*type*)ρ ∧ 322
ptype = *type*

t-dereferenced-value : *Dereferenced-value* → *Environment* → *Expression-typed*

t-dereferenced-value (*mk-Dereferenced-value* (*ptype*, *desig*))ρ $\triangleq$
let *btype* = *bound-type-of* (*ptype*)ρ in 326
host-type-of (*btype*)ρ 325

Dynamic Semantics

The value designator shall be evaluated to give a pointer value. Provided that this pointer value denotes a variable, the value of the dereferenced value shall be the value of that variable.

An exception shall be raised if the pointer value is the nil value.

An exception shall occur (but need not be raised) if the pointer value is undefined.

An exception shall occur (but need not be raised) if the value is a pointer to a variable of the wrong type.

NOTE — Since the address type is assignment compatible with all pointer types, the value may be a pointer to a variable of the wrong type if it has been assigned through a variable or parameter of the address type.

m-dereferenced-value : *Dereferenced-value* → *Environment* $\xrightarrow{o}$ *Value*

m-dereferenced-value (*mk-Dereferenced-value* (*ptype*, *desig*))ρ $\triangleq$
def *val* = *m-value-designator*(*desig*) ρ in 223
let *handler* = *current-exception-handler* (ρ) in 318
cases *val*:
mk-Address-value (*addr*, *atype*, -) →
def *valid* = *exists-loc*(*addr*) in 335
(*valid* ∧ *matching-pointer-type* (*ptype*, *atype*)ρ → *value-of-a-variable*(*addr*) , 188 224
¬ *valid* → *non-mandatory-detection*(NONEXISTENT) *handler* , 397
¬ *matching-pointer-type* (*ptype*, *atype*)ρ → *non-mandatory-detection*(WRONG-TYPE) *handler*), 188 397

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
    NIL-VALUE →
        mandatory-detection(NIL-DEREFERENCE) handler                    396
end
```

6.8.5 Function Calls

A function call specifies the evaluation of any actual parameters and the activation and execution of the block associated with the function designator of the function call. The value of a function call is the value returned by the execution of this block.

Concrete Syntax

function call = function designator, actual parameters ;

function designator = value designator ;

Abstract Syntax

types

Function-call :: *desig* : *Function-designator*
aps : *Actual-parameters*;

Function-designator = *Value-designator*
| *Standard-function-designator*
| *System-function-designator*

annotations The identifiers of predefined procedures are distinguished since these do not denote values (that can be assigned or compared in expressions).

Static Semantics

The function designator of a function call shall be a value designator that denotes a function procedure or the identifier of a predefined function procedure.

NOTES

1 A value designator that denotes a function procedure is either a procedure identifier of a function procedure heading (possibly qualified by the name or names of modules from which the procedure identifier is exported), a value designator that denotes a constant that has a function procedure value, or a value designator that denotes a variable that has a function procedure type.

2 The identifiers of predefined procedures are not value designators since predefined procedures are not values that can be assigned, compared in expressions, or passed as procedure parameters.

In the case of a call of a predefined function procedure, the actual parameters shall correspond in number and in type to those required for that predefined function; otherwise, each actual parameter shall be parameter compatible with the corresponding formal parameter of the designated function procedure.

If the return type of the designated function procedure is a subrange type, then the type of the function call shall be the host type of that subrange type; otherwise it shall be the return type of the designated function procedure.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wf-function-call : *Function-call* → *Environment* → 𝔹

wf-function-call (*mk-Function-call* (*desig*, *aps*))ρ ≙
  wf-actual-parameters (*aps*)ρ ∧ 258
  (*is-Value-designator*(*desig*) → *wf-value-designator* (*desig*)ρ ∧ 223
    let *type* = *t-value-designator* (*desig*)ρ in 223
    let *struc* = *structure-of* (*type*)ρ in 315
    is-Function-procedure-structure(*struc*) ∧
    let *mk-Function-procedure-structure* (*parms*, -) = *struc* in
    is-parameters-match (*parms*, *aps*)ρ, 261
  is-Standard-function-designator(*desig*) → *wf-standard-function-designator* (*desig*)(*aps*)ρ, 289
  is-System-function-designator(*desig*) → *wf-system-function-designator* (*desig*)(*aps*)ρ) 348

t-function-call : *Function-call* → *Environment* → *Expression-typed*

t-function-call (*mk-Function-call* (*desig*, *aps*))ρ ≙
  let *rtype* = (*is-Value-designator*(*desig*) → let *type* = *t-value-designator* (*desig*)ρ in 223
    let *struc* = *structure-of* (*type*)ρ in 315
    is-Function-procedure-structure(*struc*) ∧
    let *mk-Function-procedure-structure* (-, *return*) = *struc* in
    return,
  is-Standard-function-designator(*desig*) → *t-standard-function-designator* (*desig*)(*aps*)ρ, 289
  is-System-function-designator(*desig*) → *t-system-function-designator* (*desig*)(*aps*)ρ) in 348
  host-type-of (*rtype*)ρ 325

Dynamic Semantics

The execution of a function call shall be the evaluation of any actual parameters to give corresponding arguments, the binding of the results of this evaluation to the corresponding formal parameters, and the execution of the actions associated with the function designator. In the case of a call of a predefined function, the actions are as specified for that predefined function; otherwise, the actions are as specified by the block of the function procedure declaration.

The order of evaluation, the order of accessing and the order of binding of the actual parameters shall be implementation-dependent. The order of evaluation of the function designator and the actual parameters shall be implementation-dependent.

The actual parameters shall be evaluated before the execution of the actions associated with the function procedure designator.

The value of a function call shall be the value returned by the execution of the associated actions.

m-function-call : *Function-call* → *Environment* $\xrightarrow{o}$ ()

m-function-call (*mk-Function-call* (*desig*, *aps*))ρ ≙
  def *argvals* = *m-actual-parameters*(*aps*) ρ in 259
  (*is-Value-designator*(*desig*) → def *fid* = *m-value-designator*(*desig*) ρ in 223
    let *handler* = *current-exception-handler* (ρ) in 318
    call-function-procedure-value(*fid*) (*argvals*) *handler*, 339
  is-Standard-function-designator(*desig*) → *m-standard-function-designator*(*desig*) (*argvals*) ρ, 290
  is-System-function-designator(*desig*) → *m-system-function-designator*(*desig*) (*argvals*) ρ) 348

annotations The above evaluation order for the function designator and the actual parameters is one of those permitted; the actual parameters may instead be evaluated before the function designator, or both may be evaluated in parallel — see 4.1.

6.8.6 Value Constructors

A value constructor is an expression denoting a value of an array type, a record type, or a set type. In the case of array constructors and record constructors a list of values, known as 'structure components', is specified to define the

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

values of the components of the array value or the fields of the record value. In the case of a set constructor a list of 'members' is specified, whose elements define the elements of the set value.

CHANGE — Array constructors and record constructors are not described in *Programming in Modula-2*.

NOTE — The syntax of each form of value constructor requires a type identifier, and so a value constructor cannot be used to denote a value for a variable or a constant of an anonymous type.

Concrete Syntax

value constructor = array constructor | record constructor | set constructor ;

Abstract Syntax

types

Value-constructor = *Array-constructor* | *Record-constructor* | *Set-constructor*

Static Semantics

wf-value-constructor : *Value-constructor* → *Environment* → 𝔹

wf-value-constructor $(vc)\rho \triangleq$
  cases *vc* :
    mk-Array-constructor (-, -) → *wf-array-constructor* $(vc)\rho$, 232
    mk-Record-constructor (-, -) → *wf-record-constructor* $(vc)\rho$, 237
    mk-Set-constructor (-, -) → *wf-set-constructor* $(vc)\rho$ 242
  end

t-value-constructor : *Value-constructor* → *Environment* → *Expression-typed*

t-value-constructor $(vc)\rho \triangleq$
  cases *vc* :
    mk-Array-constructor (-, -) → *t-array-constructor* $(vc)\rho$, 232
    mk-Record-constructor (-, -) → *t-record-constructor* $(vc)\rho$, 237
    mk-Set-constructor (-, -) → *t-set-constructor* $(vc)\rho$ 242
  end

Dynamic Semantics

m-value-constructor : *Value-constructor* → *Environment* $\xrightarrow{o}$ *Value*

m-value-constructor $(vc)\rho \triangleq$
  cases *vc*:
    mk-Array-constructor (-, -) → *m-array-constructor*(vc) ρ, 232
    mk-Record-constructor (-, -) → *m-record-constructor*(vc) ρ, 237
    mk-Set-constructor (-, -) → *m-set-constructor*(vc) ρ 242
  end

6.8.6.1 Array Constructors

An array constructor is an expression that denotes a value of a specified array type. The array type is specified by an array type identifier. The array value is specified by an array constructed value that has a structure component for each component of the array. The repetition of a structure component may be specified by giving an optional repetition factor.

NOTE 1 — An array constructor cannot denote a value for a formal parameter that is an open array parameter.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Concrete Syntax

array constructor = array type identifier, array constructed value ;

array type identifier = type identifier ;

EXAMPLE 1 — Array constructors.

Given the declarations:

```
TYPE StringType = ARRAY [0..24] OF CHAR;
     RowType = ARRAY [1..3] OF REAL;
     Array1Type = ARRAY [1..2] OF RowType;
     Array2Type = ARRAY [1..2], [1..3] OF REAL;
VAR  filler: CHAR;
     string: StringType;
     row: RowType;
     array1: Array1Type;
     array2: Array2Type;
```

the following statements are possible:

```
string:= StringType{' ' BY 25};
string:= StringType{filler BY 25};
string:= StringType{filler, ' ' BY 23, filler};
row:= RowType{1.0, 1.0, 1.0};
row:= RowType{1.0 BY 3};
array1:= Array1Type{row, row};
array1:= Array1Type{row BY 2};
array1:= Array1Type{RowType{1.0, 1.0, 1.0},
                    RowType{1.0, 1.0, 1.0}};
array1:= Array1Type{RowType{1.0, 1.0, 1.0} BY 2};
array1:= Array1Type{{1.0, 1.0, 1.0}, {1.0, 1.0, 1.0}};
array1:= Array1Type{{1.0, 1.0, 1.0} BY 2};
array1:= Array1Type{{1.0 BY 3} BY 2};
array2:= Array2Type{{1.0, 1.0, 1.0}, {1.0, 1.0, 1.0}};
array2:= Array2Type{{1.0, 1.0, 1.0} BY 2};
array2:= Array2Type{{1.0 BY 3} BY 2};
```

Abstract Syntax

types

Array-constructor :: *qid* : *Qualident*
def : *Array-constructed-value*

Static Semantics

The array type identifier of an array constructor shall be a type identifier that denotes an array type. The type of the array constructor shall be that array type.

The array constructed value of an array constructor shall be well-formed with respect to a given array type that is the type of the array constructor.

wf-array-constructor : *Array-constructor* → *Environment* → $\mathbb{B}$

wf-array-constructor (*mk-Array-constructor* (*qid*, *def*))ρ $\triangleq$
  wf-qualident (*qid*)ρ $\wedge$ 53
  let *type* = *type-of* (*qid*)ρ in 314
  wf-array-definition (*type*, *def*)ρ 233

t-array-constructor : *Array-constructor* → *Environment* → *Expression-typed*

t-array-constructor (*mk-Array-constructor* (*qid*, -))ρ $\triangleq$
  type-of (*qid*)ρ 314

Dynamic Semantics

The value of an array constructor shall be the value of the array constructed value evaluated as a value of the type of the array constructor.

m-array-constructor : *Array-constructor* → *Environment* $\xrightarrow{o}$ *Array-value*

m-array-constructor (*mk-Array-constructor* (*qid*, *def*))ρ $\triangleq$
  let *type* = *type-of* (*qid*)ρ in 314
  evaluate-array-definition(*type*, *def*) ρ 234

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations Note that an exception may be detected as each individual component is evaluated.

Array Constructed Values

An array constructed value denotes a value of an array type by specifying a structure component for each component of the array. It can either form part of an array constructor, or it can be used as a structure component of an array constructor or a record constructor.

Array constructed values are evaluated as a value of a given type. This type is specified explicitly by the type identifier in the case of array constructors. For structure components, the type is implicitly the component type of an array type or the corresponding field type of a record type.

ConcreteSyntax

array constructed value =
left brace, repeated structure component, { comma, repeated structure component }, right brace ;

Abstract Syntax

types

Array-constructed-value :: *vals* : *Repeated-elements*;

Repeated-elements = *Repeated-element**

Static Semantics

The sequence of repeated structure components of an array constructed value of a given array type shall be taken, one at a time, in the order that the repeated structure components appear in the sequence of repeated structure components. Each repeated structure component having a repetition factor of N shall lead to N consecutive occurrences of its structure component in a sequence of structure components. The length of this sequence of structure components shall be equal to the number of values of the range of values defined by the index type of the given array type.

The structure components shall be well-formed with respect to a structure component type that is the component type of the given array type.

wf-array-definition : *Typed* × *Array-constructed-value* → *Environment* → $\mathbb{B}$

wf-array-definition (*type*, *def*)ρ $\triangleq$
is-array-type (*type*)ρ ∧ *size-of-array-definition* (*def*)ρ = *size-of-array* (*type*)ρ ∧ 321 234 328
let *ctype* = *component-type-of* (*type*)ρ in 325
let *mk-Array-constructed-value* (*vals*) = *def* in
∀ *val* ∈ elems *vals* · *wf-repeated-element* (*ctype*, *val*)ρ 235

annotations Check that the number of structure components of an array definition is the same as the number of values of the range of values defined by the index type of the array type.

Dynamic Semantics

The value of an array constructed value, evaluated as a value of a given array type, shall be the value of that type for which the value of the Nth component is equal to the Nth value in the sequence of values formed by concatenating the results of the evaluation of the repeated structure components as sequences of values of the component type of the given array type.

The order of evaluation of the expressions of an array constructed value shall be implementation-dependent.

NOTE 2 — The evaluation of the expressions of an array constructed value could produce a transformation of the program state, i.e. have side-effects.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

evaluate-array-definition : *Typed* × *Array-constructed-value* → *Environment* $\xrightarrow{o}$ *Value*

evaluate-array-definition (*type*, *def*)ρ $\triangleq$
  let *mk-Array-constructed-value* (*vals*) = *def* in
  let *ctype* = *component-type-of* (*type*)ρ in 325
  def *evals* = *evaluate-repeated-elements*(*ctype*, *vals*) ρ in 234
  return *construct-array-value* (*type*, *evals*)ρ 234

Auxiliaries

size-of-array-definition : *Array-constructed-value* → *Environment* → N

size-of-array-definition (*def*)ρ $\triangleq$
  let *mk-Array-constructed-value* (*vals*) = *def* in
  number-of-repetitions (*vals*)ρ 234

annotations Return the number of structure components in an array definition.

number-of-repetitions : *Repeated-elements* → *Environment* → N

number-of-repetitions (*vals*)ρ $\triangleq$
  if *vals* = [ ]
  then 0
  else let *mk-Repeated-element* (-, *by*) = hd *vals* in
    evaluate-constant-expression (*by*)ρ + *number-of-repetitions* (tl *vals*)ρ 257 234

annotations Return the number of structure components in an array definition after expanding any repetitions.

construct-array-value : *Expression-typed* × *Value** → *Environment* → *Array-value*

construct-array-value (*type*, *vals*)ρ $\triangleq$
  let *itype* = *index-type-of* (*type*)ρ in 326
  let *indx* = *ordered-values-of* (*itype*)ρ in 330
  {*indx*(*i*) ↦ *vals*(*i*) | *i* ∈ inds *indx*}

annotations Construct an array value from an evaluated array definition.

evaluate-repeated-elements : *Typed* × *Repeated-elements* → *Environment* $\xrightarrow{o}$ *Value**

evaluate-repeated-elements (*type*, *vals*)ρ $\triangleq$
  if *vals* = [ ]
  then return [ ]
  else let *i* ∈ inds *vals* in
    def *fvals* = *evaluate-repeated-elements*(*type*, *front* (*vals*, *i* − 1)) ρ in 234 614
    def *mvals* = *evaluate-repeated-element*(*type*, *vals*(*i*)) ρ in 235
    def *bvals* = *evaluate-repeated-elements*(*type*, *rest* (*vals*, *i* + 1)) ρ in 234 614
    return *fvals* ⌢ *mvals* ⌢ *bvals*

annotations Evaluate the structure components of an array definition.

Repeated Structure Components

Repeated structure components are structure components that can be followed by a repetition factor.

Concrete Syntax

repeated structure component = structure component, ["BY", repetition factor] ;

repetition factor = constant expression ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Abstract Syntax

types

Repeated-element :: *val* : *Element*
by : *Expression*

annotations If the repetition factor is not present in the concrete representation, the translation process (from concrete to abstract syntax) will supply a *by* component in the abstract representation equal to 1.

Static Semantics

If a repeated structure component contains a repetition factor, the repetition factor shall be a constant expression of a whole number type and shall not have a value less than 0.

NOTE 3 — A repetition factor of 0 is allowed for parameterized source code.

EXAMPLE 2 — Parameterized repetion factors.

```
CONST
  mayChangeToZeroOneDay = 6;
TYPE
  T = ARRAY [0 .. mayChangeToZeroOneDay] OF CARDINAL;
VAR
  v: T;
BEGIN
  v := T{0, 1 BY mayChangeToZeroOneDay};
```

wf-repeated-element : *Typed* × *Repeated-element* → *Environment* → $\mathbb{B}$

wf-repeated-element (*type*, *rval*)ρ $\triangleq$
let *mk-Repeated-element* (*val*, *by*) = *rval* in
wf-element (*type*, *val*)ρ ∧ 236
wf-expression (*by*)ρ ∧ 190
is-constant-expression (*by*)ρ ∧ 254
is-whole-number-type (*t-expression* (*by*)ρ)ρ ∧ 323 190
evaluate-constant-expression (*by*)ρ ≥ 0 257

Dynamic Semantics

The structure component shall be evaluated to give a value of the given type. The value of a repeated structure component shall be the sequence of N occurrences of that value, where N is the value of the repetition factor, if given, or is the value 1, if a repetition factor is not given.

evaluate-repeated-element : *Typed* × *Repeated-element* → *Environment* $\xrightarrow{o}$ *Value**

evaluate-repeated-element (*type*, *rval*)ρ $\triangleq$
let *mk-Repeated-element* (*val*, *by*) = *rval* in
def *eval* = *evaluate-element*(*type*, *val*) ρ in 236
def *num* = *m-expression*(*by*) ρ in 190
return [*eval* | *i* ∈ {1, . . . , *num*}]

Structure Components

Structure components define the values of the components of an array constructed value or the fields of a record constructed value.

Concrete Syntax

structure component = expression | array constructed value | record constructed value | set constructed value ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Abstract Syntax

types

Element = *Expression* | *Array-constructed-value* | *Record-constructed-value* | *Set-constructed-value*

Static Semantics

If a structure component is an expression, the structure component type shall be assignment-compatible with the type of the expression.

If a structure component is an array, record, or set constructed value, the constructed value shall be well-formed with respect to a given type that is the structure component type.

wf-element : *Typed* × *Element* → *Environment* → 𝔹

wf-element (*type*, *def*)ρ ≙
(*is-Expression*(*def*) → *wf-expression* (*def*)ρ ∧ 190
is-assignment-compatible (*type*, *t-expression* (*def*)ρ)ρ, 123 190
is-Array-constructed-value(*def*) → *wf-array-definition* (*type*, *def*)ρ, 233
is-Record-constructed-value(*def*) → *wf-record-definition* (*type*, *def*)ρ, 238
is-Set-constructed-valueis(*def*) → *wf-set-definition* (*def*)ρ ∧ 243
wf-type-and-set-definition (*type*, *def*)ρ) 243

Dynamic Semantics

The value of a structure component that is an expression shall be the value of the expression. An exception shall be raised if the value is not within the range of values defined by the structure component type.

The value of a structure component that is an array constructed value, a record constructed value, or a set constructed value shall be the value of the constructed value evaluated as a value of the structure component type.

evaluate-element : *Typed* × *Element* → *Environment* $\xrightarrow{o}$ *Value*

evaluate-element (*type*, *val*)ρ ≙
(*is-Expression*(*val*) → def *res* = *m-expression*(*val*) ρ in 190
let *rng* = *values-of* (*type*)ρ in 329
if *res* ∈ *rng*
then return *res*
else let *handler* = *current-exception-handler* (ρ) in 318
mandatory-detection(CONSTRUCTOR-RANGE) *handler* , 396
is-Array-constructed-value(*val*) → *evaluate-array-definition*(*type*, *val*) ρ, 234
is-Record-constructed-value(*val*) → *evaluate-record-definition*(*type*, *val*) ρ, 239
is-Set-constructed-value(*val*) → def *res* = *m-set-definition*(*val*) ρ in 244
evaluate-set-definition(*type*, *res*) ρ) 244

6.8.6.2 Record Constructors

A record constructor is an expression that denotes a value of a specified record type. The record type is specified by a record type identifier. The record value is specified by a record constructed value that has a structure component for each field of the record.

Concrete Syntax

record constructor = record type identifier, record constructed value ;

record type identifier = type identifier ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLE — Record constructors.

Given the declarations:

```
TYPE NameType = ARRAY [0..24] OF CHAR;
     DateType = RECORD
                   year, month, day: CARDINAL
                END ;
     PersonType = RECORD
                     name: NameType;
                     DateOfBirth: DateType
                  END ;
VAR  day, month, year: CARDINAL;
     date: DateType;  person: PersonType;
```

the following statements are possible:

```
SWholeIO.ReadCard(year);
SWholeIO.ReadCard(month);
SWholeIO.ReadCard(day);
date:= DateType{year, month, day};
date:= DateType{1623, 6, 19};
person:= PersonType{"Blaise Pascal", date};
person:=
  PersonType{"Blaise Pascal", DateType{1623, 6, 19}};
nobody:=
  PersonType{NameType{"" BY 25}, DateType{0, 0, 0}};
nobody:= PersonType{{"" BY 25}, {0, 0, 0}};
```

Abstract Syntax

types

Record-constructor :: *qid* : *Qualident*
def : *Record-constructed-value*

Static Semantics

The record type identifier of a record constructor shall be a type identifier that denotes a record type. The type of the record constructor shall be that record type.

The record constructed value of a record constructor shall be well-formed with respect to a given record type that is the type of the record constructor.

wf-record-constructor : *Record-constructor* → *Environment* → $\mathbb{B}$

wf-record-constructor (*mk-Record-constructor* (*qid*, *def*))ρ $\triangleq$
  wf-qualident (*qid*)ρ ∧ 53
  let *type* = *type-of* (*qid*)ρ in 314
  wf-record-definition (*type*, *def*)ρ 238

t-record-constructor : *Record-constructor* → *Environment* → *Expression-typed*

t-record-constructor (*mk-Record-constructor* (*qid*, -))ρ $\triangleq$
  type-of (*qid*)ρ 314

Dynamic Semantics

The value of a record constructor shall be the value of the record constructed value evaluated as a value of the type of the record constructor.

m-record-constructor : *Record-constructor* → *Environment* $\xrightarrow{o}$ *Record-value*

m-record-constructor (*mk-Record-constructor* (*qid*, *def*))ρ $\triangleq$
  let *type* = *type-of* (*qid*)ρ in 314
  evaluate-record-definition(*type*, *def*) ρ 239

annotations Note that an exception may be detected as each individual component is evaluated.

Record Constructed Values

A record constructed value denotes a value of a record type by specifying a structure component for each field of the record. It can either form part of a record constructor, or it can be used as a structure component of an array constructor or a record constructor.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Record constructed values are evaluated as a value of a given type. This type is specified explicitly by the type identifier in the case of record constructors. For structure components, the type is implicitly the component type of an array type or the corresponding field type of a record type.

Concrete Syntax

record constructed value = left brace, [structure component, { comma, structure component }], right brace ;

Abstract Syntax

types

Record-constructed-value :: *vals* : *Elements*;

Elements = *Element**

Static Semantics

A sequence of fields shall be formed from the fields of the given record type taken, one at a time, in the order that the fields textually appear in the declaration of the type denoted by the given record type. If the Nth field of the sequence of fields is a tag field, then the Nth structure component of the sequence of structure components of the record constructed value shall be an expression that is a constant expression. If there is no else variant part associated with the tag field, then the value of the constant expression shall be such that there is a variant associated with the value of the constant expression. The fields of any variant that is not associated with the value of the constant expression shall be excluded from the sequence of fields. If no variant is associated with the value of the constant expression, then the fields of the else variant part shall be included in the sequence of fields.

NOTE 1 — A constant expression must be supplied for a tag field even if the tag field is an implicit tag field.

The length of the sequence of structure components of the record constructed value shall be equal to the length of the sequence of fields.

The Nth structure component of the sequence of structure components shall be well-formed with respect to a structure component type that is the type of the Nth field of the sequence of fields.

wf-record-definition : *Typed* × *Record-constructed-value* → *Environment* → $\mathbb{B}$

wf-record-definition (*type*, *def*)ρ $\triangleq$
  is-record-type (*type*)ρ ∧ *size-of-record-definition* (*def*)ρ = *size-of-record* (*type*, *def*)ρ ∧ 321 239 239
  let *mk-Record-constructed-value* (*vals*) = *def* in
  let *mk-Record-structure* (*struc*) = *structure-of* (*type*)ρ in 315
  match-fields-list (*struc*, *vals*)ρ 240

annotations Check that the number of structure components of a record definition is the same as the number of fields of the record type, taking into account tag field variant selection.

Dynamic Semantics

The value of a record constructed value, evaluated as a value of a given record type, shall be the value of that type for which the value of the Nth field is equal to the Nth value in the sequence of values formed by evaluating the structure components as values of the corresponding field type of the given record type.

The order of evaluation of the expressions of a record constructed value shall be implementation-dependent.

NOTE 2 — The evaluation of the expressions of a record constructed value could produce a transformation of the program state, i.e. have side-effects.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

evaluate-record-definition : *Typed* × *Record-constructed-value* → *Environment* $\xrightarrow{o}$ *Value*

evaluate-record-definition (*type*, *def*)ρ $\triangleq$
  let *mk-Record-constructed-value* (*vals*) = *def*,
    mk-Record-structure (*struc*) = *structure-of* (*type*)ρ in 315
  construct-fields-list-value(*struc*, *vals*) ρ 241

Auxiliaries

size-of-record-definition : *Record-constructed-value* → *Environment* → N

size-of-record-definition (*def*)ρ $\triangleq$
  let *mk-Record-constructed-value* (*vals*) = *def* in
  len *vals*

size-of-record : *Typed* × *Record-constructed-value* → *Environment* → N

size-of-record (*type*, *def*)ρ $\triangleq$
  let *mk-Record-structure* (*struc*) = *structure-of* (*type*)ρ in 315
  let *mk-Record-constructed-value* (*vals*) = *def* in
  size-of-fields-list (*struc*, *vals*)ρ 239

size-of-fields-list : *Fields-list-structure* × *Elements* → *Environment* → N

size-of-fields-list (*fsl*, *vals*)ρ $\triangleq$
  if *fsl* = [] ∨ *vals* = []
  then 0
  else let *size* = *size-of-fields* (hd *fsl*, *vals*)ρ in 239
    let *rvals* = *rest* (*vals*, *size* + 1) in 614
    size + *size-of-fields-list* (tl *fsl*, *rvals*)ρ 239

annotations Calculate the number of components in a fields-list.

size-of-fields : *Fields-structure* × *Elements* → *Environment* → N

size-of-fields (*fs*, *vals*)ρ $\triangleq$
  (*is-Fixed-fields-structure*(*fs*) → *size-of-fixed-fields* (*fs*), 239
   is-Variant-fields-structure(*fs*)→ *size-of-variant-fields* (*fs*, *vals*)ρ) 239

annotations Calculate the number of components in a fields.

size-of-fixed-fields : *Fixed-fields-structure* → N

size-of-fixed-fields (*ffs*) $\triangleq$
  let *mk-Fixed-fields-structure* (*ids*, -) = *ffs* in
  len *ids*

annotations Calculate the number of components in a fixed-fields.

size-of-variant-fields : *Variant-fields-structure* × *Elements* → *Environment* → N

size-of-variant-fields (*vstruc*, *vals*)ρ $\triangleq$
  let *mk-Variant-fields-structure* (-, -, *variants*, *other*) = *vstruc* in
  let *tagval* = *evaluate-constant-expression* (hd *vals*)ρ in 257
  if ∃ *i* ∈ inds *variants* · *tagval* ∈ *variants*(*i*).*labels*
  then let *i* ∈ inds *variants* be st *tagval* ∈ *variants*(*i*).*labels* in
    size-of-variant (*variants*(*i*), tl *vals*)ρ + 1 240
  elseif *other* ≠ nil
  then *size-of-fields-list* (*other*, tl *vals*)ρ + 1 239
  else 0

annotations Calculate the number of components in a variant component of a record structure, selected by the value of the tag field.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

size-of-variant : *Variant-structure* × *Elements* → *Environment* → ℕ

size-of-variant (*vstruc*, *vals*)ρ ≜
  let *mk-Variant-structure* (-, *fields*) = *vstruc* in
  size-of-fields-list (*fields*, *vals*)ρ 239

annotations Calculate the number of components in a variant component of a record structure.

match-fields-list : *Fields-list-structure* × *Elements* → *Environment* → 𝔹

match-fields-list (*fsl*, *vals*)ρ ≜
  if *fsl* = [ ]
  then *vals* = [ ]
  else *match-fields* (hd *fsl*, *vals*) ∧ 240
    let *size* = *size-of-fields* (hd *fsl*, *vals*)ρ in 239
    let *rvals* = *rest* (*vals*, *size* + 1) in 614
    match-fields-list (tl *fsl*, *rvals*)ρ 240

annotations Check that the structure defined by a fields-list matches the structure given by an expression sequence component of a structured record value.

match-fields : *Fields-structure* × *Elements* → *Environment* → 𝔹

match-fields (*fs*, *vals*) ≜
  (*is-Fixed-fields-structure*(*fs*) → *match-fixed-fields* (*fs*, *vals*)ρ, 240
   is-Variant-fields-structure(*fs*) → *match-variant-fields* (*fs*, *vals*)ρ) 240

annotations Check that the structure defined by a fields matches the structure given by an expression sequence component of a structured record value.

match-fixed-fields : *Fixed-fields-structure* × *Elements* → *Environment* → 𝔹

match-fixed-fields (*ffs*, *vals*)ρ ≜
  let *mk-Fixed-fields-structure* (*ids*, *type*) = *ffs* in
  ∀ *i* ∈ inds *ids* · *wf-element* (*type*, *vals*(*i*))ρ 236

annotations Check that the type of each expression in the record constructor is assignment-compatible with the type of the corresponding component of a record type.

match-variant-fields : *Variant-fields-structure* × *Elements* → *Environment* → 𝔹

match-variant-fields (*vfs*, *vals*)ρ ≜
  let *mk-Variant-fields-structure* (-, *tagt*, *variants*, *other*) = *vfs* in
  is-constant-expression (hd *vals*)ρ ∧ 254
  wf-element (*tagt*, hd *vals*)ρ ∧ 236
  let *tagv* = *evaluate-constant-expression* (hd *vals*)ρ in 257
  if ∃ *i* ∈ inds *variants* · *tagv* ∈ *variants*(*i*).*labels*
  then let *i* ∈ inds *variants* be st *tagv* ∈ *variants*(*i*).*labels* in
    match-variant (*variants*(*i*), tl *vals*)ρ 240
  elseif *other* ≠ nil
  then *match-fields-list* (*other*, tl *vals*)ρ 240
  else undefined

annotations Check that the type of each expression in the record constructor is assignment-compatible with the type of the corresponding component of the record type; also check that the expression that corresponds to any tag field is a constant expression. The result is undefined if the structured value does not match the type.

match-variant : *Variant-structure* × *Elements* → *Environment* → 𝔹

match-variant (*variant*, *vals*)ρ ≜
  let *mk-Variant-structure* (-, *fields*) = *variant* in
  match-fields-list (*fields*, *vals*)ρ 240

annotations Check the expressions in the record constructor that correspond to a variant component.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

construct-fields-list-value : *Fields-list-structure* × *Elements* → *Environment* $\xrightarrow{o}$ *Record-value*

construct-fields-list-value (*fsl*, *vals*)ρ $\triangleq$
  if *fsl* = [ ]
  then return { }
  else def *rval* = *construct-fields-value*(hd *fsl*, *vals*) ρ in 241
    let *size* = card dom *rval* in
    let *rem* = *rest* (*vals*, *size* + 1) in 614
    def *lval* = *construct-fields-list-value*(tl *fsl*, *rem*) ρ in 241
    return *rval* ∪ *lval*

annotations Build a record value corresponding to a fields-list component of a record from an expression sequence.

construct-fields-value : *Fields-structure* × *Elements* → *Environment* $\xrightarrow{o}$ *Record-value*

construct-fields-value (*fs*, *vals*)ρ $\triangleq$
  (*is-Fixed-fields-structure*(*fs*) → *construct-fixed-fields-value*(*fs*, *vals*) ρ, 241
   is-Variant-fields-structure(*fs*) → *construct-variant-fields-value*(*fs*, *vals*) ρ) 241

annotations Build a record value corresponding to a fields component of a record from an expression sequence.

construct-fixed-fields-value : *Fixed-fields-structure* × *Expression** → *Environment* $\xrightarrow{o}$ *Record-value*

construct-fixed-fields-value (*fs*, *vals*)ρ $\triangleq$
  let *mk-Fixed-fields-structure* (*ids*, *type*) = *fs* in
  dcl *rv* : *Value* $\xrightarrow{m}$ *Value* := {↦};
  for all *i* ∈ inds *ids*
  do def *ival* = *evaluate-element*(*type*, *vals*(*i*)) ρ in 236
    rv := *rv* † {*ids*(*i*) ↦ *ival*}

annotations Build a record value corresponding to a fixed-fields component of a record from an expression sequence.

construct-variant-fields-value : *Variant-fields-structure* × *Elements* → *Environment* $\xrightarrow{o}$ *Record-value*

construct-variant-fields-value (*vfs*, *vals*)ρ $\triangleq$
  let *mk-Variant-fields-structure* (*id*, -, *variants*, *other*) = *vfs* in
  def *tagval* = *m-expression*(hd *vals*) ρ in 190
  let *tag* = if *id* ≠ nil
        then {*id* ↦ *tagval*}
        else {↦} in
  if ∃ *i* ∈ inds *variants* · *tagval* ∈ *variants*(*i*).*labels*
  then let *i* ∈ inds *variants* be st *tagval* ∈ *variants*(*i*).*labels* in
    def *vval* = *construct-variant-value*(*variants*(*i*), tl *vals*) ρ in 241
    return *vval* ∪ *tag*
  else def *fval* = *construct-fields-list-value*(*other*, tl *vals*) ρ in 241
    return *fval* ∪ *tag*

annotations Build a record value corresponding to a variant-fields component of a record from an expression sequence.

construct-variant-value : *Variant-structure* × *Elements* → *Environment* $\xrightarrow{o}$ *Record-value*

construct-variant-value (*vs*, *vals*)ρ $\triangleq$
  let *mk-Variant-structure* (-, *fields*) = *vs* in
  construct-fields-list-value(*fields*, *vals*) ρ 241

annotations Build a record value corresponding to a variant component of a record from an expression sequence.

6.8.6.3 Set Constructors

A set constructor is an expression that denotes a value of a specified set type or packedset type. The type is specified by a set type or packedset type identifier. The set or packedset value is specified by a set constructed value that specifies each of the elements of the set. An individual element can be specified by using an expression, and several elements can be specified by using an interval.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Concrete Syntax

set constructor = set type identifier, set constructed value ;

set type identifier = type identifier ;

CHANGE — In *Programming in Modula-2* the set type identifier of a set constructor may be omitted, in which case it implicitly denotes the standard type BITSET.

EXAMPLE — Set constructors.

Given the declarations:

```
TYPE YearType = CARDINAL;
     PeriodType = SET OF YearType;
     EntryType = RECORD
                    name: NameType;
                    lifetime: PeriodType
                 END ;
VAR  year: CARDINAL;
     period: PeriodType;  entry: EntryType;
```

the following statements are possible:

```
period:= PeriodType{1982, 1983, 1985, 1988};
SWholeIO.ReadCard(year);
period:= PeriodType{year-4..year+4};
period:= PeriodType{1623..1662};
entry:= EntryType{"Blaise Pascal", period};
entry:= EntryType{"Blaise Pascal",
                  PeriodType{1623..1662}};
entry:= EntryType{"Blaise Pascal", {1623..1662}};
```

Abstract Syntax

types

Set-constructor :: *qid* : *Qualident*
def : *Set-constructed-value*

Static Semantics

The set type identifier of a set constructor shall be a type identifier that denotes a set type or a packedset type. The type of the set constructor shall be that set type or packedset type.

The set constructed value of a set constructor shall be well-formed with respect to a given set type that is the type of the set constructor.

wf-set-constructor : *Set-constructor* → *Environment* → $\mathbb{B}$

wf-set-constructor (*mk-Set-constructor* (*qid*, *def*))ρ $\triangleq$
  wf-qualident (*qid*)ρ $\wedge$ 53
  wf-set-definition (*def*)ρ $\wedge$ 243
  let *type* = *type-of* (*qid*)ρ in 314
  wf-type-and-set-definition (*type*, *def*)ρ 243

t-set-constructor : *Set-constructor* → *Environment* → *Expression-typed*

t-set-constructor (*mk-Set-constructor* (*qid*, -))ρ $\triangleq$
  type-of (*qid*)ρ 314

Dynamic Semantics

The value of a set constructor shall be the value of its set constructed value evaluated as a value of the type of the set constructor.

m-set-constructor : *Set-constructor* → *Environment* $\xrightarrow{o}$ *Set-value*

m-set-constructor (*mk-Set-constructor* (*qid*, *def*))ρ $\triangleq$
  let *type* = *base-type-of* (*qid*)ρ in 325
  def *res* = *m-set-definition*(*def*) ρ in 244
  evaluate-set-definition(*type*, *res*) ρ 244

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Set Constructed Values

A set constructed value denotes a value of a set type, by specifying each of the elements of the set. It can either form part of a set constructor, or it can be used as a structure component of an array constructor or a record constructor. An individual element of the set can be specified by using an expression, and several elements can be specified by using an interval.

Set constructed values are evaluated as a value of a given type. This type is specified explicitly by the type identifier in the case of set constructors. For structure components, the type is implicitly the component type of an array type or the corresponding field type of a record type.

NOTES

1 The construct '{}' denotes the empty set of the given set type. An empty set is a value of every set type.

2 Since the base type of a set type contains at least one value, there are at least two values of a set type.

Concrete Syntax

set constructed value = left brace, [member, { comma, member }], right brace ;

Abstract Syntax

types

Set-constructed-value = *Member*-set

Static Semantics

For each element of the set constructed value, the base type of the given set type shall be assignment-compatible with the type of the element.

CLARIFICATION — In *Programming in Modula-2* it is not stated that, for each element of the set constructed value, the base type of the given set type has to be assignment-compatible with the type of the element.

wf-type-and-set-definition : *Typed* × *Set-constructed-value* → *Environment* → $\mathbb{B}$

wf-type-and-set-definition (*type*, *def*)ρ $\triangleq$
  is-set-type (*type*)ρ $\wedge$ 322
  let *btype* = *base-type-of* (*type*)ρ in 325
  is-assignment-compatible (*btype*, *t-set-definition* (*def*)ρ)ρ 123 243

wf-set-definition : *Set-constructed-value* → *Environment* → $\mathbb{B}$

wf-set-definition (*sdef*)ρ $\triangleq$
  $\forall$ *member* $\in$ *sdef* · *wf-member* (*member*)ρ $\wedge$ 245
  $\forall$ $mem_1, mem_2 \in$ *sdef* · *is-expression-compatible* (*t-member* (mem_1)ρ, *t-member* (mem_2)ρ) 121 245 245

t-set-definition : *Set-constructed-value* → *Environment* → *Expression-typed*

t-set-definition (*sdef*)ρ $\triangleq$
  let *mtypes* = {*t-member* (*mem*)ρ | *mem* $\in$ *sdef*} in 245
  type-of-set (*mtypes*) 244

annotations In the VDM-SL modelling of a Modula-2 set, the *Set-constructed-value* is given a type according to the types of the elements of the set. The given type is tested for assignment-compatibility with the type of the *Set-constructed-value*.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Dynamic Semantics

Each member of a set constructed value shall be evaluated to give a value. The value of the set constructed value shall be the union of these values.

An exception shall be raised if at least one element of this value is not within the range of values defined by the base type of the given set type.

The order of evaluation of the expressions of a set constructed value shall be implementation-dependent.

NOTE 3 — The evaluation of the expressions of a set constructor could produce a transformation of the program state, i.e. have side-effects.

m-set-definition : *Set-constructed-value* → *Environment* $\xrightarrow{o}$ *Value*-set

m-set-definition (*sdef*)ρ $\triangleq$
  (dcl *setv* : *Value*-set := { };
   for all *member* ∈ *sdef*
   do def *val* = *m-member*(*member*) ρ in 245
      setv := *setv* ∪ {*val*};
   return ⋃ *setv*)

Auxiliaries

evaluate-set-definition : *Typed* × *Value*-set → *Environment* $\xrightarrow{o}$ *Set-value*

evaluate-set-definition (*type*, *vals*)ρ $\triangleq$
  let *values* = *values-of* (*type*)ρ in 329
  if *vals* ⊆ *values*
  then return *mk-Set-value* (*vals*)
  else let *handler* = *current-exception-handler* (ρ) in 318
       mandatory-detection(SET-RANGE) *handler* 396

type-of-set : *Expression-typed*-set → *Expression-typed*

type-of-set (*types*) $\triangleq$
  cases *types* :
    {*type*} → *type*,
    {*type*1, *type*2} → *whole-number-result-type* (*type*1, *type*2), 199
    {*type*1} ∪ *rest* → let *rtype* = *type-of-set*(*rest*) in 244
                         whole-number-result-type (*type*1, *rtype*) 199
  end

Members

A member denotes a set value that either contains values from a range of values or contains a single value.

Concrete Syntax

member = interval | singleton ;

Abstract Syntax

types

  Member = *Interval* | *Singleton*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Static Semantics

If a member is an interval, then the type of the member shall be the type of the interval; otherwise it shall be the type of the singleton.

$wf\text{-}member : Member \rightarrow Environment \rightarrow \mathbb{B}$

$wf\text{-}member\,(mem)\rho \triangleq$
$\quad (is\text{-}Interval(mem) \quad \rightarrow wf\text{-}interval\,(mem)\rho,$ 245
$\quad\ is\text{-}Singleton(mem) \rightarrow wf\text{-}singleton\,(mem)\rho)$ 247

$t\text{-}member : Member \rightarrow Environment \rightarrow Expression\text{-}typed$

$t\text{-}member\,(mem)\rho \triangleq$
$\quad (is\text{-}Interval(mem) \quad \rightarrow t\text{-}interval\,(mem)\rho,$ 246
$\quad\ is\text{-}Singleton(mem) \rightarrow t\text{-}singleton\,(mem)\rho)$ 247

Dynamic Semantics

If a member is an interval, then the value of the member shall be the value of the interval; otherwise it shall be the value of the singleton.

$m\text{-}member : Member \rightarrow Environment \xrightarrow{o} Value\text{-}\mathsf{set}$

$m\text{-}member\,(mem)\rho \triangleq$
$\quad (\ is\text{-}Interval(mem) \quad \rightarrow m\text{-}interval(mem)\,\rho,$ 246
$\quad\ is\text{-}Singleton(mem) \rightarrow m\text{-}singleton(mem)\,\rho)$ 247

Intervals

An interval denotes a set value whose elements are defined by a range of values.

Concrete Syntax

interval = ordinal expression, ellipsis, ordinal expression ;

Abstract Syntax

types

$Interval :: min : Expression$
$\qquad\qquad max : Expression$

Static Semantics

Each ordinal expression of an interval shall be an expression that is of an ordinal type. The types of the ordinal expressions shall be expression-compatible.

The base type of the interval shall be the type of each ordinal expression, if they are of the same type. If the type of one ordinal expression is the $\mathbb{Z}$-type, and the type of the other ordinal expression is the signed type, the base type of the interval shall be the signed type. If the type of one ordinal expression is the $\mathbb{Z}$-type, and the type of the other ordinal expression is the unsigned type, the base type of the interval shall be the unsigned type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wf-interval : *Interval* → *Environment* → $\mathbb{B}$

wf-interval (*mk-Interval* (*min*, *max*))ρ $\triangleq$
  wf-expression (*min*)ρ ∧ 190
  wf-expression (*max*)ρ ∧ 190
  let *ltype* = *t-expression* (*min*)ρ in 190
  let *rtype* = *t-expression* (*max*)ρ in 190
  is-ordinal-type (*ltype*)ρ ∧ 322
  is-ordinal-type (*rtype*)ρ ∧ 322
  is-expression-compatible (*ltype*, *rtype*)ρ 121

t-interval : *Interval* → *Environment* → *Expression-typed*

t-interval (*mk-Interval* (*min*, *max*))ρ $\triangleq$
  let *tmin* = *t-expression* (*min*)ρ in 190
  let *tmax* = *t-expression* (*max*)ρ in 190
  if *is-whole-number-type* (*tmin*) 323
  then *whole-number-result-type* (*tmin*, *tmax*) 199
  else *tmin*

Dynamic Semantics

The value of an interval of the form '`x..y`', where `x` and `y` are ordinal expressions, shall be an empty set if the value of `x` is greater than the value of `y`; otherwise it shall be a set whose elements have the values in the closed interval from the value of `x` to the value of `y`.

m-interval : *Interval* → *Environment* $\xrightarrow{o}$ *Value*-set

m-interval (*mk-Interval* (*min*, *max*))ρ $\triangleq$
  def *vmin* = *m-expression*(*min*) ρ in 190
  def *vmax* = *m-expression*(*max*) ρ in 190
  return $\{x \mid x \in \mathit{scalar\text{-}set}\,(\mathit{vmin}) \cdot \mathit{scalar\text{-}value}\,(x) \in \{\mathit{scalar\text{-}value}\,(\mathit{vmin}), \ldots, \mathit{scalar\text{-}value}\,(\mathit{vmax})\}\}$ 246 210 210 210

annotations The above evaluation order for the minimum expression and the maximum expression is one of those permitted; the maximum expression may instead be evaluated before the minimum expression, or both may be evaluated in parallel — see 4.1.

Auxiliaries

scalar-set : *Value* → *Value*-set

scalar-set (*value*) $\triangleq$
  (*is-Enumeration-value*(*value*) → let *mk-Enumeration-value* (-, *ids*) = *value* in
    $\{\mathit{mk\text{-}Enumeration\text{-}value}\,(\mathit{id}, \mathit{ids}) \mid \mathit{id} \in \mathit{ids}\}$,
  value : *Char* → dom *collation-map*$_*$,
  value : *String-value* → dom *collation-map*,
  value : $\mathbb{B}$ → {true, false},
  others → $\{x \mid x : \mathbb{Z}\}$)

Singletons

A singleton denotes a set value with one element, defined by a expression.

Concrete Syntax

singleton = ordinal expression ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Abstract Syntax

types

Singleton :: *expr* : *Expression*

Static Semantics

The ordinal expression of a singleton shall be an expression that is of an ordinal type.

The base type of a singleton shall be the type of the ordinal expression.

wf-singleton : *Singleton* → *Environment* → 𝔹
wf-singleton (*mk-Singleton* (*expr*))ρ ≙
  wf-expression (*expr*)ρ ∧ 190
  let *type* = *t-expression* (*expr*)ρ in 190
  is-ordinal-type (*type*)ρ 322

t-singleton : *Singleton* → *Environment* → *Expression-typed*
t-singleton (*mk-Singleton* (*expr*))ρ ≙
  t-expression (*expr*)ρ 190

Dynamic Semantics

The value of a singleton shall be a set of one element that has the value of the ordinal expression.

m-singleton : *Singleton* → *Environment* $\xrightarrow{o}$ *Value*-set
m-singleton (*mk-Singleton* (*expr*))ρ ≙
  def *value* = *evaluate-to-ordinal*(*expr*) ρ in 253
  return {*value*}

6.8.7 Constant Literals

A constant literal denotes a constant value.

Concrete Syntax

constant literal = whole number literal | real literal | string literal ;

NOTE — There is no denotation for a complex literal. However, a value of the ℂ-type can be formed by applying the standard function `CMPLX` to two real literals; for example '`CMPLX(0.0, 1.0)`' — see 6.10.4.5.

Abstract Syntax

types

Constant-literal = *Whole-number-literal* | *Real-literal* | *String-literal*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Static Semantics

wf-constant-literal : *Value-designator* → *Environment* → $\mathbb{B}$

wf-constant-literal (*val*)ρ $\triangleq$
  cases *val* :
    mk-Whole-number-literal (-) → *wf-whole-number-literal* (*val*)ρ, 249
    mk-Real-literal (-) → *wf-real-literal* (*val*)ρ, 250
    mk-String-literal (-) → *wf-string-literal* (*val*)ρ 251
  end

t-constant-literal : *Value-designator* → *Environment* → *Expression-typed*

t-constant-literal (*val*)ρ $\triangleq$
  cases *val* :
    mk-Whole-number-literal (-) → *t-whole-number-literal* (*val*)ρ, 249
    mk-Real-literal (-) → *t-real-literal* (*val*)ρ, 250
    mk-String-literal (-) → *t-string-literal* (*val*)ρ 251
  end

Dynamic Semantics

m-constant-literal : *Value-designator* → *Environment* $\xrightarrow{o}$ *Value*

m-constant-literal (*val*)ρ $\triangleq$
  cases *val*:
    mk-Whole-number-literal (-) → *m-whole-number-literal*(*val*) ρ, 249
    mk-Real-literal (-) → *m-real-literal*(*val*) ρ, 250
    mk-String-literal (-) → *m-string-literal*(*val*) ρ 251
  end

6.8.7.1 Whole Number Literals

A whole number literal denotes a constant value of a whole number type. It can be denoted by means of decimal, octal, or hexadecimal notation.

Concrete Syntax

NOTE 1 — The syntax for whole number literals is given in the Lexis — see section 5.

The radix of a decimal number shall be ten; the radix of an octal number shall be eight; and the radix of a hexadecimal number shall be sixteen.

No separators shall appear within a whole number literal.

NOTES

2 The additional requirement of no separators must be made since this is not a consequence of the concrete syntax.

3 A whole number literal has a non-negative value. The syntax does not permit a whole number literal that has a negative value. A whole number literal preceded by an arithmetic negation operator is a prefix expression — see 6.8.3. Such an expression has a non-positive value of the $\mathbb{Z}$-type.

Abstract Syntax

types

Whole-number-literal :: *value* : $\mathbb{N}$

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Static Semantics

The value denoted by a whole number literal shall be less than or equal to the implementation-defined maximum value for values of the $\mathbb{Z}$-type.

The type of a whole number literal shall be the $\mathbb{Z}$-type.

NOTES

4 There is no identifier that denotes the maximum value for values of the $\mathbb{Z}$-type.

5 This maximum value is at least as large as the maximum value of any whole number type. — see 6.10.2.6.2.

wf-whole-number-literal : *Whole-number-literal* → *Environment* → $\mathbb{B}$

wf-whole-number-literal (*mk-Whole-number-literal* (*value*))ρ $\triangleq$
  value $\in$ {0, ..., *maximum* (Z-TYPE)ρ} 328

t-whole-number-literal : *Whole-number-literal* → *Environment* → *Expression-typed*

t-whole-number-literal (*mk-Whole-number-literal* (-))ρ $\triangleq$
  Z-TYPE

Dynamic Semantics

The value of a whole number literal shall be the numerical value denoted by the whole number literal.

m-whole-number-literal : *Whole-number-literal* → *Environment* $\xrightarrow{o}$ $\mathbb{Z}$

m-whole-number-literal (*mk-Whole-number-literal* (*value*))ρ $\triangleq$
  return *value*

6.8.7.2 Real Literals

A real literal denotes a constant value of a real number type. It is denoted by means of a decimal notation, using an optional scale factor.

Concrete Syntax

NOTE 1 — The syntax for real literals is given in the Lexis — see section 5.

A real literal shall be written in decimal notation with an optional scale factor. The scale factor shall give the power of ten by which the remaining part of the real literal is multiplied to produce the value of the real literal.

No separators shall appear within a real literal.

NOTES

2 The additional requirement of no separators must be made since this is not a consequence of the concrete syntax.

3 A real literal has a non-negative value. The syntax does not permit a real literal that has a negative value. A real literal preceded by an arithmetic negation operator is a prefix expression — see 6.8.3. Such an expression has a non-positive value of the $\mathbb{R}$-type.

4 A real literal always contains a decimal point.

Abstract Syntax

types

  Real-literal :: *value* : $\mathbb{R}$

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Static Semantics

The value denoted by a real literal shall be less than or equal to the implementation-defined maximum value for values of the $\mathbb{R}$-type.

The type of a real literal shall be the $\mathbb{R}$-type.

NOTES

5 There is no identifier that denotes the maximum value for values of the $\mathbb{R}$-type.

6 This maximum value is at least as large as the maximum value of any real number type. — see 6.10.2.6.2.

wf-real-literal : *Real-literal* → *Environment* → $\mathbb{B}$

wf-real-literal (*mk-Real-literal* (*value*))ρ $\triangleq$
  $0 \leq$ *value* $\wedge$
  value $\leq$ *maximum* ($\mathbb{R}$-TYPE)ρ 328

t-real-literal : *Real-literal* → *Environment* → *Expression-typed*

t-real-literal (*mk-Real-literal* (-))ρ $\triangleq$
  $\mathbb{R}$-TYPE

Dynamic Semantics

The value of a real literal shall be an implementation-defined approximation to the numerical value denoted by the real literal.

m-real-literal : *Real-literal* → *Environment* $\xrightarrow{o}$ $\mathbb{R}$

m-real-literal (*mk-Real-literal* (*value*))ρ $\triangleq$
  return *real-constant-approximation* (*value*) 197

6.8.7.3 String Literals

There are two forms of string literal, namely quoted strings and character number literals. A quoted string denotes a constant value of the string literal type which is a sequence of zero or more characters. A character number literal denotes a constant value of the string literal type of length one. String literals of length zero or length one are also known as character literals.

Concrete Syntax

NOTES

1 The syntax for string literals is given in the Lexis — see 5.4.3.

2 A string literal is not equivalent to a concrete string constant constructed with an array constructor. For example, values of the string literal type cannot be indexed, concrete string constants cannot be concatenated using the catenation symbol, and the compatibility rules for string literals are more relaxed than those for concrete string constants.

Abstract Syntax

types

String-literal = *Character-string* | *Character-number-literal*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Static Semantics

wf-string-literal : *String-literal* → *Environment* → 𝔹

wf-string-literal (*val*)ρ ≜
  cases *val* :
    mk-Character-string (-) → *wf-character-string* (*val*)ρ, 251
    mk-Character-number-literal (-) → *wf-character-number-literal* (*val*)ρ 252
  end

t-string-literal : *String-literal* → *Environment* → *Expression-typed*

t-string-literal (*val*)ρ ≜
  cases *val* :
    mk-Character-string (-) → *t-character-string* (*val*)ρ, 251
    mk-Character-number-literal (-) → *t-character-number-literal* (*val*)ρ 252
  end

Dynamic Semantics

m-string-literal : *String-literal* → *Environment* $\xrightarrow{o}$ *Value*

m-string-literal (*val*)ρ ≜
  cases *val*:
    mk-Character-string (-) → *m-character-string*(*val*) ρ, 252
    mk-Character-number-literal (-) → *m-character-number-literal*(*val*) ρ 253
  end

6.8.7.3.1 Quoted Strings

A quoted string denotes a constant value of the string literal type which is a sequence of zero or more characters.

Concrete Syntax

NOTE 1 — The syntax for quoted strings is given in the Lexis — see 5.4.3.

Abstract Syntax

types

Character-string :: *cl* : *Char**

Static Semantics

The type of a quoted string shall be the string literal type with length given by the number of quoted characters — see 6.10.2.7.

wf-character-string : *Character-string* → *Environment* → 𝔹

wf-character-string (*mk-Character-string* (-))ρ ≜
  true

t-character-string : *Character-string* → *Environment* → *Typed*

t-character-string (*cl*)ρ ≜
  mk-String-type (len *cl*)

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Dynamic Semantics

The value of a quoted string shall be the string literal value denoted by the characters, if any, between the pair of single quotes or double quotes.

NOTE 2 — A quoted string containing an empty sequence of characters is called the 'empty string literal' and is a denotation both for the empty string and the character that is the string terminator — see 6.8.7.3.3.

m-character-string : *Character-string* → *Environment* $\xrightarrow{o}$ *String-value*

m-character-string (*mk-Character-string* (*cl*))ρ $\triangleq$
  return *cl*

6.8.7.3.2 Character Number Literals

A character number literal denotes a constant value of the string literal type of length 1.

Concrete Syntax

NOTE 1 — The syntax for character number literals is given in the Lexis — see 5.4.3.

Abstract Syntax

types

Character-number-literal :: *val* : **N**

Static Semantics

The value of the sequence of octal digits that precede the 'C' shall be the ordinal number of a value of the character type.

The type of a character number literal shall be the string literal type of length one.

CHANGE — In *Programming in Modula-2* a character number literal is of the character type.

NOTES

2 The character type is expression-compatible and assignment-compatible with the string literal type of length one.

3 Concrete string types are assignment-compatible with the string literal type of length one.

wf-character-number-literal : *Character-number-literal* → *Environment* → $\mathbb{B}$

wf-character-number-literal (*mk-Character-number-literal* (*val*))ρ $\triangleq$
  $\exists\, ch \in characters \cdot collation\text{-}map(ch) = val$

t-character-number-literal : *Character-number-literal* → *Environment* → *Expression-typed*

t-character-number-literal (*mk-Character-number-literal* (-))ρ $\triangleq$
  mk-String-type (1)

Dynamic Semantics

The value of a character number literal shall be the string literal value denoted by the character whose ordinal value is given by the value of the sequence of octal digits that precede the 'C'.

NOTE 4 — The ordinal number of a character need not be the same as the internal code used to represent the character — see 6.10.2.5

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

m-character-number-literal : *Character-number-literal* → *Environment* $\xrightarrow{o}$ *String-value*

m-character-number-literal (*mk-Character-number-literal* (*val*))ρ $\triangleq$
  let *ch* ∈ *characters* be st *collation-map*(*ch*) = *val* in
  return [*ch*]

6.8.7.3.3 Character Literals

In contexts where a value of an ordinal type is required, values of the string literal type of length 0 or length 1 are interpreted as values of the character type.

Auxiliaries

to-char : *String-value* → *Char*

to-char (*sv*) $\triangleq$
  cases *sv* :
    [] → END-OF-STRING-CHAR,
    [*x*]→ *x*
  end
pre len *sv* ≤ 1

evaluate-to-ordinal : *Expression* → *Environment* $\xrightarrow{o}$ *Value*

evaluate-to-ordinal (*expr*)ρ $\triangleq$
  def *val* = *m-expression*(*expr*) ρ in
  let *res* = if *val* : *String-value*
              then *to-char* (*val*)
              else *val* in
  return *res*

6.8.8 Constant Expressions

A constant expression is an expression whose constituents are constants. It can therefore be evaluated by a mere textual scan of the program without having to execute it.

Concrete Syntax

constant expression = expression ;

Abstract Syntax

annotations The abstract syntax for *constant-expression* is the same as the abstract syntax for *expression*.

Static Semantics

An infix expression is constant if and only if its two operands are both constant expressions.

NOTE — The constant expression 'TRUE OR (1/0=1)' is permitted and has the value *true* since the right operand is not evaluated.

A prefix expression is constant if and only if its operand is a constant expression.

A constant expression shall not contain a function call, unless the actual parameters of the function call are constant expressions, and the function designator of the function call denotes a standard function, or is one of the functions CAST, MAKEADR, ROTATE, SHIFT and TSIZE from the module SYSTEM.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

A constant expression shall not contain a reference to a variable, unless that variable is contained in an actual parameter to the standard function SIZE.

A constant expression shall not contain a value constructor, unless the expressions of the value constructor are constant expressions.

The value of a constant expression that is of the $\mathbb{Z}$-type shall be within the range of values defined by the $\mathbb{Z}$-type.

The value of a constant expression that is of the $\mathbb{R}$-type shall be within the range of values defined by the $\mathbb{R}$-type.

CLARIFICATION — A constant expression may contain values of procedure types, or structured values whose components are values of procedure types.

is-constant-expression : *Expression* → *Environment* → $\mathbb{B}$

is-constant-expression (*expr*)ρ $\triangleq$
(*is-Infix-expression*(*expr*) → *is-constant-infix-expression* (*expr*)ρ, 254
is-Prefix-expression(*expr*) → *is-constant-prefix-expression* (*expr*)ρ, 254
is-Value-designator(*expr*) → *is-constant-value-designator* (*expr*)ρ, 254
is-Function-call(*expr*) → *is-constant-function-call* (*expr*)ρ, 255
is-Value-constructor(*expr*) → *is-constant-value-constructor* (*expr*)ρ, 255
is-Constant-literal(*expr*) → true)

annotations The function checks whether an expression is a constant expression. Any constant literal is a constant expression.

is-constant-infix-expression : *Infix-expression* → *Environment* → $\mathbb{B}$

is-constant-infix-expression (*mk-Infix-expression* (*left*, -, *right*))ρ $\triangleq$
is-constant-expression (*left*)ρ ∧ 254
is-constant-expression (*right*)ρ 254

annotations Check that both expression components are constant expressions.

is-constant-prefix-expression : *Prefix-expression* → *Environment* → $\mathbb{B}$

is-constant-prefix-expression (*mk-Prefix-expression* (-, *expr*))ρ $\triangleq$
is-constant-expression (*expr*)ρ 254

annotations Check that the expression component is a constant expression.

is-constant-value-designator : *Value-designator* → *Environment* → $\mathbb{B}$

is-constant-value-designator (*vd*)ρ $\triangleq$
cases *vd* :
mk-Entire-value (-) → *is-constant-entire-value* (*vd*)ρ, 254
mk-Indexed-value (-, -) → *is-constant-indexed-value* (*vd*)ρ, 254
mk-Selected-value (-, -) → *is-constant-selected-value* (*vd*)ρ, 255
mk-Dereferenced-value (-) → *is-constant-dereferenced-value* (*vd*)ρ 255
end

is-constant-entire-value : *Entire-value* → *Environment* → $\mathbb{B}$

is-constant-entire-value (*mk-Entire-value* (*qid*))ρ $\triangleq$
is-constant (*qid*)ρ ∨ 313
is-procedure (*qid*)ρ 316

annotations Check that the qualified identifier denotes a constant or a proper procedure or a function procedure.

is-constant-indexed-value : *Indexed-value* → *Environment* → $\mathbb{B}$

is-constant-indexed-value (*mk-Indexed-value* (*desig*, *expr*))ρ $\triangleq$
is-constant-value-designator (*desig*)ρ ∧ 254
is-constant-expression (*expr*)ρ 254

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations Check that the designator denotes a constant and that the index expression is a constant expression.

is-constant-selected-value : *Selected-value* → *Environment* → 𝔹

is-constant-selected-value (*mk-Selected-value* (*desig*, -))ρ ≜
  is-constant-value-designator (*desig*)ρ 254

annotations Check that the designator denotes a constant.

is-constant-dereferenced-value : *Dereferenced-value* → *Environment* → 𝔹

is-constant-dereferenced-value (*mk-Dereferenced-value* (-))ρ ≜
  false

annotations A dereferenced value cannot be a constant expression.

is-constant-function-call : *Function-call* → *Environment* → 𝔹

is-constant-function-call (*mk-Function-call* (*fd*, *args*))ρ ≜
  is-constant-function-designator (*fd*) ∧ 255
  ∀ *arg* ∈ elems *args* · *is-constant-actual-parameter* (*arg*)ρ 255
    ∨
  is-Size-designator(*fd*)

annotations Check that the function call is to a standard function or a system function that is defined to be a constant function.

is-constant-function-designator : *Value-designator* → 𝔹

is-constant-function-designator (*fd*) ≜
  (*is-Standard-function-designator*(*fd*) → *is-constant-standard-function* (*fd*), 255
   is-System-function-designator(*fd*) → *is-constant-system-function* (*fd*), 255
   others → false)

is-constant-standard-function : *Standard-function-designator* → 𝔹

is-constant-standard-function (*fd*) ≜
  fd ∈ {ABS-DESIGNATOR, CAP-DESIGNATOR, CHR-DESIGNATOR,
    CMPLX-DESIGNATOR, FLOAT-DESIGNATOR, HIGH-DESIGNATOR,
    IM-DESIGNATOR, INT-DESIGNATOR, LENGTH-DESIGNATOR,
    LFLOAT-DESIGNATOR, MAX-DESIGNATOR, MIN-DESIGNATOR,
    ODD-DESIGNATOR, ORD-DESIGNATOR, RE-DESIGNATOR,
    TRUNC-DESIGNATOR, VAL-DESIGNATOR} ∧
  fd ∉ {SIZE-DESIGNATOR}

is-constant-system-function : *System-function-designator* → 𝔹

is-constant-system-function (*fd*) ≜
  fd ∈ {CAST-DESIGNATOR, MAKEADR-DESIGNATOR, ROTATE-DESIGNATOR,
    SHIFT-DESIGNATOR, TSIZE-DESIGNATOR} ∧
  fd ∉ {ADDADR-DESIGNATOR, ADR-DESIGNATOR, DIFADR-DESIGNATOR,
    SUBADR-DESIGNATOR}

is-constant-actual-parameter : *Actual-parameter* → *Environment* → 𝔹

is-constant-actual-parameter (*ap*)ρ ≜
  (*is-Expression*(*ap*) → *is-constant-expression* (*ap*)ρ, 254
   is-Typed-expression(*ap*) → let *mk-Typed-expression* (*expr*, -) = *ap* in
     is-constant-expression (*expr*)ρ, 254
   is-Type-parameter(*ap*) → true)

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

is-constant-value-constructor : *Value-constructor* → *Environment* → 𝔹

is-constant-value-constructor (*vc*)ρ ≜
  cases *vc* :
    mk-Array-constructor (-, -) → *is-constant-array-constructor* (*vc*)ρ, 256
    mk-Record-constructor (-, -) → *is-constant-record-constructor* (*vc*)ρ, 256
    mk-Set-constructor (-, -) → *is-constant-set-constructor* (*vc*)ρ 256
  end

is-constant-array-constructor : *Array-constructor* → *Environment* → 𝔹

is-constant-array-constructor (*mk-Array-constructor* (-, *def*))ρ ≜
  is-constant-array-definition (*def*)ρ 256

is-constant-array-definition : *Array-constructed-value* → *Environment* → 𝔹

is-constant-array-definition (*mk-Array-constructed-value* (*vals*))ρ ≜
  ∀ *val* ∈ elems *vals* · *is-constant-repeated-element* (*val*)ρ 256

is-constant-repeated-element : *Repeated-element* → *Environment* → 𝔹

is-constant-repeated-element (*mk-Repeated-element* (*val*, *by*))ρ ≜
  is-constant-element (*val*)ρ 256

is-constant-element : *Element* → *Environment* → 𝔹

is-constant-element (*el*)ρ ≜
  (*is-Expression*(*el*) → *is-constant-expression* (*el*)ρ, 254
   is-Array-constructed-value(*el*) → *is-constant-array-definition* (*el*)ρ, 256
   is-Record-constructed-value(*el*) → *is-constant-record-definition* (*el*)ρ, 256
   is-Set-constructed-value(*el*) → *is-constant-set-definition* (*el*)ρ) 256

annotations Check that the array definition is a constant expression.

Check that each of the repeated element components of an array definition is a constant expression.

Check that element and repetition factor are constant expressions.

is-constant-record-constructor : *Record-constructor* → *Environment* → 𝔹

is-constant-record-constructor (*mk-Record-constructor* (-, *def*))ρ ≜
  is-constant-record-definition (*def*)ρ 256

is-constant-record-definition : *Record-constructed-value* → *Environment* → 𝔹

is-constant-record-definition (*mk-Record-constructed-value* (*vals*))ρ ≜
  ∀ *val* ∈ elems *vals* · *is-constant-element* (*val*)ρ 256

annotations Check that a record constructor is a constant expression by checking that the record definition component is a constant expression. A record definition is a constant expression if each of the elements is a constant expression.

is-constant-set-constructor : *Set-constructor* → *Environment* → 𝔹

is-constant-set-constructor (*mk-Set-constructor* (-, *def*))ρ ≜
  is-constant-set-definition (*def*)ρ 256

is-constant-set-definition : *Set-constructed-value* → *Environment* → 𝔹

is-constant-set-definition (*def*)ρ ≜
  ∀ *member* ∈ *def* · *is-constant-member* (*member*)ρ 256

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

is-constant-member : *Member* → *Environment* → 𝔹

is-constant-member (*mem*)ρ ≜
  (*is-Singleton*(*mem*) → *is-constant-singleton* (*mem*)ρ, 257
   is-Interval(*mem*) → *is-constant-interval* (*mem*)ρ) 257

is-constant-singleton : *Singleton* → *Environment* → 𝔹

is-constant-singleton (*mk-Singleton* (*expr*))ρ ≜
  is-constant-expression (*expr*)ρ 254

is-constant-interval : *Interval* → *Environment* → 𝔹

is-constant-interval (*mk-Interval* (*min*, *max*))ρ ≜
  is-constant-expression (*min*)ρ ∧ 254
  is-constant-expression (*max*)ρ 254

annotations Check that the set definition is a constant expression by checking that each of the members of the set is a constant expression. A singleton member is constant if it is a constant expression, and an interval is constant if each of its component expressions are constant expressions.

6.8.8.1 Constant Expression Evaluation

evaluate-constant-expression : *Expression* → *Environment* → *Value*

evaluate-constant-expression (*expr*)ρ ≜
  let ρ_{cont} = *c-exit*(NOT-DEFINED) nil ρ in 345
  let *mk-* (-, *result*) = *m-expression*(*expr*) (ρ_{cont}) σ_{empty} in 190
  result

annotations As a constant expression can be evaluated with just a textual scan, the evaluation of a constant expression is defined to be the meaning of an expression with an empty state.

evaluate-constant-interval : *Interval* → *Environment* → *Value*-set

evaluate-constant-interval (*intv*)ρ ≜
  let ρ_{cont} = *c-exit*(NOT-DEFINED) nil ρ in 345
  let *mk-* (-, *result*) = *m-interval*(*intv*) (ρ_{cont}) σ_{empty} in 246
  result

evaluate-constant-set-definition : *Set-constructed-value* → *Environment* → *Value*-set

evaluate-constant-set-definition (*sdef*)ρ ≜
  let ρ_{cont} = *c-exit*(NOT-DEFINED) nil ρ in 345
  let *mk-* (-, *result*) = *m-set-definition*(*sdef*) (ρ_{cont}) σ_{empty} in 244
  result

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.9 Parameter Compatibility and Argument Binding

If a proper procedure or function procedure is of a structure that has formal parameters, a call of that procedure has a corresponding actual parameter list. The actual parameters and the formal parameters must match. The execution of a proper procedure call or function procedure call involves the evaluation of any actual parameters to give corresponding arguments, the binding of the results of this evaluation to the corresponding formal parameters, and the execution of the actions associated with the procedure designator.

In the case of a declared procedure, the correspondence between actual parameters and formal parameters is established by the positions of the parameters in the lists of actual and formal parameters respectively; the actions are as specified by the block of the procedure declaration.

In the case of a call of a predefined procedure, parameter correspondence, argument binding, and procedure actions are as specified for that predefined procedure.

6.9.1 Actual Parameters

Actual parameters of declared procedures are either variable designators (if the corresponding formal parameter is a variable parameter), or expressions (if the corresponding formal parameter is a value parameter). In the case of calls of predefined procedures, there are special cases of actual parameters that are type identifiers.

Concrete Syntax

actual parameters = left parenthesis, [actual parameter list], right parenthesis ;

actual parameter list = actual parameter, { comma, actual parameter } ;

actual parameter = variable designator | expression | type parameter ;

Abstract Syntax

types

Actual-parameters = *Actual-parameter**;

Actual-parameter = *Typed-variable-designator* | *Typed-expression* | *Type-parameter*

Domains

types

Arguments = *Argument**;

Argument = *Typed-variable* | *Typed-value* | *Typed-type*

Static Semantics

An actual parameter shall be a variable designator, an expression, or a type parameter.

wf-actual-parameters : *Actual-parameters* → *Environment* → $\mathbb{B}$

wf-actual-parameters (*aps*)ρ $\triangleq$
  $\forall$ *ap* $\in$ elems *aps* · *wf-actual-parameter* (*ap*)ρ 259

annotations Check that each of the actual parameters is well-formed.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wf-actual-parameter : *Actual-parameter* → *Environment* → $\mathbb{B}$

wf-actual-parameter (*ap*)ρ $\triangleq$
  (*is-Typed-variable-designator*(*ap*) → *wf-typed-variable-designator* (*ap*)ρ, 259
   is-Typed-expression(*ap*) → *wf-typed-expression* (*ap*)ρ, 260
   is-Type-parameter(*ap*) → *wf-type-parameter* (*ap*)ρ) 261

Dynamic Semantics

Each of the actual parameters shall be evaluated. The order of evaluation, the order of accessing and the order of binding of the actual parameters shall be implementation-dependent.

m-actual-parameters : *Actual-parameters* → *Environment* $\xrightarrow{o}$ *Arguments*

m-actual-parameters ((*aps*)ρ) $\triangleq$
  if *aps* = [ ]
  then return [ ]
  else def *mhd* = *m-actual-parameter*(hd *aps*) ρ, 259
       mtl = *m-actual-parameters*(tl *aps*) ρ in 259
    return [*mhd*] $\curvearrowright$ *mtl*

annotations The above evaluation order for the the actual parameters is one of those permitted — see 4.1.

m-actual-parameter : *Actual-parameter* → *Environment* $\xrightarrow{o}$ *Argument*

m-actual-parameter (*ap*)ρ $\triangleq$
  (*is-Typed-variable-designator*(*ap*) → *m-typed-variable-designator*(*ap*) ρ, 260
   is-Typed-expression(*ap*) → *m-typed-expression*(*ap*) ρ, 260
   is-Type-parameter(*ap*) → *m-type-parameter*(*ap*) ρ) 261

6.9.1.1 Variable Actual Parameters

Actual parameters are variable designators if the corresponding formal parameter is a variable parameter.

Abstract Syntax

types

Typed-variable-designator :: *desig* : *Variable-designator*
  type : *Typed*

annotations Type information is supplied with argument variables for the effect of binding with system storage formal parameters to be specified and for calls of the predefined procedures `DEC`, `DISPOSE`, `EXCL`, `INC`, `INCL`, `NEW`, `SIZE`, and `SYSTEM.ADR`.

Domains

types

Typed-variable :: *var* : *Variable*
  type : *Variable-typed*

Static Semantics

wf-typed-variable-designator : *Typed-variable-designator* → *Environment* → $\mathbb{B}$

wf-typed-variable-designator (*mk-Typed-variable-designator* (*desig*, *type*))ρ $\triangleq$
  wf-variable-designator (*desig*)ρ $\wedge$ 182
  type = *t-variable-designator* (*desig*)ρ 182

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Dynamic Semantics

m-typed-variable-designator : *Typed-variable-designator* → *Environment* $\xrightarrow{o}$ *Typed-variable*

m-typed-variable-designator (*mk-Typed-variable-designator* (*desig*, *type*))ρ $\triangleq$
  def *var* = *m-variable-designator*(*desig*) ρ in
  return *mk-Typed-variable* (*var*, *type*)

182

6.9.1.2 Expression Actual Parameters

Actual parameters are expressions if the corresponding formal parameter is a value parameter.

Abstract Syntax

types

Typed-expression :: *expr* : *Expression*
  type : *Expression-typed*

annotations Type information is supplied with argument values for the effect of binding with system storage formal parameters to be specified and for calls of the predefined procedures `ABS`, `CMPLX`, `IM`, `RE`, `SYSTEM.ROTATE` and `SYSTEM.SHIFT`.

Domains

types

Typed-value :: *val* : *Value*
  type : *Expression-typed*

Static Semantics

wf-typed-expression : *Typed-expression* → *Environment* → $\mathbb{B}$

wf-typed-expression (*mk-Typed-expression* (*expr*, *type*))ρ $\triangleq$
  wf-expression (*expr*)ρ ∧ 190
  type = *t-expression* (*expr*)ρ 190

Dynamic Semantics

m-typed-expression : *Typed-expression* → *Environment* $\xrightarrow{o}$ *Typed-value*

m-typed-expression (*mk-Typed-expression* (*expr*, *type*))ρ $\triangleq$
  def *val* = *m-expression*(*expr*) ρ in
  return *mk-Typed-value* (*val*, *type*)

190

6.9.1.3 Type Parameters

Some predefined function procedures (`MAX`, `MIN`, `SIZE` and `VAL`) and the function procedures `SYSTEM.CAST` and `SYSTEM.TSIZE` take a type identifier as an actual parameter.

Concrete Syntax

type parameter = type identifier ;

Abstract Syntax

types

Type-parameter :: *type* : *Qualident*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Domains

types

Typed-type :: *type* : *Typed*

Static Semantics

wf-type-parameter : *Type-parameter* → *Environment* → 𝔹

wf-type-parameter (*mk-Type-parameter* (*qid*))ρ ≜
  wf-qualident (*qid*)ρ ∧ 53
  is-type (*qid*)ρ 314

Dynamic Semantics

m-type-parameter : *Type-parameter* → *Environment* $\xrightarrow{o}$ *Typed-type*

m-type-parameter (*mk-Type-parameter* (*qid*))ρ ≜
  let *typed* = *type-of* (*qid*)ρ in 314
  return *mk-Typed-type* (*typed*)

6.9.2 Parameter Matching

Calls of a declared procedure must supply actual parameters that match the formal parameters given in the procedure heading of that declared procedure.

The requirements for parameter matching for a call of a predefined proper procedure, a predefined function procedure, or a proper procedure or a function procedure exported from a system module are given separately for each proper procedure and function procedure in 6.10.3, 6.10.4 and in section 7.

Static Semantics

The formal parameters of a proper procedure or a function procedure shall match the actual parameters specified in any call of that proper procedure or function procedure. The formal parameters match the actual parameters if and only if the number of formal parameters is equal to the number of actual parameters, and each formal parameter is parameter-compatible with the corresponding actual parameter — see 6.9.3. This correspondence shall be established by the positions of the parameters in the lists of formal and actual parameters respectively.

is-parameters-match : *Formal-parameters-typed* × *Actual-parameters* → *Environment* → 𝔹

is-parameters-match (*fpl*, *apl*)ρ ≜
  len *fpl* = len *apl* ∧
  ∀ *i* ∈ inds *fpl* · *is-parameter-compatible* (*fpl*(*i*), *apl*(*i*))ρ 262

annotations Check that the number of formal parameters is equal to the number of actual parameters, and that each of the formal parameters is parameter-compatible with the actual parameter.

6.9.3 Parameter Compatibility

There are two cases of parameter-compatibility, depending on whether the formal parameter is a value parameter or a variable parameter. There are special rules common to both cases if the formal type is a system storage formal type — see 6.9.3.3.

Static Semantics

A formal parameter shall be parameter-compatible with an actual parameter if either of the following statements is true:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

a) The formal parameter is a value parameter, the actual parameter is an expression, and the formal type of the formal parameter is value parameter compatible with the type of the expression.

b) The formal parameter is a variable parameter, the actual parameter is a variable designator, and the formal type of the formal parameter is variable parameter compatible with the type of the expression.

is-parameter-compatible : *Formal-parameter-typed* × *Actual-parameter* → *Environment* → $\mathbb{B}$

is-parameter-compatible $(fp, ap)\rho \triangleq$

cases fp :

mk-Value-formal-typed (ft) → *is-Typed-expression*(ap) ∧
is-value-parameter-compatible $(ft, ap.type)\rho$, 262

mk-Variable-formal-typed (ft) → *is-Typed-variable-designator*(ap) ∧
is-variable-parameter-compatible $(ft, ap.type)\rho$ 263

end

6.9.3.1 Value Parameter Compatibility

Static Semantics

The formal type of a value parameter shall be compatible with the type of an actual expression if any of the following statements is true:

a) The formal type is a system storage formal type, or a one-dimensional open array of a system storage formal type, and is compatible with the actual type as the type of a system storage formal parameter — see 6.9.3.3.

b) The formal type is an open array, the actual type is the string literal type of any length, and the component type of the formal type is the character type.

c) The formal type is an open array (other than a one-dimensional open array of a system storage formal type), the actual type is an array type, and the component type of the formal type is compatible with the component type of the actual type as an open array component type.

d) The formal type is a single identified type that is assignment-compatible with the actual type.

The component type of an open array value parameter shall be compatible with the component type of an actual array if the component types are system storage parameter compatible, are of the identical identified type, or themselves have component types that are compatible in one or other of these ways.

EXAMPLE — Value parameter compatibility.

Given the declaration

```
TYPE Colour = (red, yellow, green);
```

The following formal types are compatible, as the types of value parameters, with the given actual types.

Formal types denoted by	Actual types denoted by
`CARDINAL`	`INTEGER`
`ARRAY OF ARRAY OF ARRAY OF CARDINAL`	`ARRAY [-7..-2] OF ARRAY BOOLEAN OF ARRAY Colour OF CARDINAL`
`ARRAY OF ARRAY OF SYSTEM.LOC`	`ARRAY Colour OF REAL`
`ADDRESS`	`POINTER TO LONGREAL`

CLARIFICATION — A formal open array parameter with the component type `T` is not parameter-compatible with an actual parameter of the type `T`; e.g. a formal parameter of the open array structure `ARRAY OF CHAR` is not compatible with an actual parameter of the character type. Note, however, that a string literal such as `"!"` is of the string literal type of length 1, with which the character type is assignment-compatible, and can therefore be passed as a parameter in this case.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

is-value-parameter-compatible : *Variable-typed* × *Expression-typed* → *Environment* → 𝔹

is-value-parameter-compatible $(ft, et)\rho \triangleq$
  (*is-system-storage-parameter* $(ft)\rho$ → *is-system-storage-parameter-compatible* $(ft, et)\rho$, 264 264
  is-Open-array-typed(ft) ∧ *is-String-type*(et) → let *mk-Open-array-typed* $(otype) = ft$ in
    is-character-type $(otype)$, 323
  is-Open-array-typed(ft) → *is-open-array-value-parameter-compatible* $(ft, et)\rho$, 263
  is-Typed(ft) → *is-assignment-compatible* $(ft, et)\rho$) 123

is-open-array-value-parameter-compatible : *Variable-typed* × *Expression-typed* → 𝔹

is-open-array-value-parameter-compatible $(ft, et)\rho \triangleq$
  let *mk-Open-array-typed* $(otype) = ft$ in
  is-array-type $(et)\rho$ ∧ 321
  let *ctype* = *component-type-of* $(et)\rho$ in 325
  is-open-array-component-compatible $(otype, ctype)\rho$ 263

is-open-array-component-compatible : *Variable-typed* × *Expression-typed* → 𝔹

is-open-array-component-compatible $(ft, et)\rho \triangleq$
  (*is-system-storage-parameter* $(ft)\rho$ → *is-system-storage-parameter-compatible* $(ft, et)\rho$, 264 264
  is-Open-array-typed(ft) → *is-open-array-value-parameter-compatible* $(ft, et)\rho$, 263
  is-Typed(ft) → $ft = et$)

6.9.3.2 Variable Parameter Compatibility

Static Semantics

The formal type of a variable parameter shall be compatible with the type of an actual variable if any of the following statements is true:

a) The formal type is the address type and the actual type is any pointer type.

b) The formal type is a system storage formal type, or a one-dimensional open array of a system storage formal type, and is compatible with the actual type as the type of a system storage formal parameter — see 6.9.3.3.

c) The formal type is an open array (other than a one-dimensional open array of a system storage formal type), the actual type is an array type, and the component type of the formal type is compatible, as the formal type of a variable parameter, with the component type of the actual type.

d) The formal type is a single identified type that is identical to the actual type.

EXAMPLE — Variable parameter compatibility.

Given the declaration

```
TYPE Colour = (red, yellow, green);
```

The following formal types are compatible, as the types of variable parameters, with the given actual types.

Formal types denoted by	Actual types denoted by
`CARDINAL`	`CARDINAL`
`ARRAY OF ARRAY OF ARRAY OF CARDINAL`	`ARRAY [-7..-2] OF ARRAY BOOLEAN OF ARRAY Colour OF CARDINAL`
`ARRAY OF ARRAY OF SYSTEM.LOC`	`ARRAY Colour OF REAL`
`ADDRESS`	`POINTER TO LONGREAL`

is-variable-parameter-compatible : *Variable-typed* × *Variable-typed* → 𝔹

is-variable-parameter-compatible $(ft, vt)\rho \triangleq$
  (*is-address-type* $(ft)\rho$ → *is-pointer-type* $(vt)\rho$, 324 322
  is-system-storage-parameter $(ft)\rho$ → *is-system-storage-parameter-compatible* $(ft, vt)\rho$, 264 264
  is-Open-array-typed(ft) → *is-open-array-variable-parameter-compatible* $(ft, vt)\rho$, 264
  is-Typed(ft) → $ft = vt$)

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

is-open-array-variable-parameter-compatible : *Variable-typed* × *Variable-typed* → 𝔹

is-open-array-variable-parameter-compatible $(ft, vt)\rho \triangleq$
  let *mk-Open-array-typed* $(otype) = ft$ in
  is-array-type $(vt)\rho \wedge$ 321
  let *ctype* = *component-type-of* (vt) in 325
  is-variable-parameter-compatible $(otype, ctype)\rho$ 263

6.9.3.3 System Storage Parameter Compatibility

There are special parameter compatibility rules for formal types that are system storage formal types or open arrays of a system storage formal type.

A formal type is a system storage formal type if it is a system storage type or a zero-based array of a system storage type.

The system storage types are the location type `SYSTEM.LOC`, the word type `SYSTEM.WORD`, and the byte type `SYSTEM.BYTE` — see 7.1.3.1. Not all implementations are required to provide the byte type — see 4.4.

Static Semantics

A formal type shall be system storage parameter compatible with the type of an actual parameter, other than a numeric literal, if any of the following statements is true:

a) The formal type `F` is a system storage formal type, and the actual parameter is of any data type `T` such that `SIZE(T)` is equal to `SIZE(F)`.

b) The formal type is a one-dimensional open array of a system storage formal type `F`, and the actual parameter is of any data type `T` such that `SIZE(T)` is a multiple of `SIZE(F)`.

CLARIFICATION — A system storage formal type is not system storage parameter compatible with numeric literal types since such types do not have a defined storage size.

CHANGE — In *Programming in Modula-2*, a zero-based array of a system storage type is not treated as a system storage formal type. This case has been added for consistency with the definition of `SYSTEM.WORD` which has the structure `ARRAY [0..SYSTEM.LOCSPERWORD] OF SYSTEM.LOC`.

NOTE — It follows from the specification of value parameter compatibility and variable parameter compatibility that a formal parameter that is an n dimensional open array of a system-storage formal type, where $n > 1$, is parameter compatible with an actual parameter that is an $n-1$ dimensional array of any component type. The upper bounds of the first $n-1$ dimensions of the formal variable is determined by the index types of the first $n-1$ dimensions of the actual parameter; the upper bound of the remaining dimension is determined by rule 6.9.3.3b.

is-system-storage-parameter-compatible : *Variable-typed* × *Expression-typed* → *Environment* → 𝔹

is-system-storage-parameter-compatible $(ftype, atype)\rho \triangleq$
  $atype \notin \{\mathbb{Z}\text{-TYPE}, \mathbb{R}\text{-TYPE}, \mathbb{C}\text{-TYPE}\} \wedge$
  (*is-Typed*$(ftype)$ → *tsize* $(ftype)\rho = tsize\,(atype)\rho$, 366 366
   is-Open-array-typed$(ftype)$ → let *mk-Open-array-typed* $(type) = ftype$ in
    $\exists\, n : \mathbb{N} \cdot tsize\,(type)\rho \times n = tsize\,(atype)\rho$) 366 366

Auxiliaries

is-system-storage-parameter : *Variable-typed* → *Environment* → 𝔹

is-system-storage-parameter $(vtype)\rho \triangleq$
  (*is-Typed*$(vtype)$ → *is-system-storage-formal-type* $(vtype)\rho$, 265
   is-Open-array-typed$(vtype)$ → let *mk-Open-array-typed* $(type) = vtype$ in
    is-system-storage-formal-type $(type)\rho$) 265

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations The formal type is a system storage formal type, or is an open array whose components are a system storage formal type.

is-system-storage-formal-type : *Typed* → *Environment* → 𝔹

is-system-storage-formal-type (*type*)ρ ≙
  is-system-storage-type (*type*)ρ 324
  ∨
  is-zero-array-type (*type*)ρ ∧ 328
  is-system-storage-type (*component-type-of* (*type*)ρ) 324 325

annotations A system storage formal type is either a storage type exported from the module **SYSTEM**, or an array whose index type is a whole number type with a lower bound of zero, and whose component type is a storage type exported from the module **SYSTEM**.

6.9.4 Argument Binding

On a call of a declared procedure, the arguments resulting from the evaluation of the actual parameters are bound to the formal parameters given in the procedure heading of that declared procedure. The form of this binding is different for value formal parameters and for variable formal parameters, and takes account of both open array formal parameters and formal parameters constucted from system storage types.

The binding of arguments for a call of a predefined proper procedure, a predefined function procedure, or a proper procedure or a function procedure exported from a system module is given explicitly for each proper procedure and function procedure in 6.10.3, 6.10.4 and in section 7.

Dynamic Semantics

The arguments arising from the evaluation of each actual parameter shall be bound to the corresponding formal parameter. This correspondence shall be established by the positions of the parameters in the lists of actual and formal parameters respectively.

The order of evaluation, the order of accessing and the order of binding of the actual parameters shall be implementation-dependent.

NOTES

1 The binding of the arguments might involve storage allocation and so might lead to the raising of a workspace overflow exception.

2 The binding of the arguments involves the assignment of the arguments to storage allocated for value parameters and so might lead to the raising of a range exception.

3 Exceptions raised during argument binding will be handled by any exception handler that is currently selected for the caller of the procedure for which the arguments are being bound.

4 Since the evaluation of the actual parameters may have side-effects, and the order of evaluation, the order of accessing and the order of binding of the actual parameters is implementation-dependent, this part of ISO/IEC 10514 gives no meaning to a program that relies upon a definition of which actual parameters have been evaluated if an exception is raised during the binding of the arguments.

5 A workspace overflow exception resulting from the elaboration of the declarations of the called procedure will also be handled by any exception handler that is currently selected for the caller of the procedure — see 6.5.8.

bind-arguments : *Formal-parameter-list* × *Arguments* → *Environment* $\xrightarrow{o}$ *Environment*

bind-arguments (*parms*, *args*)ρ ≙
  if *parms* = [ ]
  then return ρ
  else let i ∈ inds *parms* in
    let *remp* = *del* (*parms*, i), 615
      rema = *del* (*args*, i) in 615
    def $penv_i$ = *bind-argument*(*parms* (i), *args* (i)) ρ, 266

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

penv = *bind-arguments*(*remp*, *rema*) ρ in 265
return *overwrite-var-environment* (*penv*$_i$ $\cup$ *penv*)ρ 316

annotations The above binding order for the the arguments is one of those permitted — see 4.1.

bind-argument : *Formal-parameter* × *Argument* → *Environment* $\xrightarrow{o}$ *Environment*

bind-argument (*parm*, *arg*)ρ $\triangleq$
cases *parm*:
mk-Value-parameter-specification (*id*, *ftype*) → *bind-argument-to-value-parameter*(*id*, *ftype*, *arg*) ρ, 267
mk-Variable-parameter-specification (*id*, *ftype*) → *bind-argument-to-variable-parameter*(*id*, *ftype*, *arg*) ρ 268
end

annotations The argument is bound to the parameter according to whether the formal parameter is a value parameter or a variable parameter.

6.9.4.1 Argument Binding to Value Parameters

If the formal parameter is a value parameter, and the formal type is not a system storage formal type or an open array of a system storage formal type (see 7.1.3.1 and 6.9.3.3), then, on a call of the procedure, the corresponding argument shall be assigned to a formal variable allocated for the call. During the execution of the procedure block, the formal parameter identifier shall designate the formal variable. The structure of the formal variable shall be determined by the formal type of the formal parameter, and by the structure of the actual value given in the corresponding argument of the call.

NOTE 1 — Since a formal parameter identifier of a value parameter designates a formal variable that is allocated for the call, the formal variable is a local variable of the block that is additional to any local variables declared in the procedure block.

If a formal type is an open array formal type, then the formal variable shall be an array variable; the index type of the formal variable shall have the structure of a subrange type whose lower bound is 0 and whose upper bound is one less than the number of components of the corresponding actual value. The component type of the formal variable shall be determined recursively by the open array component type of the formal type, and by the structure of components of the corresponding actual value.

NOTE 2 — The upper bound of the index type may be ascertained using the predefined function procedure `HIGH` — see 6.10.4.7.

If a formal type is a single identified type, the allocated variable shall be a single variable of that identified type; storage shall be allocated for the variable and the corresponding argument value shall be assigned to the allocated variable.

EXAMPLE — Binding of an open array value parameter.

Given:

```
TYPE colour = (red, yellow, green);

VAR a: ARRAY [-7..-2] OF ARRAY BOOLEAN
                  OF ARRAY colour OF CARDINAL;

PROCEDURE
  p(f: ARRAY OF ARRAY OF ARRAY OF CARDINAL);
```

then, during activation by '`p(a)`', within the body of `p` the structure of `f` will correspond to the local declaration

```
VAR f: ARRAY [0..5] OF ARRAY [0..1]
                    OF ARRAY [0..2] OF CARDINAL;
```

and the initial value of the variable designated by `f[0][0][0]` will be the value of `a[-7][false][red]`.

System storage formal types

If the formal parameter is a value parameter, and the formal type is a system storage formal type, or an open array of a system storage formal type, the effect shall be for the value of the argument to be cast to a target type constructed from one or more storage locations, and for the the formal parameter to be bound to the resulting value. The target type shall be determined as follows:

a) If the formal type is a system storage formal type, the target type shall be that system storage formal type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

b) If the formal type is an open array of a system storage formal type F, and the actual parameter is of a data type T, the target type shall be of the structure `ARRAY [0..n-1] OF F` where `n = SIZE(T)/SIZE(F)`.

NOTE 3 — A formal type is a system storage formal type if it is a system storage type or a zero-based array of a system storage type.

bind-argument-to-value-parameter : *Identifier* × *Formal-type* × *Argument* → *Environment* $\xrightarrow{o}$ *Environment*

bind-argument-to-value-parameter (*id*, *ftype*, *arg*)ρ $\triangleq$
  let *typed* = *d-formal-type* (*ftype*)ρ in 105
  if *is-system-storage-parameter* (*typed*)ρ 264
  then def *locv* = *storage-as-loc*(*typed*, *arg*) ρ in 269
    return *overwrite-var-environment* ({*id* ↦ *locv*})ρ 316
  else def *var* = *bind-value*(*typed*, *arg*) ρ in 267
    return *overwrite-var-environment* ({*id* ↦ *var*})ρ 316

bind-value : *Variable-typed* × *Typed-value* → *Environment* $\xrightarrow{o}$ *Variable*

bind-value (*vtype*, *arg*)ρ $\triangleq$
  let *mk-Typed-value* (*val*, -) = *arg* in
  (*is-Typed*(*vtype*) → def *var* = *allocate-program-storage*(*vtype*) ρ in 337
    (*assign*(*var*, *val*) ρ; 150
    return *var*),
   is-Open-array-typed(*vtype*) → let *mk-Open-array-typed* (*ctype*) = *vtype* in
    let *indx* = *map-indices* (dom *val*) in 269
    def *array* = *bind-open-array-value*(*ctype*, *arg*, *indx*) ρ in 267
    return *mk-Array-variable* (*array*))

bind-open-array-value : *Typed* × *Value* × ($\mathbb{N}$ $\xrightarrow{m}$ *Ordinal-value*) → *Environment* $\xrightarrow{o}$ *Variable*

bind-open-array-value (*ctype*, *val*, *indx*)ρ $\triangleq$
  if *indx* = {↦}
  then return {↦}
  else let *i* ∈ dom *indx* in
    def *component* = *bind-value*(*ctype*, *val*(*indx*(*i*))) ρ in 267
    def *rest* = *bind-open-array-value*(*ctype*, *val*, {*i*} ⩤ *indx*) ρ in 267
    return {*i* ↦ *component*} ∪ *rest*

6.9.4.2 Argument Binding to Variable Parameters

If the formal parameter is a variable parameter, and the formal type is not a system storage formal type or an open array of a system storage formal type (see 7.1.3.1 and 6.9.3.3), then, during the execution of the procedure block, the formal parameter identifier shall designate a formal variable that is the actual variable given in the corresponding argument of the call.

NOTE 1 — Since a formal parameter identifier of a variable parameter designates the actual variable given in the corresponding argument of the call, the formal parameter identifier of a variable parameter stands as an alias for the actual variable during execution of the procedure body; however, the scope of the declared identifier for the actual variable does not necessarily include that procedure body.

If a formal type is an open array formal type, the index type of the formal variable shall have the structure of a subrange type whose lower bound is 0 and whose upper bound is one less than the number of components of the corresponding actual variable. The component type of the formal variable shall be determined recursively by the open array component type of the formal type, and by the structure of components of the corresponding actual variable.

NOTE 2 — The upper bound of the index type may be ascertained using the predefined function procedure `HIGH` — see 6.10.4.7.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLE — Binding of an open array variable parameter.

Given:

```
TYPE colour = (red, yellow, green);

VAR a: ARRAY [-7..-2] OF ARRAY BOOLEAN
                 OF ARRAY colour OF INTEGER;

PROCEDURE
  p(VAR f: ARRAY OF ARRAY OF ARRAY OF INTEGER);
```

then, during activation by 'p(a)', within the body of p the identifier f will be an alias for a, the structure of f will correspond to the declaration

```
VAR f: ARRAY [0..5] OF ARRAY [0..1]
                    OF ARRAY [0..2] OF INTEGER;
```

and the variable designated by f[0][0][0] will be variable designated by a[-7][false][red].

System storage formal types

If the formal parameter is a variable parameter, and the formal type is a system storage formal type F, the effect shall be for the storage for the actual variable to appear as uninterpreted storage locations when accessed through the formal variable which will have the same structure as the structure of F.

If the formal parameter is a variable parameter, and the formal type is an open array of a system storage formal type F, the effect shall be for the storage for the actual variable of type T to appear as an array of uninterpreted storage locations when accessed through the formal variable which will have the structure ARRAY [0..n-1] OF F where n = SIZE(T)/SIZE(F).

bind-argument-to-variable-parameter : *Identifier* × *Formal-type* × *Argument* → *Environment* $\xrightarrow{o}$ *Environment*

bind-argument-to-variable-parameter (*id*, *ftype*, *arg*)ρ $\triangleq$
  let *typed* = *d-formal-type* (*ftype*)ρ in 105
  if *is-system-storage-parameter* (*typed*)ρ 264
  then let *mk-Typed-variable* (*var*, *type*) = *arg* in
    def *val* = *variable-value*(*var*) in 224
    let *varg* = *mk-Typed-value* (*val*, *type*) in
    def *locv* = *storage-as-loc*(*typed*, *varg*) ρ in 269
    (*set-aliases*(*locv*, *var*) ; 336
     return *overwrite-var-environment* ({*id* ↦ *locv*})ρ) 316
  else def *var* = *bind-variable*(*typed*, *arg*) ρ in 268
    return *overwrite-var-environment* ({*id* ↦ *var*})ρ 316

annotations If the formal parameter is a system storage parameter, the actual variable is aliased to storage that consists of one or more storage locations and gives an alternative representation to the value(s) found in the actual variable; the identifier of the formal parameter is bound to this variable. If the formal parameter is not a sytem storage parameter, the identifier of the formal parameter is bound to the actual variable.

bind-variable : *Variable-typed* × *Typed-variable* → *Environment* $\xrightarrow{o}$ *Variable*

bind-variable (*vtype*, *arg*)ρ $\triangleq$
  let *mk-Typed-variable* (*var*, -) = *arg* in
  (*is-Typed*(*vtype*) → return *var*,
   is-Open-array-type(*vtype*) → let *mk-Open-array* (*ctype*) = *vtype* in
     let *mk-Array-variable* (*avar*) = *var* in
     let *indx* = *map-indices* (dom *avar*) in 269
     def *array* = *bind-open-array-variable*(*ctype*, *avar*, *indx*) ρ in 268
     return *mk-Array-variable* (*array*))

bind-open-array-variable : *Typed* × *Variable* × (N $\xrightarrow{m}$ *Ordinal-value*) → *Environment* $\xrightarrow{o}$ *Variable*

bind-open-array-variable (*ctype*, *arg*, *indx*)ρ $\triangleq$
  if *indx* = {↦}
  then return {↦}
  else let *i* ∈ dom *indx* in
    def *component* = *bind-variable*(*ctype*, *arg*(*indx*(*i*))) ρ in 268
    def *rest* = *bind-open-array-variable*(*ctype*, *arg*, {*i*} ⩤ *indx*) ρ in 268
    return {*i* ↦ *component*} ∪ *rest*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Auxiliaries

storage-as-loc : *Typed* × *Argument* → *Environment* $\xrightarrow{o}$ *Variable*

storage-as-loc (*type*, *arg*)ρ $\triangleq$
  let *mk-Typed-value* (*val*, *vtype*) = *arg* in
  (*is-Open-array-typed*(*type*) → let *ctype* = *component-type-of* (*type*) in 325
    let *sizef* = *tsize* (*ctype*)ρ in 366
    let *sizet* = *tsize* (*vtype*)ρ in 366
    let *mk-* (*atype*, ρ_e) = *build-array-type* (*sizet*/*sizef* − 1, *ctype*) in 269
    def *var* = *allocate-program-storage*(*atype*) ρ_e in 337
    def *cval* = *cast-value*(*atype*, *val*) in 365
    (*assign*(*var*, *cval*) ρ_e; 150
     return *var*),
  others → def *var* = *allocate-program-storage*(*type*) ρ in 337
    def *cval* = *cast-value*(*type*, *val*) in 365
    (*assign*(*var*, *cval*) ρ; 150
     return *var*))

annotations Allocate storage for the structure of the formal variable and initialize it to the value of the actual variable.

build-array-type : N × *Typed* → *Environment* → *Type-name* × *Environment*

build-array-type (*n*, *rtype*)ρ $\triangleq$
  let *range* = *mk-Set-value* ({0, . . . , *n*}) in
  let *sub* = *mk-Subrange-structure* (UNSIGNED-TYPE, *range*) in
  let *iname* = *generate-type-name* (ρ) in 320
  let ρ_s = *overwrite-struc-environment* ({*iname* ↦ *sub*})ρ in 315
  let *array* = *mk-Array-structure* (*iname*, *rtype*) in
  let *aname* = *generate-type-name* (ρ_s) in 320
  mk- (*aname*, *overwrite-struc-environment* ({*aname* ↦ *array*})ρ_s) 315

map-indices : *Ordinal-value*-set → (N $\xrightarrow{m}$ *Ordinal-value*)

map-indices (*ovs*) $\triangleq$
  let *rngs* = [*e* | *e* ∈ *ovs*] in
  {*i* ↦ *rngs*(*i* + 1) | *i* ∈ {0, . . . , card *ovs* − 1}}

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.10 Predefined Types, Standard Procedures and Standard Functions

The language provides some predefined entities; these include predefined constants, elementary types, proper procedures and function procedures. Besides the entities described in this clause, there are other predefined entities that are denoted by identifiers exported from system modules — see section 7.

6.10.1 The Pervasive Environment

Declaration Semantics

values

ρ_{PERV} = let *constants* = {NIL ↦ *mk-Constant*(NIL-TYPE, NIL-VALUE),
FALSE ↦ *mk-Constant*(BOOLEAN-TYPE, false),
TRUE ↦ *mk-Constant*(BOOLEAN-TYPE, true),
INTERRUPTIBLE ↦ *mk-Constant*(PROTECTION-TYPE, INTERRUPTIBLE),
UNINTERRUPTIBLE ↦ *mk-Constant*(PROTECTION-TYPE, UNINTERRUPTIBLE)},
types = {BITSET ↦ BITSET-TYPE,
BOOLEAN ↦ BOOLEAN-TYPE,
CARDINAL ↦ UNSIGNED-TYPE,
CHAR ↦ CHARACTER-TYPE,
COMPLEX ↦ COMPLEX-TYPE,
INTEGER ↦ SIGNED-TYPE,
LONGREAL ↦ LONG-REAL-TYPE,
LONGCOMPLEX ↦ LONG-COMPLEX-TYPE,
REAL ↦ REAL-TYPE,
PROC ↦ PROC-TYPE,
PROTECTION ↦ PROTECTION-TYPE},
structures = {BITSET-TYPE ↦ *bitset-structure*,
PROC-TYPE ↦ *proc-structure*},
variables = {↦},
procedures = {DEC ↦ STANDARD-PROPER-PROCEDURE,
DISPOSE ↦ STANDARD-PROPER-PROCEDURE,
EXCL ↦ STANDARD-PROPER-PROCEDURE,
HALT ↦ STANDARD-PROPER-PROCEDURE,
INC ↦ STANDARD-PROPER-PROCEDURE,
INCL ↦ STANDARD-PROPER-PROCEDURE,
NEW ↦ STANDARD-PROPER-PROCEDURE,
ABS ↦ STANDARD-FUNCTION-PROCEDURE,
CAP ↦ STANDARD-FUNCTION-PROCEDURE,
CHR ↦ STANDARD-FUNCTION-PROCEDURE,
CMPLX ↦ STANDARD-FUNCTION-PROCEDURE,
FLOAT ↦ STANDARD-FUNCTION-PROCEDURE,
HIGH ↦ STANDARD-FUNCTION-PROCEDURE,
IM ↦ STANDARD-FUNCTION-PROCEDURE,
LENGTH ↦ STANDARD-FUNCTION-PROCEDURE,
LFLOAT ↦ STANDARD-FUNCTION-PROCEDURE,
MAX ↦ STANDARD-FUNCTION-PROCEDURE,
MIN ↦ STANDARD-FUNCTION-PROCEDURE,
ODD ↦ STANDARD-FUNCTION-PROCEDURE,
ORD ↦ STANDARD-FUNCTION-PROCEDURE,
RE ↦ STANDARD-FUNCTION-PROCEDURE,
SIZE ↦ STANDARD-FUNCTION-PROCEDURE,
TRUNC ↦ STANDARD-FUNCTION-PROCEDURE,
VAL ↦ STANDARD-FUNCTION-PROCEDURE,
INT ↦ STANDARD-FUNCTION-PROCEDURE},

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

modules = {↦},
defmods = {↦},
level = 0,
local-continuations = {↦},
protection-domain = nil ,
protected-procedures = { } in
mk-Environment (*constants*, *types*, *structures*, *variables*, *procedures*, *modules*, *defmods*
level, *local-continuations*, *protection-domain*, *protected-procedures*);

ρ_{empty} = let *constants* = {↦},
types = {↦},
structures = {↦},
variables = {↦},
procedures = {↦},
modules = {↦},
defmods = {↦},
level = 0,
local-continuations = {↦},
protection-domain = nil ,
protected-procedures = { } in
mk-Environment (*constants*, *types*, *structures*, *variables*, *procedures*, *modules*, *defmods*,
level, *local-continuations*, *protection-domain*, *protected-procedures*)

6.10.2 Predefined Types

There are 16 required types other than those with identifiers exported from system modules: three whole number types, three real number types, three complex number types, the Boolean type, the character type, the string literal type, the nil type, the bitset type, the proc type, and the protection type.

The predefined types are all elementary types. The bitset type is a predefined packedset type and the proc type is a predefined procedure type, whereas the number, Boolean, character, string literal, and nil types constitute the basic types of the language.

The protection type is an implementation-defined type and may either be a basic type or a predefined constructed type, as appropriate to the underlying interrupt structure of the implementation.

The number types include types for numeric literals. The string literal type includes the character literal type as a special case. No corresponding type identifiers are provided for the literal types or for the nil type and so it is not possible to declare variables of these anonymous types; they can only appear as the value of an expression.

Of the basic types, the whole number types, the Boolean type, and the character type are ordinal types. These basic ordinal types, together with the real number types, constitute the basic scalar types. The character literal type is interpreted as a value of the character type in situations where a value of an ordinal type is expected.

6.10.2.1 Predefined Whole Number Types

Three predefined types are whole number types: the unsigned type, the signed type and the ℤ-type. The unsigned type can be used to represent non-negative whole numbers, and the signed type can be used to represent whole numbers that are possibly less than zero. The ℤ-type is the type of a whole number literal value, and of any expression all of whose factors are whole number constants.

Declaration Semantics

The unsigned type shall be a whole number type that is denoted by the pervasive type identifier `CARDINAL` . A value of the unsigned type shall be an element of an implementation-defined finite subset of the non-negative whole numbers

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

— see 6.10.2.6.

The ordinal number of a value of the unsigned type shall be the value itself.

NOTES

1 If x is a value of the unsigned type, then x satisfies the condition:

$$x \in \{0, \ldots, \texttt{MAX(CARDINAL)}\}$$

2 When choosing the range of the unsigned type, an implementor should consider the need to represent results returned by calls of the predefined functions HIGH and SIZE.

The signed type shall be a whole number type that is denoted by the pervasive type identifier INTEGER . A value of the signed type shall be an element of an implementation-defined finite subset of the whole numbers — see 6.10.2.6.

The ordinal number of a non-negative value of the signed type shall be the value itself.

NOTES

3 If x is a value of the signed type, then x satisfies the condition:

$$x \in \{\texttt{MIN(INTEGER)}, \ldots, \texttt{MAX(INTEGER)}\}$$

4 There are no ordinal numbers associated with values of the signed type that are less than zero.

The $\mathbb{Z}$-type shall be a whole number type that is the type of constant expressions all of whose factors are whole number constants. A value of the $\mathbb{Z}$-type shall be an element of an implementation-defined finite subset of the whole numbers. — see 6.10.2.6.

NOTES

5 There is no pervasive type identifier that denotes the $\mathbb{Z}$-type.

6 A whole number literal can be used to denote a value of the $\mathbb{Z}$-type — see 6.8.7.1.

7 If x is a value of the $\mathbb{Z}$-type, then x satisfies the condition:

$$\textit{min-zz-value} \leq x \leq \textit{max-zz-value}$$

8 All numeric subrange types have a host type that is either the signed type or the unsigned type; it is not possible to construct a numeric type by using values of the $\mathbb{Z}$-type for the lower and upper bounds of a subrange type that are not both within the range of either the signed type or the unsigned type — see 6.3.5.

6.10.2.2 Predfined Real Number Types

Three predefined types are real number types: the real type, the long real type and the $\mathbb{R}$-type. Both the real type and the long real type can be used to represent real numbers. These types may differ with respect to the range and the precision with which real numbers are represented. The $\mathbb{R}$-type is the type of a real number literal value, and of any expression all of whose factors are real constants.

Declaration Semantics

The real type shall be a real number type that is denoted by the pervasive type identifier REAL . A value of the real type shall be an element of an implementation-defined finite subset of the real numbers — see 6.10.2.6.

NOTE 1 — If x is a value of the real type, then x satisfies the condition:

$$\texttt{MIN(REAL)} \leq x \leq \texttt{MAX(REAL)}$$

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The long real type shall be a real number type that is denoted by the pervasive type identifier `LONGREAL` . A value of the long real type shall be an element of an implementation-defined finite subset of the real numbers — see 6.10.2.6.

NOTE 2 — If x is a value of the long real type, then x satisfies the condition:

$$\texttt{MIN(LONGREAL)} \leq x \leq \texttt{MAX(LONGREAL)}$$

The ℝ-type shall be a real number type that is the type of constant expressions all of whose factors are real constants. A value of the ℝ-type shall be an element of an implementation-defined finite subset of the real numbers — see 6.10.2.6.

NOTES

3 There is no pervasive type identifier that denotes the ℝ-type.

4 A real literal can be used to denote a value of the ℝ-type — see 6.8.7.2.

5 If x is a value of the ℝ-type, then x satisfies the condition:

$$\textit{min-rr-value} \leq x \leq \textit{max-rr-value}$$

The precision with which real number values are represented shall not decrease from the real type to the long real type, or from the long real type to the ℝ-type.

6.10.2.3 Predefined Complex Number Types

Three predefined types are complex number types: the complex type, the long complex type and the ℂ-type. Both the complex type and the long complex type can be used to represent complex numbers. These types may differ with respect to the range and the precision with which complex numbers are represented. The ℂ-type is the type of a complex number literal value, and of any expression all of whose factors are complex constants.

CHANGE — The complex number types are not described in *Programming in Modula-2*.

Declaration Semantics

The complex type shall be a complex number type that is denoted by the pervasive type identifier `COMPLEX` . A value of the complex type shall be an element of an implementation-defined finite subset of the complex numbers — see 6.10.2.6.

NOTE 1 — If z is a value of the complex type, then z satisfies the condition:

$$\texttt{MIN(REAL)} \leq \mathsf{re}\,z \leq \texttt{MAX(REAL)} \wedge \texttt{MIN(REAL)} \leq \mathsf{im}\,z \leq \texttt{MAX(REAL)}$$

The long complex type shall be a complex number type that is denoted by the pervasive type identifier `LONGCOMPLEX` A value of the long complex type shall be an element of an implementation-defined finite subset of the complex numbers — see 6.10.2.6.

NOTE 2 — If z is a value of the long complex type, then z satisfies the condition:

$$\texttt{MIN(LONGREAL)} \leq \mathsf{re}\,z \leq \texttt{MAX(LONGREAL)} \wedge \texttt{MIN(LONGREAL)} \leq \mathsf{im}\,z \leq \texttt{MAX(LONGREAL)}$$

The ℂ-type shall be a complex number type that is the type of constant expressions all of whose factors are complex constants. A value of the ℂ-type shall be an element of an implementation-defined finite subset of the complex numbers — see 6.10.2.6.

NOTES

3 There is no pervasive type identifier that denotes the ℂ-type.

4 Although there is no denotation for a complex literal, a value of the ℂ-type can be formed by applying the pervasive standard function `CMPLX` to two real literals; for example '`CMPLX(0.0, 1.0)`' — see 6.10.4.5.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

5 If z is a value of the $\mathbb{C}$-type, then z satisfies the condition:

$$min\text{-}rr\text{-}value \leq \mathsf{re}z \leq max\text{-}rr\text{-}value \wedge min\text{-}rr\text{-}value \leq \mathsf{im}z \leq max\text{-}rr\text{-}value$$

The precision with which complex number values are represented shall not decrease from the complex type to the long complex type, or from the long complex type to the $\mathbb{C}$-type.

6.10.2.4 The Boolean Type

The Boolean type can be used to represent truth values.

Declaration Semantics

The Boolean type shall be an ordinal type that is denoted by the pervasive type identifier `BOOLEAN` . A value of the Boolean type shall be one of the logical truth values *false* or *true*. The values *false* and *true* shall be denoted by the pervasive constant identifiers `FALSE` and `TRUE` respectively .

The ordinal numbers of the values *false* and *true* shall be the values 0 and 1 respectively.

6.10.2.5 The Character Type

The character type can be used to represent single characters.

Declaration Semantics

The character type shall be an ordinal type that is denoted by the pervasive type identifier `CHAR` . A value of the character type shall be an element of a finite set of characters. This set shall contain unique values representing the minus character, the period character, the plus character, the space character, the string terminator, the lower-case letters, the upper-case letters, and the digits. Any other values of this set of characters shall be implementation-defined.

CHANGE — There is no requirement that the value of the string terminator be the same as the value denoted by the character number literal '`0C`', as is specified in *Programming in Modula-2*.

NOTES

1 If c is a value of the character type, then c satisfies the condition:

$$c \in \{\texttt{MIN(CHAR)}, \ldots, \texttt{MAX(CHAR)}\}$$

2 A value of the character type may represent a character that does not have a graphic representation.

3 The character type is assignment-compatible and expression-compatible with the character literal type, which is the string literal type of length 1 or of length 0 — see 6.8.7.3.3.

4 The string terminator character can be denoted by either of the character literals '`''`' or '`""`'. Both literals represent the same value of the string literal type.

Domains

types

$Char$ = char-set

inv $ch \triangleq$
$ch \in characters$

annotations The type *Char* is used to model the character type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

values

characters = *required-characters* ∪ *implementation-characters*;

required-characters = *required-lower-cases* ∪ *required-upper-cases* ∪ *digits* ∪ *required-other-characters*;

required-lower-cases = elems *required-lower-cases-order*;

required-upper-cases = elems *required-upper-cases-order*;

digits = elems *digits-order*;

required-lower-cases-order = ['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'];

required-upper-cases-order = ['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'];

digits-order = ['0','1','2','3','4','5','6','7','8','9'];

required-other-characters = END-OF-STRING-CHAR ∪ {*minus-character*, *period-character*, *plus-character*, *space-character*};

minus-character = '-';

period-character = '.';

plus-character = '+';

space-character = ' ';

implementation-characters = *implementation-lower-cases* ∪ *implementation-upper-cases* ∪ *implementation-other-characters*;

implementation-lower-cases = elems *implementation-lower-cases-order*;

implementation-upper-cases = elems *implementation-upper-cases-order*;

implementation-lower-cases-order : char* = implementation-defined;

implementation-upper-cases-order : char* = implementation-defined;

implementation-other-characters : char-set = implementation-defined

properties

required-characters ∩ *implementation-characters* = { }
card *implementation-lower-cases* = len *implementation-lower-cases-order*
card *implementation-upper-cases* = len *implementation-upper-cases-order*
card *implementation-lower-cases* = card *implementation-upper-cases*
is-disj ({*implementation-lower-cases*, *implementation-upper-cases*, *implementation-other-characters*}) 614

The ordinal numbers of the character values shall be implementation-defined values of the unsigned type that satisfy the conditions:

— the ordinal number of each character in the set of characters shall be different from the ordinal number of any other character in the set;

— the ordinal number associated with the $(i + 1)$th lower-case letter shall be greater than that associated with the ith lower-case letter;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

— the ordinal number associated with the $(i + 1)$th upper-case letter shall be greater than that associated with the ith upper-case letter;

— the ordinal number associated with the $(i + 1)$th decimal digit shall be one more than that associated with the ith decimal digit.

values

$collation\text{-}map : Char \xrightarrow{m} \mathbb{N} =$ implementation-defined

annotations The collation map associated with the type *Char* maps from values of the character type to the ordinal numbers of those values.

properties

$\text{dom } collation\text{-}map = characters$
$\forall i \in \{1, \ldots, 25\} \cdot$
$\quad collation\text{-}map(required\text{-}lower\text{-}cases\text{-}order(i + 1)) > collation\text{-}map(required\text{-}lower\text{-}cases\text{-}order(i))$
$\forall i \in \{1, \ldots, 25\} \cdot$
$\quad collation\text{-}map(required\text{-}upper\text{-}cases\text{-}order(i + 1)) > collation\text{-}map(required\text{-}upper\text{-}cases\text{-}order(i))$
$\forall i \in \{1, \ldots, 9\} \cdot$
$\quad collation\text{-}map(digits\text{-}order(i + 1)) = collation\text{-}map(digits\text{-}order(i)) + 1$

NOTES

5 The ordering relationship between any two character values is the same as that between their ordinal numbers — see 6.8.2.7.2. Thus if `x` and `y` are any two character values, then:

$$\texttt{x} < \texttt{y} \Rightarrow \texttt{VAL(CARDINAL,x)} < \texttt{VAL(CARDINAL,y)}$$

6 The ordinal number of a character need not be the same as the internal code used to represent the character. For example, if `ch` is a variable of type `CHAR`, whose current value is `5C`, then, although '`VAL(CARDINAL,ch)`' has the value 5, this need not be the value of '`SYSTEM.CAST(CARDINAL,ch)`'.

6.10.2.6 Values of the Basic Types

6.10.2.6.1 Ranges of the Basic Types

$maximum\text{-}of\text{-}basic\text{-}type : Basic\text{-}type \rightarrow Value$

$maximum\text{-}of\text{-}basic\text{-}type\,(type) \triangleq$
$\quad (type = \text{UNSIGNED-TYPE} \rightarrow max\text{-}unsigned\text{-}value,$
$\quad\ type = \text{SIGNED-TYPE} \rightarrow max\text{-}signed\text{-}value,$
$\quad\ type = \mathbb{Z}\text{-TYPE} \rightarrow max\text{-}zz\text{-}value,$
$\quad\ type = \text{REAL-TYPE} \rightarrow max\text{-}real\text{-}value,$
$\quad\ type = \text{LONG-REAL-TYPE} \rightarrow max\text{-}long\text{-}real\text{-}value,$
$\quad\ type = \mathbb{R}\text{-}type \rightarrow max\text{-}rr\text{-}value,$
$\quad\ type = \text{BOOLEAN-TYPE} \rightarrow \text{true},$
$\quad\ type = \text{CHARACTER-TYPE} \rightarrow \text{let } high = maxs(\text{dom } characters) \text{ in } characters(high))$ 614

annotations Return the largest value of a basic scalar type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

minimum-of-basic-type : *Basic-type* → *Value*

minimum-of-basic-type (*type*) ≙
  (*type* = UNSIGNED-TYPE → 0,
   type = SIGNED-TYPE → *min-signed-value*,
   type = ℤ-TYPE → *min-zz-value*,
   type = REAL-TYPE → *min-real-value*,
   type = LONG-REAL-TYPE → *min-long-real-value*,
   type = ℝ-*type* → *min-rr-value*,
   type = BOOLEAN-TYPE → false,
   type = CHARACTER-TYPE → let *low* = *mins* (dom *characters*) in *characters* (*low*)) 614

annotations Return the smallest value of a basic scalar type.

values-of-basic-type : *Basic-type* → *Value*-set

values-of-basic-type (*type*) ≙
  (*is-Number-type*(*type*) →
      {*x* | *x* : *type* · *minimum-of-basic-type* (*type*) ≤ *x* ∧ *x* ≤ *maximum-of-basic-type* (*type*)}, 276 276
   type = BOOLEAN-TYPE → {false, true},
   type = CHARACTER-TYPE → elems *characters*)

annotations Return the set of values associated with a basic ordinal type.

6.10.2.6.2 Ranges of the Predefined Numeric Types

The implementation-defined values specifying the ranges of the predefined numeric types shall have the following properties.

properties

(let $k_1 = -$ *min-signed-value*,
   k_2 = *max-signed-value*,
   k_3 = *max-unsigned-value* in
$k_1 > 0 \wedge k_2 > 0 \wedge k_3 > 0) \wedge$
min-zz-value ≤ *min-signed-value* ∧
max-zz-value ≥ *max-signed-value* ∧
max-zz-value ≥ *max-unsigned-value* ∧
max-rr-value ≥ *max-long-real-value* ∧
max-long-real-value ≥ *max-real-value* ∧
min-rr-value ≤ *min-long-real-value* ∧
min-long-real-value ≤ *min-real-value* ∧
min-real-value < 0 ∧ *max-real-value* > 0 ∧
re-max-complex-value ≤ *max-real-value* ∧
re-max-complex-value ≤ *max-real-value* ∧
im-max-long-complex-value ≤ *max-long-real-value* ∧
im-max-long-complex-value ≤ *max-long-real-value* ∧
re-max-cc-value ≤ *max-rr-value* ∧
re-max-cc-value ≤ *max-rr-value* ∧
re-min-complex-value ≥ *min-real-value* ∧
re-min-complex-value ≥ *min-real-value* ∧
im-min-long-complex-value ≥ *min-long-real-value* ∧
im-min-long-complex-value ≥ *min-long-real-value* ∧
re-min-cc-value ≥ *min-rr-value* ∧
re-min-cc-value ≥ *min-rr-value*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

values

max-unsigned-value = An implementation-defined unsigned value;

max-signed-value = An implementation-defined signed value;

max-zz-value = An implementation-defined constant value;

max-real-value = An implementation-defined real value;

max-long-real-value = An implementation-defined long real value;

max-rr-value = An implementation-defined real constant value;

re-max-complex-value = An implementation-defined real constant value;

im-max-complex-value = An implementation-defined real constant value;

re-max-long-complex-value = An implementation-defined real constant value;

im-max-long-complex-value = An implementation-defined real constant value;

re-max-cc-value = An implementation-defined real constant value;

im-max-cc-value = An implementation-defined real constant value;

min-signed-value = An implementation-defined signed value;

min-zz-value = An implementation-defined constant value;

min-real-value = An implementation-defined real value;

min-long-real-value = An implementation-defined long real value;

min-rr-value = An implementation-defined real constant value;

re-min-complex-value = An implementation-defined real constant value;

im-min-complex-value = An implementation-defined real constant value;

re-min-long-complex-value = An implementation-defined real constant value;

im-min-long-complex-value = An implementation-defined real constant value;

re-min-cc-value = An implementation-defined real constant value;

im-min-cc-value = An implementation-defined real constant value

6.10.2.7 The String Literal Type

The string literal type is the type of string literals, of string literal constants, and of the result of string literal catenation.

The string literal type has an associated length. If the length associated with the string literal type is zero or one, the type is also known as the character literal type.

Declaration Semantics

The string literal type shall be an elementary type that is the type of constant expressions all of whose factors are string literals or string literal constants.

A value of the string literal type shall be a sequence of zero or more characters. The string literal type shall have an associated length equal to the number of characters in a value of that type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NOTES

1 The only operation allowed in string literal expressions is catenation — see 6.8.2.6.

2 There is no pervasive type identifier that denotes the string literal type.

3 A quoted string can be used to denote a value of the string literal type of length zero or more — see 6.8.7.3.1.

4 A character number literal can be used to denote a value of the string literal type of length one that has a particular ordinal number — see 6.8.7.3.2.

5 A concrete string type of capacity n, which is an array type with n components of the character type, is assignment-compatible with the string literal type of length l provided $l \leq n$ — see 6.4.2.

6 An open array of the character type is value parameter compatible with the string literal type — see 6.9.3.1.

7 If the length associated with the string literal type is zero or one, the type is also known as the character literal type.

8 Values of the character literal type are interpreted as values of the character type in contexts where a value of an ordinal type is required — see 6.8.7.3.3.

9 The character type is expression-compatible and assignment-compatible with the character literal type — see 6.4.1 and 6.4.2.

10 An implementation may choose different representations for the elements of a string literal and the values of the character type.

6.10.2.8 The Nil Type

The nil type is the type of the nil value.

Declaration Semantics

The nil type shall be the type of the nil value.

The nil value shall be denoted by the pervasive constant identifier `NIL` .

NOTE — All pointer types are assignment-compatible and expression-compatible with the nil type.

6.10.2.9 The Bitset Type

The bitset type is a packedset type whose base type is a subrange from zero to one less than the number of bits in a machine word of the underlying computer system. The bitset type may therefore be used to represent small sets in an efficient manner, or to manipulate the bits in a machine word — see 6.3.7 and 7.1.7.1.

Declaration Semantics

The bitset type shall be a packedset type that is denoted by the pervasive type identifier `BITSET` . The base type of the bitset type shall be a subrange of the unsigned type. The lower bound of this base type shall be the value 0, and the upper bound shall be the value denoted by the constant expression `SYSTEM.BITSPERLOC*SYSTEM.LOCSPERWORD-1`.

NOTE — Although there is no denotation for a literal of the bitset type, a value of the bitset type can be formed by using a set constructor; for example '`BITSET{1, 4..6, 9}`' — see 6.8.6.3.

values

bitset-structure =
mk-Packed-set-structure (*mk-Subrange-structure* (UNSIGNED-TYPE, $\{0, \ldots, bitsperloc \times locsperword - 1\}$))

6.10.2.10 The Proc Type

The proc type is a procedure type that can be used to represent the type of parameterless proper procedures that are declared at level 0.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Declaration Semantics

The proc type shall be a proper procedure type that is denoted by the pervasive type identifier PROC . A value of the proc type shall denote a parameterless proper procedure that has been declared at level 0.

NOTE — The proc type is effectively defined by the implicit declaration:

```
TYPE
  PROC = PROCEDURE;
```

values

proc-structure = *mk-Proper-procedure-structure* ([])

6.10.2.11 The Protection Type

The protection type can be used to represent an extent of protection from interrupts.

CHANGE — The treatment of protection differs from that described (as "priority") in *Programming in Modula-2*, which was modelled on the PDP-11 interrupt structure.

Declaration Semantics

The protection type shall be an implementation-defined elementary type that is denoted by the pervasive type identifier PROTECTION. A value of the protection type shall be one of an implementation-defined set of values that shall include the values INTERRUPTIBLE and UNINTERRUPTIBLE; these values shall be denoted by the pervasive constant identifiers INTERRUPTIBLE and UNINTERRUPTIBLE respectively.

NOTES

1 An implementation may define the protection type as a basic type, distinct from the other basic types of the language, as identical to another basic type, or as a predefined constructed type, as appropriate to the interrupt structure of the underlying system.

2 It is likely that the interrupt structure of the system underlying an implementation will be richer than the two protection values defined here. An implementation may reflect this richer structure by defining the protection type so that it includes other appropriate values. The underlying interrupt structure might be totally or partially ordered: an ordinal type or a set type would then result for the protection type. The protection relational operators (see 6.8.2.7.6) are those that are common both to ordinal types and to set types. If the protection type is defined to be identical to a basic ordinal type, or is defined as a constructed ordinal type, other corresponding operations will be applicable to protection values.

3 If P is a value of the protection type, then P satisfies the condition:

```
INTERRUPTIBLE <= P AND P <= UNINTERRUPTIBLE
```

— see 6.8.2.7.6.

4 A value P1 of the protection type is less restrictive than another value P2 if and only if P1 does not prevent all the interrupts prevented by P2, but P2 prevents all the interrupts prevented by P1 — see 6.8.2.7.6.

6.10.3 Standard Procedures

The standard procedures are denoted by the pervasive identifiers DEC, DISPOSE, EXCL, HALT, INC, INCL and NEW.

6.10.3.1 Standard Procedure Designators

Abstract Syntax

types

Standard-procedure-designator :: *ident* : *Identifier*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

inv *ident* $\triangleq$
ident $\in$ {DEC, DISPOSE, EXCL, HALT, INC, INCL, NEW}

Static Semantics

wf-standard-procedure-designator :
Standard-procedure-designator $\rightarrow$ *Actual-parameters* $\rightarrow$ *Environment* $\rightarrow \mathbb{B}$

wf-standard-procedure-designator (*desig*)(*aps*)ρ $\triangleq$
access-environment (*desig*)ρ = STANDARD-PROPER-PROCEDURE $\wedge$ 319
cases *desig* :
DEC $\rightarrow$ *wf-dec-designator* (*aps*)ρ, 281
DISPOSE $\rightarrow$ *wf-dispose-designator* (*aps*)ρ, 283
EXCL $\rightarrow$ *wf-excl-designator* (*aps*)ρ, 285
HALT $\rightarrow$ *wf-halt-designator* (*aps*)ρ, 285
INC $\rightarrow$ *wf-inc-designator* (*aps*)ρ, 286
INCL $\rightarrow$ *wf-incl-designator* (*aps*)ρ, 287
NEW $\rightarrow$ *wf-new-designator* (*aps*)ρ 287
end

Dynamic Semantics

m-standard-procedure-designator :
Standard-procedure-designator $\rightarrow$ *Arguments* $\rightarrow$ *Environment* $\xrightarrow{o}$ ()

m-standard-procedure-designator (*desig*)(*args*)ρ $\triangleq$
cases *desig*:
DEC $\rightarrow$ *m-dec-designator*(*args*) ρ, 282
DISPOSE $\rightarrow$ *m-dispose-designator*(*args*) ρ, 284
EXCL $\rightarrow$ *m-excl-designator*(*args*) ρ, 285
HALT $\rightarrow$ *m-halt-designator*(*args*) ρ, 285
INC $\rightarrow$ *m-inc-designator*(*args*) ρ, 286
INCL $\rightarrow$ *m-incl-designator*(*args*) ρ, 287
NEW $\rightarrow$ *m-new-designator*(*args*) ρ 288
end

6.10.3.2 The Procedure DEC

The procedure DEC can be used to decrement the value of a variable that is of an ordinal type.

Static Semantics

A call of DEC shall have one or two actual parameters. The first parameter shall be a variable that is of an ordinal type. If there is a second parameter, it shall be an expression that is of a whole number type.

If the second parameter is of the $\mathbb{Z}$-type, its value shall be subject to a constraint that depends on the type of the first parameter: if the first parameter is of the signed type (or is of a subrange type whose host type is the signed type), the value of the second parameter shall be less than or equal to the maximum value of the signed type; otherwise it shall be greater than or equal to zero.

NOTE — These constraints are present because signed arithmetic will be used if the first parameter is of the signed type, and unsigned arithmetic will be used in all other cases.

CLARIFICATION — In *Programming in Modula-2* the types of the parameters of DEC are not specified.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wf-dec-designator : *Actual-parameters* → *Environment* → 𝔹

wf-dec-designator (*aps*)ρ ≜
  len *aps* ≥ 1 ∧
  is-Typed-variable-designator(*aps*(1)) ∧
  cases *aps* :
    [*tdesig*] → let *mk-Typed-variable-designator* (*desig*, *type*) = *tdesig* in
      wf-dec-inc-var (*type*, *desig*)ρ, 282
    [*tdesig*, *expr*] → let *mk-Typed-variable-designator* (*desig*, *type*) = *tdesig* in
      wf-dec-inc-var (*type*, *desig*)ρ ∧ 282
      wf-dec-inc-value (*type*, *expr*)ρ 282
  end

Dynamic Semantics

The call `DEC(x)` shall assign to the variable `x` the predecessor of its current value. An exception shall be raised if the predecessor lies outside the range of the type of `x`, or if there is no predecessor.

The call `DEC(x,N)` shall assign to the variable `x` the Nth predecessor of its current value. An exception shall be raised if the Nth predecessor lies outside the range of the type of `x`, or if there is no N^{th} predecessor.

m-dec-designator : *Arguments* → *Environment* $\xrightarrow{o}$ ()

m-dec-designator (*args*)ρ ≜
  cases *args*:
    [*tvar*] → let *mk-Typed-variable* (*var*, *type*) = *tvar* in
      m-inc-dec(*type*, *var*, −1) ρ, 282
    [*tvar*, *val*] → let *mk-Typed-variable-designator* (*var*, *type*) = *tvar* in
      m-inc-dec(*type*, *var*, −*val*) ρ 282
  end

m-inc-dec : *Typed* × *Variable* × *Value* → *Environment* $\xrightarrow{o}$ ()

m-inc-dec (*rtype*, *var*, *step*)ρ ≜
  let *vtype* = if *host-type-of* (*rtype*)ρ = SIGNED-TYPE 325
    then SIGNED-TYPE
    else UNSIGNED-TYPE in
  def *vval* = *variable-value*(*var*) in 224
  def *val* = *converted-result*(*vtype*, *vval*) ρ, 306
    change = *converted-result*(*vtype*, *step*) ρ in 306
  def *rval* = *converted-result*(*rtype*, *val* + *change*) ρ in 306
  let *handler* = *current-exception-handler* (ρ) in 318
  assign(*var*, *rval*) *handler* 150

Auxiliaries

wf-dec-inc-var : *Typed* × *Actual-parameter* → *Environment* → 𝔹

wf-dec-inc-var (*rtype*, *var*)ρ ≜
  is-Variable-designator(*var*) ∧
  rtype = *t-variable-designator* (*var*)ρ ∧ 182
  is-ordinal-type (*rtype*)ρ 322

annotations Check whether the first parameter of a call of `DEC` or `INC` is valid.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

$wf\text{-}dec\text{-}inc\text{-}value : Typed \times Actual\text{-}parameter \rightarrow Environment \rightarrow \mathbb{B}$

$wf\text{-}dec\text{-}inc\text{-}value\,(rtype, expr)\rho \triangleq$
  $is\text{-}Expression(expr) \wedge$
  let $vtype = host\text{-}type\text{-}of\,(rtype)\rho$ in 325
  let $etype = t\text{-}expression\,(expr)\rho$ in 190
  $etype \in \{\text{UNSIGNED-TYPE}, \text{SIGNED-TYPE}\} \vee$
  $(etype = \mathbb{Z}\text{-TYPE} \wedge$
  let $val = evaluate\text{-}constant\text{-}expression\,(expr)\rho$ in 257
  $vtype = \text{SIGNED-TYPE} \wedge val \leq maximum\,(\text{SIGNED-TYPE})\rho$ 328
    $\vee$
  $vtype \neq \text{SIGNED-TYPE} \wedge val \geq 0)$

annotations Check whether the two parameters of a call of DEC or INC are valid. This includes checking that the second parameter is within range when it is of the $\mathbb{Z}$-type.

6.10.3.3 The Procedure DISPOSE

The procedure DISPOSE can often be used to release an area of dynamically allocated storage that was earlier claimed by a call to the procedure NEW.

NOTE 1 — The word *often* appears because there is no requirement that the procedure DEALLOCATE that is associated with DISPOSE necessarily be the procedure Storage.DEALLOCATE, or a procedure that performs a similar task.

Static Semantics

A call of DISPOSE shall have one or more actual parameters. The first parameter shall be a variable that is of a pointer type.

If the second and subsequent parameters are present, they shall be constant expressions, and the bound type of the first parameter shall be a record type.

The identifier DEALLOCATE shall be visible at the point of the call of DISPOSE and shall designate a procedure, procedure constant, or procedure variable with the structure of a proper procedure that has two formal parameters. The first formal parameter of this proper procedure structure shall be a variable parameter that is of the address type, and the second formal parameter shall be a value parameter that is of the unsigned type.

NOTE 2 — The separate module Storage exports a procedure called DEALLOCATE of the correct structure for a call of DISPOSE. The requirement that the identifier DEALLOCATE is visible at the point of the call of DISPOSE is often satisfied by including the import list:

```
FROM Storage IMPORT DEALLOCATE;
```

in the module containing the call of DISPOSE.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wf-dispose-designator : *Actual-parameters* → *Environment* → 𝔹

wf-dispose-designator (*aps*)ρ ≜
  len *aps* ≥ 1 ∧
  let [*var*] ↷ *exprs* = *aps* in
  is-Typed-variable-designator(*var*) ∧
  let *mk-Typed-variable-designator* (*desig*, *type*) = *var* in
  type = *t-variable-designator* (*desig*)ρ ∧ 182
  is-pointer-type (*type*)ρ ∧ 322
  wf-value-designator (DEALLOCATE)ρ ∧ 223
  let *typed* = *t-value-designator* (DEALLOCATE)ρ in 223
  structure-of (*typed*)ρ = *storage-call* ∧ 315
  (*expr* = [ ]
    ∨
  expr ≠ [ ] ∧
  let *btype* = *bound-type-of* (*type*)ρ in 326
  let *rs* = *structure-of* (*btype*)ρ in 315
  is-Record-structure(*rs*) ∧
  is-tags-in-field-list (*rs*, *exprs*) ∧ 366
  ∀ *expr* ∈ elems *exprs* · *is-Expression*(*expr*) ∧
    wf-expression (*expr*)ρ ∧ 190
    is-constant-expression (*expr*)ρ) 254

Dynamic Semantics

The call `DISPOSE(P)` shall have the same effect as the call `DEALLOCATE(P,SIZE(B))`, where the variable `P` is of a pointer type `T`, and `B` is the bound type given in the declaration of `T`.

The call `DISPOSE(P, <` *expression-list* `>)` shall have the same effect as the call

`DEALLOCATE(P,SYSTEM.TSIZE(B, <` *expression-list* `>))`,

where the variable `P` is of a pointer type `T`, and `B` is the bound type given in the declaration of `T`.

m-dispose-designator : *Arguments* → *Environment* $\xrightarrow{o}$ ()

m-dispose-designator (*args*)ρ ≜
  let [*mk-Typed-variable* (*var*, *vtype*)] ↷ *vals* = *args* in
  let *size* = *tsize* (*vtype.type*, *vals*)ρ in 366
  def *procid* = *m-value-designator*(DEALLOCATE) ρ in 223
  let *handler* = *current-exception-handler* (ρ) in 318
  call-proper-procedure-value(*procid*) ([*var*, *size*]) *handler* 339

Auxiliaries

values

  storage-call =
    let *args* = [*mk-Variable-formal-typed* (ADDRESS-TYPE),
      mk-Value-formal-typed (UNSIGNED-TYPE)] in
    mk-Proper-procedure-structure (*args*)

6.10.3.4 The Procedure EXCL

The procedure `EXCL` can be used to remove an element from a variable that is of a set type or a packedset type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Static Semantics

A call of EXCL shall have two actual parameters. The first parameter shall be a variable that is of a set type or a packedset type, and the second parameter shall be an expression with which the the host type of the base type of the set type or packedset type is assignment-compatible.

wf-excl-designator : *Actual-parameters* → *Environment* → $\mathbb{B}$

wf-excl-designator (*aps*)ρ $\triangleq$
  len *aps* = 2 ∧
  let [*var*, *expr*] = *aps* in
  is-Typed-variable-designator(*var*) ∧
  let *mk-Typed-variable-designator* (*desig*, *rtype*) = *var* in
  is-Expression(*expr*) ∧
  rtype = *t-variable-designator* (*desig*)ρ ∧ 182
  is-set-type (*rtype*)ρ ∧ 322
  is-assignment-compatible (*host-type-of* (*base-type-of* (*rtype*)ρ)ρ, *t-expression* (*expr*)ρ)ρ 123 325 325

Dynamic Semantics

The call EXCL(s,x) shall assign to the set variable s the value that is obtained by removing the value of x from the current value of s. An exception shall be raised if the value of x is not a value of the base type of s.

m-excl-designator : *Arguments* → *Environment* $\xrightarrow{o}$ ()

m-excl-designator (*args*)ρ $\triangleq$
  let [*mk-Typed-variable* (*var*, *rtype*), *val*] = *args* in
  let *mk-Set-structure* (*btype*) = *structure-of*(*rtype*)ρ in
  let *range* = *values-of* (*btype*)ρ in 329
  if *val* ∈ *range*
  then def *value* = *variable-value*(*var*) in 224
    assign(*var*, *value* − {*val*}) ρ 150
  else let *handler* = *current-exception-handler* (ρ) in 318
    mandatory-detection(EXCLNOTBASE) *handler* 396

6.10.3.5 The Procedure HALT

The procedure HALT can be used to terminate the execution of a program at a point other than the end of the program block.

Static Semantics

A call of HALT shall have no actual parameters.

wf-halt-designator : *Actual-parameters* → *Environment* → $\mathbb{B}$

wf-halt-designator (*aps*)ρ $\triangleq$
  len *aps* = 0

Dynamic Semantics

The call HALT shall be a program termination event — see 6.1.7

m-halt-designator : *Arguments* → *Environment* $\xrightarrow{o}$ ()

m-halt-designator ([])ρ $\triangleq$
  note-HALT() 341

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.10.3.6 The Procedure INC

The procedure INC can be used to increment the value of a variable that is of an ordinal type.

Static Semantics

A call of INC shall have one or two actual parameters. The first parameter shall be a variable that is of an ordinal type. If there is a second parameter, it shall be an expression that is of a whole number type.

If the second parameter is of the $\mathbb{Z}$-type, its value shall be subject to a constraint that depends on the type of the first parameter: if the first parameter is of the signed type (or is of a subrange type whose host type is the signed type), the value of the second parameter shall be less than or equal to the maximum value of the signed type; otherwise it shall be greater than or equal to zero.

NOTE — These constraints are present because signed arithmetic will be used if the first parameter is of the signed type, and unsigned arithmetic will be used in all other cases.

CLARIFICATION — In *Programming in Modula-2* the types of the parameters of INC are not specified.

wf-inc-designator : *Actual-parameters* → *Environment* → $\mathbb{B}$

wf-inc-designator $(aps)\rho \triangleq$
  is-Typed-variable-designator$(aps(1)) \wedge$
  cases *aps* :
    $[tdesig]$ → let *mk-Typed-variable-designator* $(desig, type) = tdesig$ in
      wf-dec-inc-var $(type, desig)\rho$, 282
    $[tdesig, expr]$ → let *mk-Typed-variable-designator* $(desig, type) = tdesig$ in
      wf-dec-inc-var $(type, desig)\rho \wedge$ 282
      wf-dec-inc-value $(type, expr)\rho$ 282
  end

Dynamic Semantics

The call INC(x) shall assign to the variable x the successor of its current value. An exception shall be raised if the successor lies outside the range of the type of x, or if there is no successor.

The call INC(x,N) shall assign to the variable x the Nth successor of its current value. An exception shall be raised if the Nth successor lies outside the range of the type of x, or if there is no N^{th} successor.

m-inc-designator : *Arguments* → *Environment* $\xrightarrow{o}$ ()

m-inc-designator $(args)\rho \triangleq$
  cases *args*:
    $[tvar]$ → let *mk-Typed-variable* $(var, type) = tvar$ in
      m-inc-dec$(type, var, 1)\ \rho$, 282
    $[tvar, val]$ → let *mk-Typed-variable-designator* $(var, type) = tvar$ in
      m-inc-dec$(type, var, val)\ \rho$ 282
  end

6.10.3.7 The Procedure INCL

The procedure INCL can be used to add an element to a variable that is of a set type or a packedset type.

Static Semantics

A call of INCL shall have two actual parameters. The first parameter shall be a variable that is of a set type or a packedset type, and the second parameter shall be an expression with which the the host type of the base type of the

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

set type or packedset type is assignment-compatible.

wf-incl-designator : *Actual-parameters* → *Environment* → 𝔹

wf-incl-designator (*aps*)ρ ≜
  len *aps* = 2 ∧
  let [*mk-Typed-variable-designator* (*desig*, *rtype*), *expr*] = *aps* in
  is-Variable-designator(*desig*) ∧
  is-Expression(*expr*) ∧
  rtype = *t-variable-designator* (*desig*)ρ ∧ 182
  is-set-type (*rtype*)ρ ∧ 322
  is-assignment-compatible (*host-type-of* (*base-type-of* (*rtype*)ρ)ρ, *t-expression* (*expr*)ρ)ρ 123 325 325

Dynamic Semantics

The call `INCL(s,x)` shall assign to the set variable `s` the value that is obtained by including the value of `x` in the current value of `s`. An exception shall be raised if the value of `x` is not a value of the base type of `s`.

m-incl-designator : *Arguments* → *Environment* $\xrightarrow{o}$ ()

m-incl-designator (*args*)ρ ≜
  let [*mk-Typed-variable* (*var*, *rtype*), *val*] = *args* in
  let *mk-Set-structure* (*btype*) = *structure-of* (*rtype*)ρ in 315
  let *range* = *values-of* (*btype*)ρ in 329
  if *val* ∈ *range*
  then def *value* = *variable-value*(*var*) in 224
       assign(*var*, *value* ∪ {*val*}) ρ 150
  else let *handler* = *current-exception-handler* (ρ) in 318
       mandatory-detection(INCLNOTBASE) handler 396

6.10.3.8 The Procedure NEW

The procedure `NEW` can often be used to obtain an area of dynamically allocated storage that is to be bound to a variable that is of a pointer type.

NOTE 1 — The word *often* appears because there is no requirement that the procedure `ALLOCATE` that is associated with `NEW` necessarily be the procedure `Storage.ALLOCATE`, or a procedure that performs a similar task.

Static Semantics

A call of `NEW` shall have one or more actual parameters. The first parameter shall be a variable that is of a pointer type.

If the second and subsequent parameters are present, they shall be constant expressions and the bound type of the first parameter shall be a record type.

The identifier `ALLOCATE` shall be visible at the point of the call of `NEW` and shall designate a procedure, procedure constant, or procedure variable with the structure of a proper procedure that has two formal parameters. The first formal parameter of this proper procedure structure shall be a variable parameter that is of the address type, and the second formal parameter shall be a value parameter that is of the unsigned type.

NOTE 2 — The separate module `Storage` exports a procedure called `ALLOCATE` of the correct structure for a call of `NEW`. The requirement that the identifier `ALLOCATE` is visible at the point of the call of `NEW` is often satisfied by including the import list:

```
FROM Storage IMPORT ALLOCATE;
```

in the module containing the call of `NEW`.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wf-new-designator : *Actual-parameters* → *Environment* → 𝔹

wf-new-designator (*aps*)ρ ≙
  len *aps* ≥ 1 ∧
  let [*mk-Typed-variable-designator* (*desig*, *rtype*)] ↷ *exprs* = *aps* in
  is-Variable-designator(*desig*) ∧
  rtype = *t-variable-designator* (*desig*)ρ ∧ 182
  is-pointer-type (*rtype*)ρ ∧ 322
  wf-value-designator (ALLOCATE)ρ ∧ 223
  let *typed* = *t-value-designator* (ALLOCATE)ρ in 223
  structure-of (*typed*)ρ = *storage-call* ∧ 315
  (*exprs* = [ ]
    ∨
   let *btype* = *bound-type-of* (*rtype*)ρ in 326
   let *rs* = *structure-of* (*btype*)ρ in 315
   is-Record-structure(*rs*) ∧
   len *exprs* ≥ 1 ∧
   is-tags-in-field-list (*rs*, *exprs*) ∧ 366
   ∀ *expr* ∈ elems *exprs* ·
     is-Expression(*expr*) ∧
     wf-expression (*expr*)ρ ∧ *is-constant-expression* (*expr*)ρ) 190 254

Dynamic Semantics

The call `NEW(P)` shall have the same effect as the call `ALLOCATE(P,SIZE(B))`, where the variable `P` is of a pointer type `T`, and `B` is the bound type given in the declaration of `T`.

The call `NEW(P,` < *expression-list* >`)` shall have the same effect as the call

`ALLOCATE(P,SYSTEM.TSIZE(B,` < *expression-list* >`))`,

where the variable `P` is of a pointer type `T`, and `B` is the bound type given in the declaration of `T`.

m-new-designator : *New-designator* → *Arguments* → *Environment* $\xrightarrow{o}$ ()

m-new-designator (*args*)ρ ≙
  let [*mk-Typed-variable* (*var*, *vtype*)] ↷ *vals* = *args* in
  let *size* = *tsize* (*vtype.type*, *vals*)ρ in 366
  def *procid* = *m-value-designator*(ALLOCATE) ρ in 223
  let *handler* = *current-exception-handler* (ρ) in 318
  call-proper-procedure-value(*procid*) ([*var*, *size*]) *handler* 339

6.10.4 Standard Functions

The standard functions are denoted by the pervasive identifiers `ABS`, `CAP`, `CHR`, `CMPLX`, `FLOAT`, `HIGH`, `IM`, `INT`, `LENGTH`, `LFLOAT`, `MAX`, `MIN`, `ODD`, `ORD`, `RE`, `SIZE`, `TRUNC` and `VAL`.

CHANGE — The functions `CMPLX`, `IM`, `INT`, `LENGTH`, `LFLOAT` and `RE` are not described in *Programming in Modula-2*.

6.10.4.1 Standard Function Designators

Abstract Syntax

types

Standard-function-designator :: *ident* : *Identifier*

inv *ident* ≙
  ident ∈ {ABS, CAP, CHR, CMPLX, FLOAT, HIGH, IM, INT, LENGTH, LFLOAT, MAX, MIN, ODD, ORD, RE, SIZE, TRUNC, VAL}

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Static Semantics

wf-standard-function-designator :
Standard-function-designator → *Actual-parameters* → *Environment* → 𝔹

wf-standard-function-designator (*desig*)(*aps*)ρ ≙
access-environment (*desig*)ρ = STANDARD-FUNCTION-PROCEDURE ∧ 319
cases *desig* :
ABS → *wf-abs-designator* (*aps*)ρ, 290
CAP → *wf-cap-designator* (*aps*)ρ, 291
CHR → *wf-chr-designator* (*aps*)ρ, 292
CMPLX → *wf-cmplx-designator* (*aps*)ρ, 293
FLOAT → *wf-float-designator* (*aps*)ρ, 294
HIGH → *wf-high-designator* (*aps*)ρ, 294
IM → *wf-im-designator* (*aps*)ρ, 295
INT → *wf-int-designator* (*aps*)ρ, 296
LENGTH → *wf-length-designator* (*aps*)ρ, 297
LFLOAT → *wf-lfloat-designator* (*aps*)ρ, 298
MAX → *wf-max-designator* (*aps*)ρ, 298
MIN → *wf-min-designator* (*aps*)ρ, 299
ODD → *wf-odd-designator* (*aps*)ρ, 300
ORD → *wf-ord-designator* (*aps*)ρ, 301
RE → *wf-re-designator* (*aps*)ρ, 301
SIZE → *wf-size-designator* (*aps*)ρ, 302
TRUNC → *wf-trunc-designator* (*aps*)ρ, 303
VAL → *wf-val-designator* (*aps*)ρ 305
end

t-standard-function-designator :
Standard-function-designator → *Actual-parameters* → *Environment* → *Expression-typed*

t-standard-function-designator (*desig*)(*aps*)ρ ≙
cases *desig* :
ABS → *t-abs-designator* (*aps*)ρ, 290
CAP → *t-cap-designator* (*aps*)ρ, 291
CHR → *t-chr-designator* (*aps*)ρ, 292
CMPLX → *t-cmplx-designator* (*aps*)ρ, 293
FLOAT → *t-float-designator* (*aps*)ρ, 294
HIGH → *t-high-designator* (*aps*)ρ, 294
IM → *t-im-designator* (*aps*)ρ, 295
INT → *t-int-designator* (*aps*)ρ, 296
LENGTH → *t-length-designator* (*aps*)ρ, 297
LFLOAT → *t-lfloat-designator* (*aps*)ρ, 298
MAX → *t-max-designator* (*aps*)ρ, 298
MIN → *t-min-designator* (*aps*)ρ, 299
ODD → *t-odd-designator* (*aps*)ρ, 300
ORD → *t-ord-designator* (*aps*)ρ, 301
RE → *t-re-designator* (*aps*)ρ, 302
SIZE → *t-size-designator* (*aps*)ρ, 303
TRUNC → *t-trunc-designator* (*aps*)ρ, 303
VAL → *t-val-designator* (*aps*)ρ 305
end

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Dynamic Semantics

m-standard-function-designator :
Standard-function-designator → *Arguments* → *Environment* $\xrightarrow{o}$ *Value*

m-standard-function-designator (*desig*)(*args*)ρ $\triangleq$
cases *desig*:
ABS → *m-abs-designator*(*args*) ρ, 291
CAP → *m-cap-designator*(*args*) ρ, 291
CHR → *m-chr-designator*(*args*) ρ, 292
CMPLX → *m-cmplx-designator*(*args*) ρ, 293
FLOAT → *m-float-designator*(*args*) ρ, 294
HIGH → *m-high-designator*(*args*) ρ, 295
IM → *m-im-designator*(*args*) ρ, 296
INT → *m-int-designator*(*args*) ρ, 297
LENGTH → *m-length-designator*(*args*) ρ, 297
LFLOAT → *m-lfloat-designator*(*args*) ρ, 298
MAX → *m-max-designator*(*args*) ρ, 299
MIN → *m-min-designator*(*args*) ρ, 300
ODD → *m-odd-designator*(*args*) ρ, 301
ORD → *m-ord-designator*(*args*) ρ, 301
RE → *m-re-designator*(*args*) ρ, 302
SIZE → *m-size-designator*(*args*) ρ, 303
TRUNC → *m-trunc-designator*(*args*) ρ, 304
VAL → *m-val-designator*(*args*) ρ 305
end

6.10.4.2 The Function ABS

The function ABS can be used to obtain the absolute value of a value of a real number type or a whole number type.

Static Semantics

A call of ABS shall have one actual parameter, which shall be an expression that is of a real number type or a whole number type, except that the expression shall not be of the unsigned type.

The type of a call of ABS shall be the same type as the expression.

CLARIFICATION — In *Programming in Modula-2* the type of the parameter of ABS is not specified.

NOTE 1 — An expression that is a value designator that is of a subrange type is regarded as being of the host type of the subrange type. A call of ABS applied to such an expression returns a value of the host type.

wf-abs-designator : *Actual-parameters* → *Environment* → $\mathbb{B}$

wf-abs-designator (*aps*)ρ $\triangleq$
len *aps* = 1 ∧
let [*mk-Typed-expression* (*expr*, *rtype*)] = *aps* in
is-Expression(*expr*) ∧
(*rtype* = SIGNED-TYPE ∨ *rtype* = $\mathbb{Z}$-TYPE ∨ *is-Real-number-type*(*rtype*)) ∧
t-expression (*expr*)ρ = *rtype* 190

t-abs-designator : *Actual-parameters* → *Environment* → *Expression-typed*

t-abs-designator (*aps*)ρ $\triangleq$
let [*mk-Typed-expression* (-, *rtype*)] = *aps* in
rtype

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Dynamic Semantics

The value of the call `ABS(x)` shall be the absolute value of `x`. An exception shall be raised if this value lies outside the range of the type of `x`.

NOTE 2 — An exception occurs if the value of `x` is less than zero and the corresponding positive value does not belong to the type of `x` — this can only occur if this type has a non-symmetric range.

m-abs-designator : *Arguments* → *Environment* $\xrightarrow{o}$ *Value*

m-abs-designator (*args*)ρ $\triangleq$
  let [*mk-Typed-value* (*arg*, *rtype*)] = *args* in
  let *handler* = *current-exception-handler* (ρ) in 318
  (*arg* $\geq$ 0 → return arg ,
   rtype = SIGNED-TYPE → *get-whole-result*(*rtype*, −*arg*) *handler*, 199
   rtype = $\mathbb{Z}$-TYPE → *get-whole-result*(*rtype*, −*arg*) *handler*, 199
   is-Real-number-type(*rtype*) → *get-real-result*(*rtype*, −*arg*) *handler*) 197

6.10.4.3 The Function CAP

The function `CAP` can be used to obtain the upper-case equivalent of a lower-case character for those characters that have such equivalents.

Static Semantics

A call of `CAP` shall have one actual parameter, which shall be an expression that is of the character type.

The type of a call of `CAP` shall be the character type.

wf-cap-designator : *Actual-parameters* → *Environment* → $\mathbb{B}$

wf-cap-designator (*mk-Cap-designator* ())(*aps*)ρ $\triangleq$
  len *aps* = 1 ∧
  let [*expr*] = *aps* in
  is-Expression(*expr*) ∧
  t-expression (*expr*)ρ = CHARACTER-TYPE 190

t-cap-designator : *Actual-parameters* → *Environment* → *Typed*

t-cap-designator (*mk-Cap-designator* ())(*aps*)ρ $\triangleq$
  CHARACTER-TYPE

Dynamic Semantics

If the value of `c` is a lower-case letter, the value of the call `CAP(c)` shall be the implementation-defined upper-case letter corresponding to the value of `c`; otherwise it shall be the value of `c`.

m-cap-designator : *Arguments* → *Environment* $\xrightarrow{o}$ *Value*

m-cap-designator (*args*)ρ $\triangleq$
  let [*ch*] = *args* in
  if *ch* ∈ dom *capitalizations*
  then return *capitalizations* (*ch*)
  else return *ch*

CLARIFICATION — In *Programming in Modula-2* the behaviour of `CAP` is only specified when the parameter is a letter. However, some examples in *Programming in Modula-2* assume that `CAP` is defined for all values of the character type. This part of ISO/IEC 10514 defines `CAP` for all values of the character type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Auxiliaries

values

$$required\text{-}capitalizations = \{required\text{-}lower\text{-}cases\text{-}order(i) \mapsto required\text{-}upper\text{-}cases\text{-}order(i) \mid i \in \mathsf{inds}\ required\text{-}lower\text{-}cases\text{-}order\};$$

$$implementation\text{-}capitalizations = \{implementation\text{-}lower\text{-}cases\text{-}order(i) \mapsto implementation\text{-}upper\text{-}cases\text{-}order(i) \mid i \in \mathsf{inds}\ implementation\text{-}lower\text{-}cases\text{-}order\};$$

$$capitalizations = required\text{-}capitalizations \cup implementation\text{-}capitalizations$$

annotations Defining *capitalizations* in this way avoids a dependency on the collating sequence of the character values. *implementation-capitalizations* is a means by which implementations may have extra capitalization rules (e.g. 'γ' → 'Γ' or 'ü' → 'Ü).

6.10.4.4 The Function CHR

The function CHR can be used to obtain the value of the character type that has a specified ordinal number.

Static Semantics

A call of CHR shall have one actual parameter, which shall be an expression that is of a whole number type.

The type of a call of CHR shall be the character type.

$wf\text{-}chr\text{-}designator : Actual\text{-}parameters \to Environment \to B$

$wf\text{-}chr\text{-}designator\ (aps)\rho \triangleq$
  len $aps = 1 \wedge$
  let $[expr] = aps$ in
  $is\text{-}Expression(expr) \wedge$
  $is\text{-}whole\text{-}number\text{-}type\ (t\text{-}expression\ (expr)\rho)\rho$ 323 190

$t\text{-}chr\text{-}designator : Actual\text{-}parameters \to Environment \to Typed$

$t\text{-}chr\text{-}designator\ (aps)\rho \triangleq$
  CHARACTER-TYPE

Dynamic Semantics

The value of the call CHR(x) shall be the same as the value of the call VAL(CHAR,x) (see 6.10.4.19).

NOTE — An exception occurs if the value of x is less than zero, or if it lies outside the range of the ordinal numbers of the character type.

$m\text{-}chr\text{-}designator : Arguments \to Environment \xrightarrow{o} Value$

$m\text{-}chr\text{-}designator\ (args)\rho \triangleq$
  let $[value] = args$ in
  $converted\text{-}result(\text{CHARACTER-TYPE}, value)\ \rho$ 306

6.10.4.5 The Function CMPLX

The function CMPLX can be used to obtain a value of a complex number type from a pair of expressions specifying its real and imaginary parts.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Static Semantics

A call of CMPLX shall have two actual parameters. Each parameter shall be an expression, which shall be of a real number type.

If both expressions are of the real type, or if one is of the real type and the other is of the ℝ-type, the type of a call of CMPLX shall be the complex type. If both expressions are of the long real type, or if one is of the long real type and the other is of the ℝ-type, the type of the call shall be the long complex type. If both expressions are of the ℝ-type, the type of the call shall be the ℂ-type.

NOTE 1 — The following table gives the permitted types and the type of the call.

	real type	long real type	ℝ-type
real type	complex type	error	complex type
long real type	error	long complex type	long complex type
ℝ-type	complex type	long complex type	ℂ-type

wf-cmplx-designator : *Actual-parameters* → *Environment* → 𝔹

wf-cmplx-designator (*aps*)ρ ≙
  len *aps* = 2 ∧
  let [*re-texpr*, *im-texpr*] = *aps* in
  is-Typed-expression(*re-texpr*) ∧ *is-Typed-expression*(*im-texpr*) ∧
  let *mk-Typed-expression* (*re-expr*, *re-type*) = *re-texpr* in
  let *mk-Typed-expression* (*im-expr*, *im-type*) = *im-texpr* in
  is-Expression(*re-expr*) ∧ *is-Real-number-type*(*t-expression* (*re-expr*)ρ) ∧ 190
  is-Expression(*im-expr*) ∧ *is-Real-number-type*(*t-expression* (*im-expr*)ρ) ∧ 190
  re-type = *im-type* ∧
  re-type = *complex-result-type* (*t-expression* (*re-expr*)ρ, *t-expression* (*im-expr*)ρ) 194 190 190

t-cmplx-designator : *Actual-parameters* → *Environment* → *Expression-typed*

t-cmplx-designator (*aps*)ρ ≙
  let [*mk-Typed-expression* (-, *re-type*), -] = *aps* in
  re-type

Dynamic Semantics

The value of the call CMPLX(x,y) shall be an implementation-defined approximation to the complex number whose real part is the value of x and whose imaginary part is the value of y.

NOTE 2 — An exception occurs if this approximation lies outside the range of the type of the call.

m-cmplx-designator : *Arguments* → *Environment* $\xrightarrow{o}$ *Value*

m-cmplx-designator (*args*)ρ ≙
  let [*mk-Typed-Value* (*x*, *re-type*), *mk-Typed-Value* (*y*, -)] = *args* in
  let *handler* = *current-exception-handler* (ρ) in 318
  get-complex-result(*re-type*, *x* + i*y*) *handler* 194

6.10.4.6 The Function FLOAT

The function FLOAT can be used to obtain a value of the real type from a value of a real number type or a whole number type.

Static Semantics

A call of FLOAT shall have one actual parameter, which shall be an expression that is of a real number type or a whole number type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The type of a call of FLOAT shall be the real type.

wf-float-designator : *Actual-parameters* → *Environment* → 𝔹

wf-float-designator (*aps*)ρ ≙
  len *aps* = 1 ∧
  let [*expr*] = *aps* in
  is-Expression(*expr*) ∧
  is-number-type (*t-expression* (*expr*)ρ)ρ 324 190

t-float-designator : *Actual-parameters* → *Environment* → *Typed*

t-float-designator (*aps*)ρ ≙
  REAL-TYPE

Dynamic Semantics

The value of the call FLOAT(x) shall be the same as the value of the call VAL(REAL,x) (see 6.10.4.19).

NOTE — An exception occurs if the value of x lies outside the range of the real type.

m-float-designator : *Arguments* → *Environment* $\xrightarrow{o}$ *Value*

m-float-designator (*args*)ρ ≙
  let [*value*] = *args* in
  converted-result(REAL-TYPE, *value*) ρ 306

6.10.4.7 The Function HIGH

The function HIGH can be used to obtain the upper bound of the index type of a one-dimensional open array formal parameter, or the upper bound of an index type of a multi-dimensional open array formal parameter.

CHANGE — Multi-dimensional open array parameters are not described in *Programming in Modula-2*.

CLARIFICATION — This part of ISO/IEC 10514 permits HIGH to be applied only to an open array parameter; it does not permit HIGH to be applied to an ordinary array variable.

Static Semantics

A call of HIGH shall have one actual parameter, which shall be a designator of a variable that is of an open array formal type.

NOTE — A variable that is of an open array formal type is either an open array formal parameter or a component of an open array formal parameter that has an open array component type.

The type of a call of HIGH shall be the unsigned type.

wf-high-designator : *Actual-parameters* → *Environment* → 𝔹

wf-high-designator (*aps*)ρ ≙
  len *aps* = 1 ∧
  let [*var*] = *aps* in
  is-Variable-designator(*var*) ∧
  wf-high-argument (*var*)ρ 295

t-high-designator : *Actual-parameters* → *Environment* → *Typed*

t-high-designator (*mk-High-designator* ())(*aps*)ρ ≙
  UNSIGNED-TYPE

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Dynamic Semantics

The value of the call `HIGH(x)` shall be the upper bound of the index type of an array type that has the same number of components of the same component type as `x`, but has an index type declared as a subrange of the unsigned type with a lower bound of zero.

An exception shall be raised if the value to be returned lies outside the range of the unsigned type.

EXAMPLE — Result of `HIGH`.

Given:

```
TYPE colour = (red, yellow, green);

VAR a: ARRAY [-7..-2] OF ARRAY BOOLEAN
                      OF ARRAY colour OF REAL;

PROCEDURE p (f: ARRAY OF ARRAY OF ARRAY OF REAL);
```

then, for a call of `HIGH` from within the body of `p` that was activated by '`p(a)`'

expression	result
`HIGH(f)`	5
`HIGH(f[0])`	1
`HIGH(f[0,0])`	2
`HIGH(f[0,0,0])`	error

m-high-designator : *Arguments* → *Environment* $\xrightarrow{o}$ *Value*

m-high-designator (*args*)ρ $\triangleq$
  let [*array*] = *args* in
  let *value* = card (dom *array*) − 1 in
  let *handler* = *current-exception-handler* (ρ) in 318
  get-whole-result(UNSIGNED-TYPE, *value*) *handler* 199

Auxiliaries

wf-high-argument : *Variable-designator* → *Environment* → $\mathbb{B}$

wf-high-argument (*aps*)ρ $\triangleq$
  (*is-Entire-designator*(*aps*) → let *type* = *t-entire-designator* (*aps*)ρ in 183
    is-open-array (*type*)ρ, 328
   is-Indexed-designator(*aps*) → *wf-high-argument* (*aps.desig*)ρ) 295

6.10.4.8 The Function IM

The function `IM` can be used to obtain the imaginary part of a value of a complex number type.

Static Semantics

A call of `IM` shall have one actual parameter, which shall be an expression that is of a complex number type.

If the expression is of the complex type, the type of a call of `IM` shall be the real type; if the expression is of the long complex type, the type of the call shall be the long real type; otherwise the expression shall be of the $\mathbb{C}$-type, and the type of the call shall be the $\mathbb{R}$-type.

wf-im-designator : *Actual-parameters* → *Environment* → $\mathbb{B}$

wf-im-designator (*aps*)ρ $\triangleq$
  len *aps* = 1 ∧
  let [*texpr*] = *aps* in
  is-Typed-expression(*texpr*) ∧
  let *mk-Typed-expression* (*expr*, *rtype*) = *texpr* in
  is-Expression(*expr*) ∧
  let *etype* = *t-expression* (*expr*)ρ in 190
  is-Complex-number-type(*etype*) ∧
  rtype = *complex-component-type* (*etype*) 296

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

t-im-designator : *Actual-parameters* → *Environment* → *Expression-typed*

t-im-designator (*aps*)ρ $\triangleq$
  let [*mk-Typed-expression* (-, *rtype*)] = *aps* in
  rtype

Auxiliaries

complex-component-type : *Complex-number-type* → *Real-number-type*

complex-component-type (*type*) $\triangleq$
  cases *type* :
    COMPLEX-TYPE → REAL-TYPE,
    LONG-COMPLEX-TYPE → LONG-REAL-TYPE,
    $\mathbb{C}$-TYPE → $\mathbb{R}$-TYPE
  end

Dynamic Semantics

The value of the call `IM(z)` shall be an implementation-defined approximation to the imaginary part of `z`.

m-im-designator : *Arguments* → *Environment* $\xrightarrow{o}$ *Value*

m-im-designator (*args*)ρ $\triangleq$
  let [*mk-Typed-value* (*value*, *rtype*)] = *args* in
  let *handler* = *current-exception-handler* (ρ) in 318
  get-real-result(*rtype*, im*value*) *handler* 197

6.10.4.9 The Function INT

The function `INT` can be used to obtain a value of the signed type from a value of an ordinal type or a real number type.

Static Semantics

A call of `INT` shall have one actual parameter, which shall be an expression that is of an ordinal type or a real number type.

The type of a call of `INT` shall be the signed type.

wf-int-designator : *Actual-parameters* → *Environment* → $\mathbb{B}$

wf-int-designator (*mk-Int-designator* ())(*aps*)ρ $\triangleq$
  len *aps* = 1 $\wedge$
  let [*expr*] = *aps* in
  is-Expression(*expr*) $\wedge$
  let *etype* = *t-expression* (*expr*)ρ in 190
  is-ordinal-type (*etype*)ρ $\vee$ *is-real-number-type*(*etype*) 322

t-int-designator : *Actual-parameters* → *Environment* → *Typed*

t-int-designator (*mk-Int-designator* ())(*aps*)ρ $\triangleq$
  SIGNED-TYPE

Dynamic Semantics

The value of the call `INT(x)` shall be the same as the value of the call `VAL(INTEGER,x)` (see 6.10.4.19).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NOTE — An exception occurs if the value to be returned lies outside the range of the signed type.

m-int-designator : *Arguments* → *Environment* $\xrightarrow{o}$ *Value*

m-int-designator (*args*)ρ $\triangleq$
  let [*value*] = *args* in
  converted-result(SIGNED-TYPE, *value*) ρ 306

6.10.4.10 The Function LENGTH

The function LENGTH can be used to obtain the length of a string value.

Static Semantics

A call of LENGTH shall have one actual parameter, which shall be an expression that is of a string literal type or is an array that contains a string value.

The type of a call of LENGTH shall be the unsigned type.

wf-length-designator : *Actual-parameters* → *Environment* → $\mathbb{B}$

wf-length-designator (*aps*)ρ $\triangleq$
  len *aps* = 1 ∧
  let [*expr*] = *aps* in
  is-Expression(*expr*) ∧
  let *etype* = *t-expression* (*expr*)ρ in 190
  is-String-type(*etype*)
    ∨
  is-array-type (*etype*)ρ ∧ 321
  component-type-of (*etype*)ρ = CHARACTER-TYPE 325

t-length-designator : *Actual-parameters* → *Environment* → *Typed*

t-length-designator (*aps*)ρ $\triangleq$
  UNSIGNED-TYPE

Dynamic Semantics

The value of the call LENGTH(s) shall be the number of characters in the string value s.

m-length-designator : *Arguments* → *Environment* $\xrightarrow{o}$ *Value*

m-length-designator (*args*)ρ $\triangleq$
  let [*value*] = *args* in
  if END-OF-STRING-CHAR ∈ rng *value*
  then let *end* = *mins* ({*i* | *i* ∈ dom *value* · *value*(*i*) = END-OF-STRING-CHAR}) in 614
      return *end*
  else return (card dom value)

6.10.4.11 The Function LFLOAT

The function LFLOAT can be used to obtain a value of the long real type from a value of a real number type or a whole number type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Static Semantics

A call of LFLOAT shall have one actual parameter, which shall be an expression that is of a real number type or a whole number type.

The type of a call of LFLOAT shall be the long real type.

wf-lfloat-designator : *Actual-parameters* → *Environment* → 𝔹
wf-lfloat-designator (*aps*)ρ ≙
  len *aps* = 1 ∧
  let [*expr*] = *aps* in
  is-Expression(*expr*) ∧
  is-number-type (*t-expression* (*expr*)ρ)ρ 324 190

t-lfloat-designator : *Actual-parameters* → *Environment* → *Typed*
t-lfloat-designator (*aps*)ρ ≙
  LONG-REAL-TYPE

Dynamic Semantics

The value of the call LFLOAT(x) shall be the same as the value of the call VAL(LONGREAL,x) (see 6.10.4.19).

NOTE — An exception occurs if the value of x lies outside the range of the long real type.

m-lfloat-designator : *Arguments* → *Environment* $\xrightarrow{o}$ *Value*
m-lfloat-designator (*args*)ρ ≙
  let [*value*] = *args* in
  converted-result(LONG-REAL-TYPE, *value*) ρ 306

6.10.4.12 The Function MAX

The function **MAX** can be used to obtain the maximum value of a scalar type.

Static Semantics

A call of **MAX** shall have one actual parameter, which shall be a type parameter that denotes a scalar type.

If the type denoted by the type parameter is a real number type or a whole number type, or the host type of the denoted type is a whole number type (in the case where the denoted type is a subrange type), the type of a call of **MAX** shall be the type of a literal value of that number type; otherwise the type of the call shall be the host type of the type denoted by the type parameter.

CLARIFICATION — The function **MAX** may be applied to any scalar type (including the real number types denoted by **REAL** and **LONGREAL**).

wf-max-designator : *Actual-parameters* → *Environment* → 𝔹
wf-max-designator (*aps*)ρ ≙
  len *aps* = 1 ∧
  let [*mk-Type-parameter* (*qid*)] = *aps* in
  is-type (*qid*)ρ ∧ 314
  let *type* = *type-of* (*qid*)ρ in 314
  is-scalar-type (*type*)ρ 322

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

t-max-designator : *Actual-parameters* → *Environment* → *Expression-typed*

t-max-designator (*mk-Max-designator* ())(*aps*)ρ $\triangleq$
  let [*mk-Type-parameter* (*qid*)] = *aps* in
  let *type* = *type-of*(*qid*)ρ in
  let *host* = *host-type-of*(*type*)ρ in
  (*host* ∈ {UNSIGNED-TYPE, SIGNED-TYPE} → $\mathbb{Z}$-TYPE,
   host ∈ {REAL-TYPE, LONG-REAL-TYPE} → $\mathbb{R}$-TYPE,
   others → *host*)

Dynamic Semantics

The value of the call `MAX(T)` shall be the largest value of the type denoted by `T`.

EXAMPLE — Result of `MAX`.

Given:

```
TYPE
  percentage = CARDINAL[0..100];
  countdown = INTEGER[-9..-1];
  colours = (red, yellow, green);
  coward = colours[yellow..yellow];
```

then:

expression	value	type
`MAX(percentage)`	100	$\mathbb{Z}$-type
`MAX(countdown)`	-1	$\mathbb{Z}$-type
`MAX(colours)`	`green`	`colours`
`MAX(coward)`	`yellow`	`colours`

'`MAX(CARDINAL)`' and '`MAX(INTEGER)`' are also of the $\mathbb{Z}$-type.

m-max-designator : *Arguments* → *Environment* $\xrightarrow{o}$ *Value*

m-max-designator (*args*)ρ $\triangleq$
  let [*type*] = *args* in
  return *maximum* (*type*)ρ 328

6.10.4.13 The Function MIN

The function `MIN` can be used to obtain the minimum value of a scalar type.

Static Semantics

A call of `MIN` shall have one actual parameter, which shall be a type parameter that denotes a scalar type.

If the type denoted by the type parameter is a real number type or a whole number type, or the host type of the denoted type is a whole number type (in the case where the denoted type is a subrange type), the type of a call of `MIN` shall be the type of a literal value of that number type; otherwise the type of the call shall be the host type of the type denoted by the type parameter.

CLARIFICATION — The function `MIN` may be applied to any scalar type (including the real number types denoted by `REAL` and `LONGREAL`).

wf-min-designator : *Actual-parameters* → *Environment* → $\mathbb{B}$

wf-min-designator (*aps*)ρ $\triangleq$
  len *aps* = 1 ∧
  let [*mk-Type-parameter* (*qid*)] = *aps* in
  is-type (*qid*)ρ ∧ 314
  let *type* = *type-of* (*qid*)ρ in 314
  is-scalar-type (*type*)ρ 322

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

t-min-designator : *Actual-parameters* → *Environment* → *Expression-typed*

t-min-designator (*mk-Min-designator* ())(*aps*)ρ ≜
  let [*mk-Type-parameter* (*qid*)] = *aps* in
  let *type* = *type-of* (*qid*)ρ in 314
  let *host* = *host-type-of* (*type*)ρ in 325
  (*host* ∈ {UNSIGNED-TYPE, SIGNED-TYPE} → $\mathbb{Z}$-TYPE,
   host ∈ {REAL-TYPE, LONG-REAL-TYPE} → $\mathbb{R}$-TYPE,
   others → *host*)

Dynamic Semantics

The value of the call `MIN(T)` shall be the smallest value of the type denoted by `T`.

NOTE — The value of the call `MIN(R)`, where `R` denotes a real number type, is the largest negative value of that type, and not the smallest non-zero positive value of that type. The smallest non-zero positive values of the real type and the long real type are defined by the values of the constants `LowReal.small` and `LowLong.small` respectively — see 8.2.

EXAMPLE — Result of `MIN`.

Given:

```
TYPE
  percentage = CARDINAL[0..100];
  countdown = INTEGER[-9..-1];
  colours = (red, yellow, green);
  coward = colours[yellow..yellow];
```

then:

expression	value	type
`MIN(percentage)`	0	$\mathbb{Z}$-type
`MIN(countdown)`	-9	$\mathbb{Z}$-type
`MIN(colours)`	red	colours
`MIN(coward)`	yellow	colours

'`MIN(CARDINAL)`' and '`MIN(INTEGER)`' are also of the $\mathbb{Z}$-type.

m-min-designator : *Arguments* → *Environment* $\xrightarrow{o}$ *Value*

m-min-designator (*args*)ρ ≜
  let [*type*] = *args* in
  return *minimum* (*type*)ρ 329

6.10.4.14 The Function ODD

The function `ODD` can be used to determine whether a value of a whole number type is even or odd.

Static Semantics

A call of `ODD` shall have one actual parameter, which shall be an expression that is of a whole number type.

The type of a call of `ODD` shall be the Boolean type.

wf-odd-designator : *Actual-parameters* → *Environment* → $\mathbb{B}$

wf-odd-designator (*aps*)ρ ≜
  len *aps* = 1 ∧
  let [*expr*] = *aps* in
  is-Expression(*expr*) ∧
  is-whole-number-type (*t-expression* (*expr*)ρ)ρ 323 190

t-odd-designator : *Actual-parameters* → *Environment* → *Typed*

t-odd-designator (*aps*)ρ ≜
  BOOLEAN-TYPE

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Dynamic Semantics

If the absolute value of x is odd, the value of the call ODD(x) shall be the value *true*; otherwise it shall be the value *false*.

m-odd-designator : *Arguments* → *Environment* $\xrightarrow{o}$ *Value*

m-odd-designator (*args*)ρ $\triangleq$
  let [*arg*] = *args* in
  return $\exists\, n \in \mathbb{Z} \cdot 2 \times n + 1 = arg$

6.10.4.15 The Function ORD

The function ORD can be used to obtain the ordinal number of a value of an ordinal type.

Static Semantics

A call of ORD shall have one actual parameter, which shall be an expression that is of an ordinal type.

The type of a call of ORD shall be the unsigned type.

wf-ord-designator : *Actual-parameters* → *Environment* → $\mathbb{B}$

wf-ord-designator (*aps*)ρ $\triangleq$
  len *aps* = 1 ∧
  let [*expr*] = *aps* in
  is-Expression(*expr*) ∧
  is-ordinal-type (*t-expression* (*expr*)ρ)ρ 322 190

t-ord-designator : *Actual-parameters* → *Environment* → *Typed*

t-ord-designator (*aps*)ρ $\triangleq$
  UNSIGNED-TYPE

Dynamic Semantics

The value of the call ORD(x) shall be the same as the value of the call VAL(CARDINAL,x) (see 6.10.4.19).

NOTE — An exception occurs if the value to be returned lies outside the range of the unsigned type.

m-ord-designator : *Arguments* → *Environment* $\xrightarrow{o}$ *Value*

m-ord-designator (*args*)ρ $\triangleq$
  let [*value*] = *args* in
  converted-result(UNSIGNED-TYPE, *value*) ρ 306

6.10.4.16 The Function RE

The function RE can be used to obtain the real part of a value of a complex number type.

Static Semantics

A call of RE shall have one actual parameter, which shall be an expression that is of a complex number type.

If the expression is of the complex type, the type of a call of RE shall be the real type; if the expression is of the long complex type, the type of the call shall be the long real type; otherwise the expression shall be of the $\mathbb{C}$-type, and the type of the call shall be the $\mathbb{R}$-type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wf-re-designator : *Re-designator* → *Actual-parameters* → *Environment* → $\mathbb{B}$

wf-re-designator (*aps*)ρ $\triangleq$
  len *aps* = 1 $\wedge$
  let [*texpr*] = *aps* in
  is-Typed-expression(*texpr*) $\wedge$
  let *mk-Typed-expression* (*expr*, *rtype*) = *texpr* in
  is-Expression(*expr*) $\wedge$
  let *etype* = *t-expression* (*expr*)ρ in 190
  is-Complex-number-type(*etype*) $\wedge$
  rtype = *complex-component-type* (*etype*) 296

t-re-designator : *Actual-parameters* → *Environment* → *Expression-typed*

t-re-designator (*aps*)ρ $\triangleq$
  let [*mk-Typed-expression* (-, *rtype*)] = *aps* in
  rtype

Dynamic Semantics

The value of the call `RE(z)` shall be an implementation-defined approximation to the real part of `z`.

m-re-designator : *Arguments* → *Environment* $\xrightarrow{o}$ *Value*

m-re-designator (*args*)ρ $\triangleq$
  let [*mk-Typed-value* (*value*, *rtype*)] = *args* in
  let *handler* = *current-exception-handler* (ρ) in 318
  get-real-result(*rtype*, revalue) *handler* 197

6.10.4.17 The Function SIZE

The function `SIZE` can be used to ascertain the amount of storage that is used for a variable of a particular type.

Static Semantics

A call of `SIZE` shall have one actual parameter, which shall either be a type parameter, or an entire designator that does not denote an open array parameter.

The type of a call of `SIZE` shall be the $\mathbb{Z}$-type.

CHANGE — In *Programming in Modula-2* the parameter may be the identifier of a type or a variable. There is no constraint that the variable is an entire designator that does not denote an open array parameter.

NOTES

1 If the parameter to `SIZE` is a variable designator, it must be an entire designator: it cannot be a component of an array, a field of a record, or the variable referenced by a pointer variable.

2 The constraints on the parameter ensure that a call of `SIZE` is always permitted in a constant expression.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wf-size-designator : *Actual-parameters* → *Environment* → 𝔹

wf-size-designator (*aps*)ρ ≙
  len *aps* = 1 ∧
  cases *aps* :
    [*mk-Typed-variable-designator* (*desig*, *type*)] → *is-Entire-variable*(*desig*) ∧
                                                        type = *t-variable-designator* (*desig*)ρ ∧ 182
                                                        ¬ *is-open-array* (*type*)ρ, 328
    [*mk-Type-parameter* (*qid*)]   → *is-type* (*qid*)ρ 314
  end

t-size-designator : *Actual-parameters* → *Environment* → *Expression-typed*

t-size-designator (*aps*)ρ ≙
  ℤ-TYPE

Dynamic Semantics

The value of the call `SIZE(T)`, where `T` denotes any identified type, shall be the number of storage locations (see 7.1.3.1) that would be occupied by a variable that is of type `T`. If `T` denotes a record type with a group or groups of variant fields, the value of the call shall be the maximum number of storage locations that could be occupied. An exception shall be raised if this number lies outside the range of the ℤ-type.

The value of the call `SIZE(v)`, where `v` is an entire designator of any type, shall be the same as the value of a call `SIZE(T)` with an actual parameter `T` that is a qualified identifier denoting the type of `v`.

NOTE 3 — In calls of the form `SIZE(v)`, `v` may designate a variable of an anonymous type.

m-size-designator : *Arguments* → *Environment* $\xrightarrow{o}$ *Value*

m-size-designator (*args*)ρ ≙
  let [*arg*] = *args* in
  let *anytyped* = cases *arg* :
                     mk-Typed-variable (-, *typed*) → *typed*,
                     mk-Typed-type (*typed*)   → *typed*
                   end in
  let *size* = *tsize* (*anytyped*, [])ρ in 366
  let *handler* = *current-exception-handler* (ρ) in 318
  get-whole-result(ℤ-TYPE, *size*) *handler* 199

6.10.4.18 The Function TRUNC

The function `TRUNC` can be used to obtain a value of the unsigned type from a value of a real number type.

Static Semantics

A call of `TRUNC` shall have one actual parameter, which shall be an expression that is of a real number type.

The type of a call of `TRUNC` shall be the unsigned type.

wf-trunc-designator . *Actual-parameters* → *Environment* → 𝔹

wf-trunc-designator (*aps*)ρ ≙
  len *aps* = 1 ∧
  let [*expr*] = *aps* in
  is-Expression(*expr*) ∧
  is-Real-number-type(*t-expression* (*expr*)ρ) 190

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

t-trunc-designator : *Actual-parameters* → *Environment* → *Typed*

t-trunc-designator (*aps*)ρ $\triangleq$
UNSIGNED-TYPE

Dynamic Semantics

The value of the call `TRUNC(x)` shall be the same as the value of the call `VAL(CARDINAL,x)` (see 6.10.4.19).

NOTE — An exception occurs if the integral part of `x` lies outside the range of the unsigned type.

m-trunc-designator : *Arguments* → *Environment* $\xrightarrow{o}$ *Value*

m-trunc-designator (*args*)ρ $\triangleq$
let [*value*] = *args* in
converted-result(UNSIGNED-TYPE, *value*) ρ 306

6.10.4.19 The Function VAL

The function `VAL` can be used to obtain a value of a specified scalar type from an expression of a scalar type.

CHANGE — The definition of `VAL` given in *Programming in Modula-2* has been extended.

Static Semantics

A call of `VAL` shall have two actual parameters. The first parameter shall be a type parameter that denotes a scalar type. If the type denoted by the type parameter is a subrange type, the type of a call of `VAL` shall be the host type of the subrange type; otherwise it shall be the type denoted by the type parameter. The type of the call is called the final-type.

The second parameter shall be an expression that is of a scalar type, subject to the restriction that at least one of the following shall hold:

a) The final-type and the type of the expression are identical.

b) The final-type is a whole number type or a real number type, and the type of the expression is a whole number type or a real number type.

c) The final-type is a whole number type.

d) The type of the expression is a whole number type.

NOTE 1 — In the following table, √ denotes a valid combination of types, and × denotes an invalid combination; `T` is some enumeration type.

the type of the expression	the type denoted by the type parameter						
	`CARDINAL`*	`INTEGER`*	`REAL`	`LONGREAL`	`CHAR`*	`BOOLEAN`*	the type `T`*
`CARDINAL`	√	√	√	√	√	√	√
`INTEGER`	√	√	√	√	√	√	√
ℤ-type	√	√	√	√	√	√	√
`REAL`	√	√	√	√	×	×	×
`LONGREAL`	√	√	√	√	×	×	×
ℝ-type	√	√	√	√	×	×	×
`CHAR`†	√	√	×	×	√	×	×
`BOOLEAN`	√	√	×	×	×	√	×
the type `T`	√	√	×	×	×	×	√

* or a subrange of this type
† or string literal type of length 0 or 1

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wf-val-designator : *Actual-parameters* → *Environment* → 𝔹

wf-val-designator (*aps*)ρ ≙
  len *aps* = 2 ∧
  let [*type*, *expr*] = *aps* in
  is-scalar-type (*type*)ρ ∧ 322
  is-Expression(*expr*) ∧
  let *etype* = *t-expression* (*expr*)ρ in 190
  is-scalar-type (*etype*)ρ ∧ 322
  let *htype* = *host-type-of* (*type*)ρ in 325
  (*htype* = *etype*
    ∨
  is-number-type (*type*)ρ ∧ *is-number-type* (*etype*)ρ 324 324
    ∨
  is-whole-number-type (*htype*)ρ 323
    ∨
  is-whole-number-type (*etype*)ρ) 323

t-val-designator : *Actual-parameters* → *Environment* → *Typed*

t-val-designator (*aps*)ρ ≙
  let [*type*, -] = *aps* in
  host-type-of (*type*)ρ 325

Dynamic Semantics

The value of the call `VAL(R,x)`, where the type `R` is a real number type, shall be a value of the type `R` that is an implementation-defined approximation to the value of `x`. An exception shall be raised if the value of `x` lies outside the range of the type `R`.

The value of the call `VAL(T,x)`, where the type `T` is an ordinal type, the host type of this type is a whole number type, and the type of the expression `x` is a whole number type or a real number type, shall be the integral part of the value of `x`. An exception shall be raised if the integral part of `x` lies outside the range of the type `T`.

The value of the call `VAL(T,x)`, where the type `T` or the type of `x` (or both of these types) are not whole number types or real number types, shall be the value of the type `T` that has an ordinal number equal to the ordinal number of the value of `x`. An exception shall be raised if the value of `x` is a numerical value that is less than zero, or if the ordinal number of the value of `x` lies outside the range of the ordinal numbers of the type `T`.

NOTE 2 — `VAL` does not round. If `x` is of a real number type, the expressions `VAL(INTEGER,x)` and `VAL(CARDINAL,x)` both have the value of `x` truncated.

EXAMPLE — Result of `VAL`.

Given:

```
TYPE
  Days = (Sun, Mon, Tue, Wed, Thu, Fri, Sat);
  WeekDays = [Mon..Fri];
VAR
  i, j, z: INTEGER;
  r: REAL;
```

and:

```
i := 42;  j := -1;  z := 0;  r := -2.7;
```

then:

expression	value	type
`VAL(CARDINAL,i)`	42	`CARDINAL`
`VAL(CARDINAL,j)`	Exception	
`VAL(CARDINAL,r)`	Exception	
`VAL(INTEGER,i)`	42	`INTEGER`
`VAL(INTEGER,r)`	-2	`INTEGER`
`VAL(REAL,i)`	42.0	`REAL`
`VAL(REAL,TRUE)`	Error	
`VAL(LONGREAL,r)`	-2.7	`LONGREAL`
`VAL(CHAR,z)`	0C	`CHAR`
`VAL(CHAR,r)`	Error	
`VAL(BOOLEAN,0)`	FALSE	`BOOLEAN`
`VAL(Days,5)`	Fri	`Days`
`VAL(WeekDays,5)`	Fri	`Days`
`VAL(WeekDays,z)`	Exception	

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

m-val-designator : *Arguments* → *Environment* $\xrightarrow{o}$ *Value*

m-val-designator (*args*)ρ $\triangleq$
  let [*type*, *value*] = *args* in
  converted-result(*type*, *value*) ρ 306

6.10.5 Auxiliary Definitions

converted-result : *Typed* × *Value* → *Environment* $\xrightarrow{o}$ *Value*

converted-result (*type*, *value*)ρ $\triangleq$
  let *rtype* = *host-type-of* (*type*)ρ in 325
  let *handler* = *current-exception-handler* (ρ) in 318
  if *is-Real-number-type*(*rtype*)
  then *get-real-result*(*rtype*, *scalar-value* (*value*)) *handler* 197 210
  elseif *is-whole-number-type* (*rtype*)ρ 323
  then let *tvalue* = *trunc* (*scalar-value* (*value*)) in 306 210
    get-whole-result(*rtype*, *tvalue*) *handler* 199
  else let *tvalue* = *trunc* (*scalar-value* (*value*)) in — ordinal type other than whole number type 306 210
    let *values* = *ordered-values-of* (*rtype*)ρ in 330
    if *tvalue* + 1 ∈ inds *values*
    then let *value* = *values* (*tvalue* + 1)ρ in
      if *value* ∈ *values-of* (*type*)ρ 329
      then return *value*
      else let *handler* = *current-exception-handler* (ρ) in 318
        mandatory-detection(CONVERSIONERROR) handler 396
    else let *handler* = *current-exception-handler* (ρ) in 318
      mandatory-detection(CONVERSIONERROR) handler 396

annotations Carry out type conversion; convert the argument to a corresponding value of another type.

trunc : $\mathbb{R} \to \mathbb{Z}$

trunc (r) $\triangleq$
  if $r \geq 0$
  then let $n : \mathbb{N}$ be st $n \leq r \wedge r < n + 1$ in
    n
  else $-$ *trunc* ($-$ r) 306

annotations Return the integral part of a given numerical value.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.11 The Environment and Auxiliary Formal Definitions

6.11.1 Environments

An environment associates an identifier with an entity, and with the properties of that entity. This association involves at most one of the following:

— For an identifier that denotes a constant entity, the property is its type.

— For an identifier that denotes a type, the entity is either a basic type or a unique type name. For an entity that is of a constructed type the environment associates the type name with the structure of that type including, for a structured (composite) type, the component type names.

— For an identifier that denotes a variable, when checking the well-formed properties of a program, the property of the entity is its type; when giving a meaning to the program, the entity is the associated variable.

— For an identifier that denotes a declared procedure, when checking the well-formed properties of a program, the properties of the entity are its structure and its declaration level; when giving a meaning to the program, the entity incorporates the elaborated formal parameters, and the identity of the procedure denotation (as constructed from the elaboration of the procedure declaration).

— For an identifier that denotes a module, the entity is the environment defined by the declarations contained in the module, as restricted by any exports.

An environment contains the declaration level of the block to which it applies; the static continuations for RETURN, EXCEPTION, LOOPEXIT, and RETRY. and information on the protection domain, if any, that applies.

Domains

types

Environment :: *consts* : *Constants*
types : *Types*
strucs : *Structures*
vars : *Variables*
procs : *Procedures*
mods : *Modules*
lenv : *Def-modules*
level : N
conts : *Continuation*
pdom : [*Protection-domain*]
prot : *Identifier*-set

inv *mk-Environment* (*consts*, *types*, -, *vars*, *procs*, *mods*, *defmods*, -, -, -, -) $\triangleq$
is-disj ({dom *consts*, dom *types*, dom *vars*, dom *procs*, dom *mods*}) $\wedge$ 614
dom *defmods* $\subseteq$ dom *mods*;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Constants = *Identifier* $\xrightarrow{m}$ *Constant-value*;

Types = *Identifier* $\xrightarrow{m}$ [*Denoted-type*];

Structures = *Type-name* $\xrightarrow{m}$ [*Structure*];

Variables = *Identifier* $\xrightarrow{m}$ [*Variable-typed* | *Variable*];

Procedures = *Identifier* $\xrightarrow{m}$ *Procedures-typed*;

Procedures-typed = *Procedure-value* | *Predefined-procedure-typed*;

Modules = *Identifier* $\xrightarrow{m}$ [*Module-environment*];

Def-mods = *Identifier* $\xrightarrow{m}$ [*Module-environment*];

Continuation = *Exit* $\xrightarrow{m}$ *Program-cont*

6.11.1.1 Constant Values

types

Constant-value :: *type* : *Expression-typed*
value : *Value*

6.11.1.2 Types

types

Denoted-type = *Denoted-basic-type*
| *Type-name*
| *System-basic-type*
| *Implementation-defined-type*;

Denoted-basic-type = COMPLEX-TYPE
| LONG-COMPLEX-TYPE
| REAL-TYPE
| LONG-REAL-TYPE
| UNSIGNED-TYPE
| SIGNED-TYPE
| BOOLEAN-TYPE
| CHARACTER-TYPE;

Typed = *Basic-type*
| *Type-name*
| *System-basic-type*
| *Implementation-defined-type*;

Basic-type = *Number-type*
| BOOLEAN-TYPE
| CHARACTER-TYPE;

Number-type = *Whole-number-type*
| *Real-number-type*
| *Complex-number-type*;

Whole-number-type = UNSIGNED-TYPE
| SIGNED-TYPE
| $\mathbb{Z}$-TYPE;

Real-number-type = REAL-TYPE
| LONG-REAL-TYPE
| $\mathbb{R}$-TYPE;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Complex-number-type = COMPLEX-TYPE
| LONG-COMPLEX-TYPE
| ℂ-TYPE;

Type-name = token_{type} — An infinite set of tokens denoting new types

annotations The declaration of a new type will create a new type name that is used to denote that new type. This type name is unique within a program.

types

System-basic-type = LOC-TYPE
| COROUTINE-TYPE
| EXCEPTIONSOURCE-TYPE;

Implementation-defined-type = INTERRUPTSOURCE-TYPE
| PROTECTION-TYPE

6.11.1.3 Structures

types

Structure = *Enumeration-structure*
| *Subrange-structure*
| *Set-structure*
| *Packed-set-structure*
| *Pointer-structure*
| OPAQUE
| *Procedure-structure*
| *Array-structure*
| *Record-structure*;

Enumeration-structure :: *values* : *Identifier*$^+$

annotations The structure of an enumeration type is the sequence of identifiers defined by the type. The order of the identifiers in the sequence is identical to their order in the type definition.

types

Subrange-structure :: *rtype* : *Typed*
range : *Set-value*

annotations The structure of a subrange type is the range type and a set of values defined by the lower and upper bounds of the type definition.

types

Set-structure :: *btype* : *Typed*

annotations The structure of a set type is the type name or basic type of the base type.

types

Packed-set-structure :: *btype* : *Typed*

annotations The structure of a packedset type is the type name or basic type of the base type.

types

Pointer-structure :: *type* : [*Typed*]

annotations The structure of a pointer type is the the type name or basic type of the bound type. The bound type is optional because forward use of a type name is allowed in the context of a pointer type.

types

Procedure-structure = *Proper-procedure-structure* | *Function-procedure-structure*;

Proper-procedure-structure :: *parms* : *Formal-parameters-typed*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations The structure of a proper procedure or proper procedure type is the type names (or basic types) of the formal parameters of the procedure.

types

Function-procedure-structure :: *parms* : *Formal-parameters-typed*
return : [*Typed*]

annotations The structure of a function procedure or function procedure type is the type names (or basic types) of the formal parameters of the procedure, together with the the type names or basic types of the result. The return type is optional because forward use of a type name is allowed in the context of a procedure type.

types

Formal-parameters-typed = *Formal-parameter-typed**;

Formal-parameter-typed = *Value-formal-typed* | *Variable-formal-typed*;

Value-formal-typed :: *type* : *Variable-typed*;

Variable-formal-typed :: *type* : *Variable-typed*;

Variable-typed = [*Typed*] | *Open-array-typed*

annotations A variable can either be introduced in a variable declaration or it can be a parameter of a procedure. Thus it will either be associated with its type name or its basic type, or it can be an open array parameter. The type is optional here because forward use of a type name is allowed in the context of a procedure type.

types

Open-array-typed :: *type* : *Component-typed*;

Component-typed = *Variable-typed*;

Array-structure :: *itype* : *Typed*
ctype : *Typed*

annotations The structure of an array type is the type names (or basic types) of the index and component types.

types

Record-structure :: *fields* : *Fields-list-structure*

annotations The structure of a record type is the type names (or basic types) of the components of the record.

types

Fields-list-structure = *Fields-structure**;

Fields-structure = *Fixed-fields-structure* | *Variant-fields-structure*;

Fixed-fields-structure :: *ids* : *Identifier**
type : *Typed*;

Variant-fields-structure :: *tag* : [*Identifier*]
tagt : *Typed*
variants : *Variant-structure**
other : *Fields-list-structure*;

Variant-structure :: *labels* : *Set-value*
fields : *Fields-list-structure*;

String-type :: *length* : N

annotations The *String-type* is an basic type with a structure that allows the length of a literal string to be represented. Since it is an anonymous type, it can only be the type of an expression.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.11.1.4 Expression Types

types

Expression-typed = *Typed* | *String-type* | NIL-TYPE | *Procedure-typed* | *Open-array-typed*

annotations The result of an expression can be:

— a value of a basic type, or a value whose type has been defined in a declaration;
— a string literal;
— the nil value;
— a procedure; or
— an open array.

types

Expression-structure = *Structure* | *String-type* | *Open-array-typed*

annotations The structure of the result of an expression can be:

— the structure of a type that has been defined in a declaration;
— the structure of a string literal;
— the structure of a procedure (which is the same as the structure of a procedure type under *Structure*; or
— the structure of an open array.

6.11.1.5 Types for Declared Procedures

types

Procedure-value :: *ptyped* : *Procedure-typed*
value : [*Procedure-id*]

annotations *ptyped* is the type structure and declaration level information from the elaboration of the procedure declaration. *value* is the value that is assigned to procedure variables and the value that is used in procedure calls to retrieve the procedure denotation from the state; it is optional because procedure headings can appear separately from procedure declarations in definitions and in forward declarations.

types

Procedure-typed = *Proper-procedure-typed* | *Function-procedure-typed*;

Proper-procedure-typed :: *type* : *Proper-procedure-structure*
level : ℕ

annotations *type* corresponds to the type structure for procedure types and is used for procedure calls and for checking compatibility. *level* is the declaration level at which the procedure is declared and is used in checks of assignment comaptibility.

types

Function-procedure-typed :: *type* : *Function-procedure-structure*
level : ℕ

annotations *type* corresponds to the type structure for procedure types and is used for procedure calls and for checking compatibility. *level* is the declaration level at which the procedure is declared and is used in checks of assignment comaptibility.

types

Procedure-id = token_{procid}

6.11.1.6 Types for Predefined Procedures

types

Predefined-procedure-typed = *Predefined-proper-procedure-typed*
| *Predefined-function-procedure-typed*;

Predefined-proper-procedure-typed = *Standard-proper-procedure-typed*
| *System-proper-procedure-typed*;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Predefined-function-procedure-typed = *Standard-function-procedure-typed*
| *System-function-procedure-typed*;

Standard-proper-procedure-typed = STANDARD-PROPER-PROCEDURE;

System-proper-procedure-typed = SYSTEM-PROPER-PROCEDURE
| COROUTINE-PROPER-PROCEDURE
| EXCEPTIONS-PROPER-PROCEDURE;

Standard-function-procedure-typed = STANDARD-FUNCTION-PROCEDURE;

System-function-procedure-typed = SYSTEM-FUNCTION-PROCEDURE
| COROUTINE-FUNCTION-PROCEDURE
| EXCEPTIONS-FUNCTION-PROCEDURE
| TERMINATION-FUNCTION-PROCEDURE
| M2EXCEPTION-FUNCTION-PROCEDURE

6.11.1.7 Module Environments

types

Module-environment = *Unqualified-environment* | *Qualified-environment* | *System-environment*;

Unqualified-environment :: *env* : *Environment*;

Qualified-environment :: *env* : *Environment*;

System-environment :: *env* : *Environment*

6.11.1.8 Variables

types

Variable = *Elementary-variable*
| *Pointer-variable*
| *Array-variable*
| *Record-variable*
| *Variant-variable*
| *Tag-variable*
| *Tagged-variable*;

Elementary-variable :: *loc* : *Loc*
range : [*Set-value*]

annotations If the identifier is associated with an elementary variable that is of a subrange type, then the range component contains the possible set of values that can be assigned to the variable.

types

Pointer-variable :: *loc* : *Loc*
type : *Typed*

annotations If the identifier is associated with a pointer variable then the type of the pointer is associated with the variable so that any pointer dereferencing can be checked.

types

Array-variable :: *vars* : *Array-components*;

Array-components = *Ordinal-value* $\xrightarrow{m}$ *Variable*

annotations The components of an array variable are all of the same type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

types

Record-variable :: *fields* : *Record-components*;

Record-components = *Identifier* $\xrightarrow{m}$ *Variable*;

Variant-variable :: *var* : *Variable*
tagloc : *Elementary-variable*
tagvals : *Value*-set

annotations A variant variable contains information about the explicit tag associated with the variant fields of which the variable is a component; thus before accessing the value of the variable, the tag can be checked to see if its value is such that the variant that contains the variable is active.

types

Tag-variable :: *var* : *Elementary-variable*
variants : *Variant-component*-set

annotations A tag variable contains information about the variant components that are associated with it. Thus any change to the value of the tag that will cause a different component to be selected can be reflected by setting the variables of the old component to undefined.

types

Tagged-variable :: *var* : *Variable*
tagvar : *Tag-variable*
tagvals : *Value*-set

annotations A tagged variable contains information about the implicit tag associated with the variant fields of which the variable is a component; thus before accessing the value of the variable, the tag can be checked to see if its value is such that the variant that contains the variable is active.

types

Variant-component :: *labels* : *Value*-set
fields : *Record-components*

6.11.1.9 Protection Domains

types

Protection-domain :: *val* : *Protection*

6.11.1.10 Local Continuations

types

Exit = RETURN | EXCEPTION | RETRY | LOOPEXIT

6.11.2 Operations on the Environment

6.11.2.1 Functions Accessing the Constants Component of the Environment

is-constant : *Qualident* → *Environment* → $\mathbb{B}$

is-constant (*qid*)ρ $\triangleq$
let *mk-* (*id*, ρ_{mod}) = *unqualify* (*qid*)ρ in
id ∈ dom ρ_{mod}.*consts*

53

annotations Check that a qualident denotes a constant.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

type-of-constant : *Qualident* → *Environment* → *Typed*

type-of-constant (*qid*)ρ $\triangleq$
  let *mk-*(*id*, ρ_{mod}) = *unqualify* (*qid*)ρ in 53
  let *mk-Constant-value* (*type*, -) = ρ_{mod}.*consts* (*id*) in
  type

annotations The result is the unique type of a qualident that denotes a constant.

associated-constant-value : *Qualident* → *Environment* → *Value*

associated-constant-value (*qid*)ρ $\triangleq$
  let *mk-*(*id*, ρ_{mod}) = *unqualify* (*qid*)ρ in 53
  let *mk-Constant-value* (-, *value*) = ρ_{mod}.*consts* (*id*) in
  value

annotations The result is the constant value associated with a qualident that is introduced in a constant declaration.

overwrite-const-environment : *Constants* → *Environment* → *Environment*

overwrite-const-environment (*consts*)ρ $\triangleq$
  let ρ_{left} = *remove-from-environment* (dom *consts*)ρ in 320
  let *nconsts* = ρ_{left}.*consts* † *consts* in
  μ (ρ_{left}, *consts* ↦ *nconsts*)

annotations Construct an environment by overwriting the given environment with the given constants component. Any identifiers from the given environment that do not appear in the given constants component will remain visible in the new environment.

6.11.2.2 Functions Accessing the Types Component of the Environment

is-type : *Qualident* → *Environment* → $\mathbb{B}$

is-type (*qid*)ρ $\triangleq$
  let *mk-*(*id*, ρ_{mod}) = *unqualify* (*qid*)ρ in 53
  id ∈ dom ρ_{mod}.*types*

annotations Check that a qualident denotes a type.

type-of : *Qualident* → *Environment* → [*Typed*]

type-of (*qid*)ρ $\triangleq$
  let *mk-*(*id*, ρ_{mod}) = *unqualify* (*qid*)ρ in 53
  let *typem* = ρ_{mod}.*types* in
  if *id* ∈ dom *typem*
  then *typem* (*id*)
  else nil

annotations The result is the type name or basic type associated with a qualident that denotes a type.

overwrite-type-environment : *Types* → *Environment* → *Environment*

overwrite-type-environment (*types*)ρ $\triangleq$
  let ρ_{left} = *remove-from-environment* (dom *types*)ρ in 320
  let *ntypes* = ρ_{left}.*types* † *types* in
  μ (ρ_{left}, *types* ↦ *ntypes*)

annotations Construct an environment by overwriting the given environment with the given types component. Any identifiers from the given environment that do not appear in the given types component will remain visible in the new environment.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.11.2.3 Functions Accessing the Structures Component of the Environment

structure-of : *Expression-typed* → *Environment* → [*Expression-structure*]

structure-of (*etype*)ρ $\triangleq$
  (*is-Type-name*(*etype*) → ρ.*strucs* (*etype*),
   is-Procedure-typed(*etype*) → *procedure-structure* (*etype*), 316
   others → *etype*)

annotations The result is the structure associated with a type or with a procedure. Only constructed types, procedures, and the *String-type* have a structure; other types are synonyms for the basic types or the basic types themselves.

associated-structure : *Qualident* → *Environment* → [*Structure*]

associated-structure (*qid*)ρ $\triangleq$
  let *mk-* (*id*, ρ_{mod}) = *unqualify* (*qid*)ρ in 53
  let *type* = ρ_{mod}.*types* (*id*) in
  structure-of (*type*)ρ_{mod} 315

annotations The result is the structure of a qualident that denotes a type.

overwrite-struc-environment : *Structures* → *Environment* → *Environment*

overwrite-struc-environment (*strucs*)ρ $\triangleq$
  let ρ_{left} = *remove-from-environment* (dom *strucs*)ρ in 320
  let *nstrucs* = ρ_{left}.*strucs* † *strucs* in
  μ (ρ_{left}, *strucs* ↦ *nstrucs*)

annotations Construct an environment by overwriting the given environment with the given structure component. Any type names from the given environment that do not appear in the given structure component will remain in the domain of the new environment.

6.11.2.4 Functions Accessing the Variables Component of the Environment

is-simple-variable-identifier : *Identifier* → *Environment* → $\mathbb{B}$

is-simple-variable-identifier (*id*)ρ $\triangleq$
  id ∈ dom ρ.*vars*

is-variable-or-field : *Qualident* → *Environment* → $\mathbb{B}$

is-variable-or-field (*qid*)ρ $\triangleq$
  let *mk-* (*id*, ρ_{mod}) = *unqualify* (*qid*)ρ in 53
  id ∈ dom ρ_{mod}.*vars*

annotations Check if a qualident denotes a variable; this may be a field of a record inside a with statement.

type-of-variable-or-field : *Qualident* → *Environment* → *Variable-typed*

type-of-variable-or-field (*qid*)ρ $\triangleq$
  let *mk-* (*id*, ρ_{mod}) = *unqualify* (*qid*)ρ in 53
  ρ_{mod}.*vars* (*id*)

annotations The result is the type name or basic type of a qualident that denotes a variable; used with environments constructed for the *wf-* and *t-* functions.

associated-variable : *Qualident* → *Environment* → *Variable*

associated-variable (*qid*)ρ $\triangleq$
  let *mk-* (*id*, ρ_{mod}) = *unqualify* (*qid*)ρ in 53
  ρ_{mod}.*vars* (*id*)

annotations The result is the variable associated with a qualident that denotes a variable; used with environments constructed for the *m-* functions.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

overwrite-var-environment : *Variables* → *Environment* → *Environment*

overwrite-var-environment (*vars*)ρ ≜
  let ρ_{left} = *remove-from-environment* (dom *vars*)ρ in 320
  let *nvars* = ρ_{left}.*vars* † *vars* in
  μ (ρ_{left}, *vars* ↦ *nvars*)

annotations Construct an environment by overwriting the given environment with the given variables component. Any identifiers from the given environment that do not appear in the given variables component will remain visible in the new environment.

6.11.2.5 Functions Accessing the Procedures Component of the Environment

is-procedure : *Qualident* → *Environment* → $\mathbb{B}$

is-procedure (*qid*)ρ ≜
  let *mk-* (*id*, ρ_{mod}) = *unqualify* (*qid*)ρ in 53
  id ∈ dom ρ_{mod}.*procs*

annotations Check if a qualident denotes a procedure.

type-of-procedure : *Qualident* → *Environment* → *Procedure-typed*

type-of-procedure (*qid*)ρ ≜
  let *mk-* (*id*, ρ_{mod}) = *unqualify* (*qid*)ρ in 53
  let *mk-Procedure-value* (*ptyped*, -) = ρ_{mod}.*procs* (*id*) in
  ptyped

annotations The result is the type of a qualident that denotes a declared procedure; this is a record comprising the structure and declaration level information.

procedure-structure : *Procedure-typed* → *Procedure-structure*

procedure-structure (*ptyped*) ≜
  cases *ptyped* :
    mk-Proper-procedure-typed (*type*, -) → *type*,
    mk-Function-procedure-typed (*type*, -) → *type*
  end

annotations The result is the type structure for a declared procedure.

procedure-level : *Procedure-typed* → [$\mathbb{N}$]

procedure-level (*ptyped*) ≜
  cases *ptyped* :
    mk-Proper-procedure-typed (-, *level*) → *level*,
    mk-Function-procedure-typed (-, *level*) → *level*
  end

annotations The result is the declaration level of a declared procedure.

overwrite-proc-environment : *Procedures* → *Environment* → *Environment*

overwrite-proc-environment (*procs*)ρ ≜
  let ρ_{left} = *remove-from-environment* (dom *procs*)ρ in 320
  let *nprocs* = ρ_{left}.*procs* † *procs* in
  μ (ρ_{left}, *procs* ↦ *nprocs*)

annotations Construct an environment by overwriting the given environment with the given procs component. Any identifiers from the given environment that do not appear in the given procs component will remain visible in the new environment.

procedures-of : *Environment* → *Procedures*

procedures-of (ρ) ≜
  ρ.*procs*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.11.2.6 Functions Accessing the Modules Component of the Environment

is-module : *Qualident* → *Environment* → $\mathbb{B}$

is-module (*qid*)ρ $\triangleq$
  let *mk-*(*id*, ρ_{mod}) = *unqualify* (*qid*)ρ in
  id ∈ dom ρ_{mod}.*mods*

53

annotations Checks if a qualident denotes a module.

associated-module : *Qualident* → *Environment* → *Module-environment*

associated-module (*qid*)ρ $\triangleq$
  let *mk-*(*id*, ρ_{mod}) = *unqualify* (*qid*)ρ in
  ρ_{mod}.*mods* (*id*)

53

annotations Return the module environment associated with the qualident.

get-module-environment : *Module-environment* → *Environment*

get-module-environment (*menv*) $\triangleq$
  cases *menv* :
    mk-Qualified-environment (*ρ*) → *ρ*,
    mk-Unqualified-environment (*ρ*) → *ρ*,
    mk-System-environment (*ρ*) → *ρ*
  end

annotations Return the environment associated with the module environment.

overwrite-mod-environment : *Modules* → *Environment* → *Environment*

overwrite-mod-environment (*mods*)ρ $\triangleq$
  let ρ_{left} = *remove-from-environment* (dom *mods*)ρ in
  let *nmods* = ρ_{left}.*mods* † *mods* in
  μ (ρ_{left}, *mods* ↦ *nmods*)

320

annotations Construct an environment by overwriting the given environment with the given module component. Any identifiers from the given environment that do not appear in the given module component will remain visible in the new environment.

construct-module-environment : *Identifier* → *Environment* → *Environment*

construct-module-environment (*name*)ρ $\triangleq$
  let *menv* = {*name* ↦ *mk-Qualified-environment* (*ρ*)} in
  mk-Environment ({↦}, {↦}, {↦}, {↦}, {↦}, *menv*, {↦}, 0, {↦}, nil , { })

associated-local-environment : *Qualident* → *Environment* → *Module-environment*

associated-local-environment (*qid*)ρ $\triangleq$
  let *mk-*(*id*, ρ_{mod}) = *unqualify* (*qid*)ρ in
  ρ_{mod}.*lenv* (*id*)

53

annotations Return the local definition module environment associated with the qualident.

construct-local-environment : *Identifier* → *Environment* → *Environment*

construct-local-environment (*name*)ρ $\triangleq$
  let *lenv* = {*name* ↦ *mk-Qualified-environment* (*ρ*)} in
  mk-Environment ({↦}, {↦}, {↦}, {↦}, {↦}, {↦}, *lenv*, 0, {↦}, nil , { })

6.11.2.7 Functions Accessing the Procedure Level Component of the Environment

current-level : *Environment* → N

current-level (*ρ*) $\triangleq$
  ρ.*level*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations Return the current declaration level of the environment.

new-level : *Environment* → *Environment*

new-level (*ρ*) ≜
  μ (*ρ*, *level* ↦ *ρ.level* + 1)

annotations Increase the declaration level of declarations by one.

6.11.2.8 Functions Accessing the Continuations Component of the Environment

current-exception-handler : *Environment* → *Program-cont*

current-exception-handler (*ρ*) ≜
  ρ.conts(EXCEPTION)

annotations Extract the current exception handler.

6.11.2.9 Functions Accessing the Protection Components of the Environment

add-protection : [*Protection-domain*] → *Environment* → *Environment*

add-protection (*pdomv*)*ρ* ≜
  if *ρ.pdom* ≠ nil ∧
    pdomv = nil
  then *ρ*
  else *μ* (*ρ*, *pdom* ↦ *pdomv*)

annotations If a module is declared within a protected module, and this module is not protected, the same protection applies; otherwise the protection of this module applies.

module-protection-domain : *Environment* → [*Protection-domain*]

module-protection-domain (*ρ*) ≜
  ρ.pdom

annotations Extract the module protection expression and procedures from the environment.

is-protected- domain : *Environment* → 𝔹

is-protected-domain (*ρ*) ≜
  ρ.pdom ≠ nil

mark-exported-from-protected : *Identifier*-set → *Environment* → *Environment*

mark-exported-from-protected (*ids*)*ρ* ≜
  μ (*ρ*, *prot* ↦ *ids*)

annotations Add the set of exported identifiers to the environment.

exported-from-protected : *Environment* → *Identifier*-set

exported-from-protected (*ρ*) ≜
  ρ.prot

annotations Extract the set of exported identifiers from the environment.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.11.2.10 Functions Accessing the Environment

types

$$\begin{aligned} \mathit{Entity} = {} & \mathit{Constant\text{-}value} \\ & \mid \mathit{Denoted\text{-}type} \\ & \mid \mathit{Variable\text{-}typed} \\ & \mid \mathit{Variable} \\ & \mid \mathit{Procedures\text{-}typed} \\ & \mid \mathit{Module\text{-}environment} \end{aligned}$$

$$\mathit{access\text{-}environment} : \mathit{Qualident} \to \mathit{Environment} \to \mathit{Entity}$$

$$\begin{aligned} & \mathit{access\text{-}environment}\,(\mathit{qid})\rho \triangleq \\ & \quad \textsf{let } \mathit{mk\text{-}}(\mathit{id}, \rho_{\mathit{mod}}) = \mathit{unqualify}\,(\mathit{qid})\rho \textsf{ in} \qquad 53 \\ & \quad \textsf{let } \mathit{mk\text{-}Environment}\,(\mathit{consts}, \mathit{types}, \text{-}, \mathit{vars}, \mathit{procs}, \mathit{mods}, \text{-}, \text{-}, \text{-}, \text{-}, \text{-}) = \rho_{\mathit{mod}} \textsf{ in} \\ & \quad (\mathit{id} \in \textsf{dom } \mathit{consts} \to \mathit{consts}\,(\mathit{id}), \\ & \quad\ \mathit{id} \in \textsf{dom } \mathit{types} \to \mathit{types}\,(\mathit{id}), \\ & \quad\ \mathit{id} \in \textsf{dom } \mathit{vars} \to \mathit{vars}\,(\mathit{id}), \\ & \quad\ \mathit{id} \in \textsf{dom } \mathit{procs} \to \mathit{procs}\,(\mathit{id}), \\ & \quad\ \mathit{id} \in \textsf{dom } \mathit{mods} \to \mathit{mods}\,(\mathit{id})) \end{aligned}$$

annotations Look up a (possibly qualified) identifier in the environment, and return the associated entity and its properties.

$$\mathit{visible\text{-}identifiers} : \mathit{Environment} \to \mathit{Identifier}\text{-set}$$

$$\begin{aligned} & \mathit{visible\text{-}identifiers}\,(\rho) \triangleq \\ & \quad \textsf{let } \mathit{mk\text{-}Environment}\,(\mathit{consts}, \mathit{types}, \text{-}, \mathit{vars}, \mathit{procs}, \mathit{mods}, \text{-}, \text{-}, \text{-}, \text{-}, \text{-}) = \rho \textsf{ in} \\ & \quad \textsf{dom } \mathit{consts} \cup \textsf{dom } \mathit{types} \cup \textsf{dom } \mathit{vars} \cup \textsf{dom } \mathit{procs} \cup \textsf{dom } \mathit{mods} \end{aligned}$$

annotations The set of visible identifiers is the set defined by the union of all of the domains of various (mapping) components of the environment. The scope of these identifiers includes the points where the environment applies.

6.11.2.11 Operations to Update Environments

$$\mathit{merge\text{-}environment} : \mathit{Environment} \times \mathit{Environment} \to \mathit{Environment}$$

$$\begin{aligned} & \mathit{merge\text{-}environment}\,(\rho_a, \rho_b) \triangleq \\ & \quad \textsf{let } \mathit{mk\text{-}Environment}\,(\mathit{nconsts}, \mathit{ntypes}, \mathit{nstrucs}, \mathit{nvars}, \mathit{nprocs}, \mathit{nmods}, \mathit{nlenv}, \mathit{nlevel}, \mathit{nconts}, \mathit{npdom}, \mathit{nprot}) = \rho_a \textsf{ in} \\ & \quad \textsf{let } \mathit{mk\text{-}Environment}\,(\mathit{consts}, \mathit{types}, \mathit{strucs}, \mathit{vars}, \mathit{procs}, \mathit{mods}, \mathit{lenv}, \text{-}, \text{-}, \text{-}) = \rho_b \textsf{ in} \\ & \quad \mathit{mk\text{-}Environment}\,(\mathit{nconsts} \cup \mathit{consts}, \mathit{ntypes} \cup \mathit{types}, \mathit{nstrucs} \cup \mathit{strucs}, \mathit{nvars} \cup \mathit{vars}, \\ & \qquad\qquad \mathit{nprocs} \cup \mathit{procs}, \mathit{nmods} \cup \mathit{mods}, \mathit{nlenv} \cup \mathit{lenv}, \mathit{nlevel}, \mathit{nconts}, \mathit{npdom}, \mathit{nprot}) \\ & \textsf{pre } \mathit{visible\text{-}identifiers}\,(\rho_a) \cap \mathit{visible\text{-}identifiers}\,(\rho_b) = \{\,\} \qquad 319\ 319 \end{aligned}$$

annotations Merge two environments.

$$\mathit{merge\text{-}environments} : \mathit{Environment}\text{-set} \to \mathit{Environment}$$

$$\begin{aligned} & \mathit{merge\text{-}environments}\,(\mathit{envs}) \triangleq \\ & \quad \textsf{if } \mathit{envs} = \{\,\} \\ & \quad \textsf{then } \{\,\} \\ & \quad \textsf{else let } \rho \in \mathit{envs} \textsf{ in} \\ & \qquad \textsf{let } \rho_{\mathit{rest}} = \mathit{merge\text{-}environments}\,(\mathit{envs} - \rho) \textsf{ in} \qquad 319 \\ & \qquad \mathit{merge\text{-}environment}\,(\rho_{\mathit{rest}}, \rho) \qquad 319 \\ & \textsf{pre } \mathit{is\text{-}disj}\,(\{\mathit{visible\text{-}identifiers}\,(\rho) \mid \qquad 614\ 319 \\ & \qquad\qquad \rho \in \mathit{envs}\}) \end{aligned}$$

annotations Merge all the environments contained in a set of environments.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

overwrite-environment : *Environment* → *Environment* → *Environment*

overwrite-environment (*env*)ρ $\triangleq$
  let *mk-Environment* (*econsts*, *etypes*, *estrucs*, *evars*, *eprocs*, *emods*, *elenv*, *elevel*, *econts*, *epdom*, *eprot*) = *env* in
  let *mk-Environment* (*consts*, *types*, *strucs*, *vars*, *procs*, *mods*, *lenv*, -, -, -, -) = ρ in
  mk-Environment (*consts* † *econsts*, *types* † *etypes*, *strucs* † *estrucs*, *vars* † *evars*, *procs* † *eprocs*,
    mods † *emods*, *lenv* † *elenv*, *elevel*, *econts*, *epdom*, *eprot*)

annotations Overwrite the environment ρ with the environment *env*.

restrict-environment : *Identifier*-set → *Environment* → *Environment*

restrict-environment (*ids*)ρ $\triangleq$
  let *mk-Environment* (*consts*, *types*, *strucs*, *vars*, *procs*, *mods*, *lenv*, *level*, *conts*, *pdom*, *prot*) = ρ in
  mk-Environment (*ids* ◁ *consts*, *ids* ◁ *types*, *strucs*, *ids* ◁ *vars*, *ids* ◁ *procs*, *ids* ◁ *mods*, *ids* ◁ *lenv*, *level*, *conts*, *pdom*, *prot*)

annotations *visible-identifiers*(*restrict-environment*(*ids*)ρ) = *ids*

remove-from-environment : *Identifier*-set → *Environment* → *Environment*

remove-from-environment (*ids*)ρ $\triangleq$
  let *mk-Environment* (*consts*, *types*, *strucs*, *vars*, *procs*, *mods*, *lenv*, *level*, *conts*, *pdom*, *prot*) = ρ in
  mk-Environment (*ids* ⩤ *consts*, *ids* ⩤ *types*, *strucs*, *ids* ⩤ *vars*, *ids* ⩤ *procs*, *ids* ⩤ *mods*, *ids* ⩤ *lenv*, *level*, *conts*, *pdom*, *prot*)

to-be-declared : *Environment* → *Environment*

to-be-declared (ρ) $\triangleq$
  let *constants* = {*id* ↦ nil | *id* ∈ dom ρ.*consts*},
    types = {*id* ↦ nil | *id* ∈ dom ρ.*types*},
    structures = {↦},
    variables = {*id* ↦ nil | *id* ∈ dom ρ.*vars*},
    procedures = {*id* ↦ nil | *id* ∈ dom ρ.*procs*},
    modules = {*id* ↦ nil | *id* ∈ dom ρ.*mods*},
    defmods = {*id* ↦ nil | *id* ∈ dom ρ.*lenv*},
    level = 0,
    local-continuations = {↦},
    protection-domain = nil ,
    protected-procedures = { } in
  mk-Environment (*constants*, *types*, *structures*, *variables*, *procedures*, *modules*, *defmods*,
    level, *local-continuations*, *protection-domain*, *protected-procedures*)

annotations The identifiers are visible in the new environment, but their meanings are unknown; used in checking declare before use rules.

combine-definition-and-imported-environments : *Environment* × *Environment* → *Environment*

combine-definition-and-imported-environments (ρ_d, ρ_i) $\triangleq$
  let *opaques* = *opaque-types-of* (ρ_d), 328
    procs = *procedures-of* (ρ_d) in 316
  let *idef* = *remove-from-environment* (dom *procs* ∪ *opaques*)ρ_d in 320
  merge-environment (*idef*, ρ_i) 319

annotations Combine the environments of a definition module and the associated implementation module. Note that the declarations for opaque types and procedures are taken from the implementation module.

6.11.3 Types

6.11.3.1 New Type Names

generate-type-name : *Environment* → *Type-name*

generate-type-name (ρ) $\triangleq$
  let *type* : *Type-name* be st *type* ∉ dom ρ.*strucs* in
  type

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations Generate a type name that has not been used elsewhere in the current environment.

6.11.3.2 Structured (Composite) Types

A structured (composite) type is an array type or a record type.

is-structured-type : *Expression-typed* → *Environment* → 𝔹

is-structured-type (*type*)ρ ≙
  is-array-type (*type*)ρ ∨ 321
  is-record-type (*type*)ρ 321

annotations Check that a type is a structured type.

is-array-type : *Expression-typed* → *Environment* → 𝔹

is-array-type (*type*)ρ ≙
  is-Array-structure(*structure-of* (*type*)ρ) ∨ 315
  is-open-array (*type*)ρ 328

annotations Check that a type denotes an array type.

is-record-type : *Expression-typed* → *Environment* → 𝔹

is-record-type (*type*)ρ ≙
  is-Record-structure(*structure-of* (*type*)ρ) 315

annotations Check that a type denotes a record type.

6.11.3.3 Elementary Types

An elementary type is either a procedure type, a pointer type, an opaque type, a set type, a packedset type, a scalar type, or a complex number type.

is-elementary-type : *Expression-typed* → *Environment* → 𝔹

is-elementary-type (*type*)ρ ≙
  is-procedure-type (*type*)ρ ∨ 321
  is-set-type (*type*)ρ ∨ 322
  is-pointer-type (*type*)ρ ∨ 322
  is-opaque-type (*type*)ρ ∨ 322
  is-scalar-type (*type*)ρ ∨ 322
  is-Complex-number-type(*type*)

annotations Check that a type is an elementary type.

is-procedure-type : *Expression-typed* → *Environment* → 𝔹

is-procedure-type (*type*)ρ ≙
  is-proper-procedure-type (*type*)ρ ∨ 321
  is-function-procedure-type (*type*)ρ 322

annotations Check that a type-name denotes a proper procedure type or a function procedure type; these are the types of procedure variables.

is-proper-procedure-type : *Expression-typed* → *Environment* → 𝔹

is-proper-procedure-type (*type*)ρ ≙
  is-Proper-procedure-structure(*structure-of* (*type*)ρ) 315

annotations Check that a type denotes a proper procedure type; this is the type of a procedure variable of a proper procedure structure.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

is-function-procedure-type : *Expression-typed* → *Environment* → 𝔹

is-function-procedure-type (*type*)ρ ≙
  is-Function-procedure-structure(*structure-of* (*type*)ρ) 315

annotations Check that a type denotes a function procedure type; this is the type of a procedure variable of a function procedure structure.

is-set-type : *Expression-typed* → *Environment* → 𝔹

is-set-type (*type*)ρ ≙
  is-Set-structure(*structure-of* (*type*)ρ) ∨ 315
  is-Packed-set-structure(*structure-of* (*type*)ρ) 315

annotations Check that a type denotes a set type.

is-packed-set-type : *Expression-typed* → *Environment* → 𝔹

is-packed-set-type (*type*)ρ ≙
  is-Packed-set-structure(*structure-of* (*type*)ρ) 315

annotations Check that a type denotes a packedset type.

is-pointer-type : *Expression-typed* → *Environment* → 𝔹

is-pointer-type (*type*)ρ ≙
  is-Pointer-structure(*structure-of* (*type*)ρ) 315

annotations Check that a type denotes a pointer type.

is-opaque-type : *Expression-typed* → *Environment* → 𝔹

is-opaque-type (*type*)ρ ≙
  structure-of (*type*)ρ = OPAQUE 315

annotations Check that a type denotes an opaque type.

6.11.3.4 Scalar Types

A scalar type is either an ordinal type or a real number type.

is-scalar-type : *Expression-typed* → *Environment* → 𝔹

is-scalar-type (*type*)ρ ≙
  is-Real-number-type(*type*) ∨
  is-ordinal-type (*type*)ρ 322

annotations Check that a type denotes a scalar type.

6.11.3.5 Ordinal Types

An ordinal type is either the Boolean type, the character type, an enumeration type, a subrange type, or a whole number type; the character literal type is included since it is interpreted as a value of the character type in contexts where an ordinal type is expected.

is-ordinal-type : *Expression-typed* → *Environment* → 𝔹

is-ordinal-type (*type*)ρ ≙
  type = BOOLEAN-TYPE ∨
  is-character-type (*type*)ρ ∨ 323
  is-enumeration-type (*type*)ρ ∨ 323
  is-subrange-type (*type*)ρ ∨ 323
  is-whole-number-type (*type*)ρ ∨ 323
  is-character-constant-type (*type*)ρ 324

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations Check that a type is an ordinal type.

is-character-type : *Expression-typed* → *Environment* → $\mathbb{B}$

is-character-type (*type*)ρ $\triangleq$
  type = CHARACTER-TYPE

annotations Check that a type is the character type.

is-enumeration-type : *Expression-typed* → *Environment* → $\mathbb{B}$

is-enumeration-type (*type*)ρ $\triangleq$
  is-Enumeration-structure(*structure-of* (*type*)ρ) 315

annotations Check that a type is an enumeration type.

is-enumeration-constant : *Identifier* → *Environment* → $\mathbb{B}$

is-enumeration-constant (*id*)ρ $\triangleq$
  let *ent* = *access-environment* (*id*)ρ in 319
  is-Constant-value(*ent*) ∧
  let *mk-Constant-value* (*type*, *val*) = *ent* in
  is-enumeration-type (*type*)ρ ∧ 323
  let *mk-Enumeration-value* (*cid*, -) = *val* in
  id = *cid*

annotations Check that an identifier is an enumeration constant and thus denotes itself.

is-subrange-type : *Expression-typed* → *Environment* → $\mathbb{B}$

is-subrange-type (*type*)ρ $\triangleq$
  is-Subrange-structure(*structure-of* (*type*)ρ) 315

annotations Check that a type is a subrange type.

6.11.3.6 Whole Number Types

A whole number type is either the unsigned type, the signed type or the type of an expression whose operands are whole number literal values.

is-whole-number-type : *Expression-typed* → *Environment* → $\mathbb{B}$

is-whole-number-type (*type*)ρ $\triangleq$
  is-unsigned-type (*type*)ρ ∨ 323
  is-signed-type (*type*)ρ ∨ 323
  is-ZZ-type (*type*)ρ 323

annotations Check that a type denotes a whole number type.

is-unsigned-type : *Expression-typed* → *Environment* → $\mathbb{B}$

is-unsigned-type (*type*)ρ $\triangleq$
  type = UNSIGNED-TYPE

annotations Check that a type denotes the unsigned type.

is-signed-type : *Expression-typed* → *Environment* → $\mathbb{B}$

is-signed-type (*type*)ρ $\triangleq$
  type = SIGNED-TYPE

annotations Check that a type denotes the signed type.

is-ZZ-type : *Expression-typed* → *Environment* → $\mathbb{B}$

is-ZZ-type (*type*)ρ $\triangleq$
  type = $\mathbb{Z}$-TYPE

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations Check that a type denotes the $\mathbb{Z}$-TYPE.

6.11.3.7 Number Types

A number type is a real number type or a whole number type.

NOTE — A complex number type is not a 'number type' as it does not permit relational operations.

$$\textit{is-number-type} : \textit{Expression-typed} \rightarrow \textit{Environment} \rightarrow \mathbb{B}$$

$$\textit{is-number-type}\,(\textit{type})\rho \triangleq \\ \quad \textit{is-Real-number-type}(\textit{type}) \vee \\ \quad \textit{is-whole-number-type}\,(\textit{type})\rho$$ 323

annotations Check that a type denotes a real number type or a whole number type.

6.11.3.8 String Types

The string type is the type of a literal string; it has a structure representing the length of the literal string. Values of the string literal type of length 0 or length 1 are distinguished as values of the character literal type since the character type is assignment compatible and expression compatible with this type.

$$\textit{is-character-constant-type} : \textit{Expression-typed} \rightarrow \textit{Environment} \rightarrow \mathbb{B}$$

$$\textit{is-character-constant-type}\,(\textit{type})\rho \triangleq \\ \quad \textit{is-String-type}(\textit{type}) \wedge \\ \quad (\textit{type.length} = 0 \vee \textit{type.length} = 1)$$

annotations Check that a type denotes a string type of length 0 or 1.

6.11.3.9 System Storage Types

A system storage type is either the location type, denoted by `SYSTEM.LOC`, the word type, denoted by `SYSTEM.WORD`, or the byte type, denoted by `SYSTEM.BYTE` if provided by an implementation.

$$\textit{is-system-storage-type} : \textit{Expression-typed} \rightarrow \textit{Environment} \rightarrow \mathbb{B}$$

$$\textit{is-system-storage-type}\,(\textit{type})\rho \triangleq \\ \quad \textit{is-loc-type}\,(\textit{type})\rho \vee \\ \quad \textit{type} = \text{WORD-TYPE} \vee \\ \quad \textit{type} = \text{BYTE-TYPE}$$ 324

annotations Check that a type denotes a system storage type.

$$\textit{is-loc-type} : \textit{Expression-typed} \rightarrow \textit{Environment} \rightarrow \mathbb{B}$$

$$\textit{is-loc-type}\,(\textit{type})\rho \triangleq \\ \quad \textit{type} = \text{LOC-TYPE}$$

annotations Check that a type denotes the location type.

$$\textit{is-address-type} : \textit{Expression-typed} \rightarrow \textit{Environment} \rightarrow \mathbb{B}$$

$$\textit{is-address-type}\,(\textit{type})\rho \triangleq \\ \quad \textit{type} = \text{ADDRESS-TYPE}$$

annotations Check that a type denotes the address type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.11.4 Unit Types

A unit type is either the range type of a subrange type, the host type of a subrange type, the base type of a set type, the base type of a packedset type, the component type of an array type, the index type of an array type, or the bound type of a pointer type.

6.11.4.1 Range Type

The range type of a subrange type is the ordinal type from which the subrange type is directly derived.

range-type-of : *Typed* → *Environment* → *Typed*

range-type-of (*type*)ρ $\triangleq$
  let *mk-Subrange-structure* (*rtype*, -) = *structure-of* (*type*)ρ in 315
  rtype

annotations The result is the range type of a subrange type.

6.11.4.2 Host Type

The host type of a subrange type is the ordinal type from which the subrange type is indirectly derived by defining a subrange of a previously defined ordinal type. It is the Boolean type, the character type, an enumeration type, the signed type, or the unsigned type.

host-type-of : *Typed* → *Environment* → *Typed*

host-type-of (*type*)ρ $\triangleq$
  if *is-subrange-type* (*type*)ρ 323
  then let *rtype* = *range-type-of* (*type*)ρ in 325
    host-type-of (*rtype*)ρ 325
  else *type*

annotations The result is the host type of a subrange type.

6.11.4.3 Base Type

The base type of a set type or a packedset type is the ordinal type to which the members of the set type or packedset type directly belong.

base-type-of : *Typed* → *Environment* → *Typed*

base-type-of (*type*)ρ $\triangleq$
  let *mk-Set-structure* (*btype*) = *structure-of* (*type*)ρ in 315
  btype

annotations The result is the base type of a set type or a packedset type.

6.11.4.4 Component Type

The component type of an array type is the type of the components of an array structure.

component-type-of : *Expression-typed* → *Environment* → *Typed*

component-type-of (*type*)ρ $\triangleq$
  let *struc* = *structure-of* (*type*)ρ in 315
  cases *struc* :
    mk-Array-structure (-, *ctype*) → *ctype*,
    mk-Open-array-typed (*type*) → *type*
  end

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations The result is the component type of an array type.

6.11.4.5 Index Type

The index type of an array type is the type of the index of an array structure.

index-type-of : *Expression-typed* → *Environment* → *Typed*

index-type-of (*type*)ρ ≙
  let *struc* = *structure-of* (*type*)ρ in 315
  cases *struc* :
    mk-Array-structure (*itype*, -) → *itype*,
    mk-Open-array-typed (-) → UNSIGNED-TYPE
  end

annotations The result is the index type of an array type.

6.11.4.6 Bound Type

The bound type of a pointer type is the type of the variable pointed to by a value of the pointer type.

bound-type-of : *Typed* → *Environment* → *Typed*

bound-type-of (*type*)ρ ≙
  let *mk-Pointer-structure* (*btype*) = *structure-of* (*type*)ρ in 315
  btype

annotations The result is the bound type of a pointer type.

6.11.5 Other Type Functions

6.11.5.1 Return Types

return-types-of : *Expression-typed* → *Environment* → *Typed*

return-types-of (*type*)ρ ≙
  let *mk-Function-procedure-structure* (-, *return*) = *structure-of* (*type*) in 315
  return

6.11.5.2 Formal Parameter Types

parameter-types-of : *Expression-typed* → *Environment* → *Formal-parameters-typed*

parameter-types-of (*type*)ρ ≙
  let *struc* = *structure-of* (*type*) in 315
  cases *struc* :
    mk-Proper-procedure-structure (*parms*) → *parms*,
    mk-Function-procedure-structure (*parms*, -) → *parms*
  end

annotations Applied to the type of a value designator, this function returns the formal parameter structure and types for procedures, procedure constants, and procedure variables.

6.11.5.3 Auxiliary Functions for Records

field-types-of-record : *Expression-typed* → *Environment* → (*Identifier* $\xrightarrow{m}$ *Typed*)

field-types-of-record (*type*)ρ ≙
  let *mk-Record-structure* (*fields*) = *structure-of* (*type*)ρ in 315
  fields-of-fields-list (*fields*) 327

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations The result is a mapping whose domain is the field identifiers of the fixed part of a record together with the field identifiers of the variant component(s). In the mapping each field identifier is mapped to its declared type.

fields-of-fields-list : *Fields-list-structure* → (*Identifier* $\xrightarrow{m}$ *Typed*)

fields-of-fields-list (*fields*) ≙
  ⋃ {*fields-of-fields* (*field*) | *field* ∈ elems *fields*} 327

annotations Construct a mapping that is the union of the mappings constructed from each of the fields-structure components of the field-list-structure.

fields-of-fields : *Fields-structure* → (*Identifier* $\xrightarrow{m}$ *Typed*)

fields-of-fields (*field*) ≙
  cases *field* :
    mk-Fixed-fields-structure (-, -) → *fields-of-fixed-fields* (*field*), 327
    mk-Variant-fields-structure (-, -, -, -) → *fields-of-variant-fields* (*field*) 327
  end

annotations Construct a mapping from either a fixed-fields-structure or a variant-fields-structure component of a field-list-structure.

fields-of-fixed-fields : *Fixed-fields-structure* → (*Identifier* $\xrightarrow{m}$ *Typed*)

fields-of-fixed-fields (*field*) ≙
  let *mk-Fixed-fields-structure* (*ids*, *type*) = *field* in
  {*id* ↦ *type* |
      id ∈ elems *ids*}

annotations Construct a mapping whose domain is the field identifiers of all the variant components of a record, and in which each field identifier is mapped to its declared type.

fields-of-variant-fields : *Variant-fields-structure* → (*Identifier* $\xrightarrow{m}$ *Typed*)

fields-of-variant-fields (*field*) ≙
  let *mk-Variant-fields-structure* (*tag*, *tagt*, *variant*, *other*) = *field* in
  (if *tag* ≠ nil
  then {*tag* ↦ *tagt*}
  else { }) ∪ *fields-of-variants* (*variant*) ∪ *fields-of-fields-list* (*other*) 327 327

annotations Construct a mapping whose domain is the tag identifier, the field identifiers of the variant components of a record and the field identifiers of the ELSE component, and in which each field identifier is mapped to its declared type.

fields-of-variants : *Variant-structure** → (*Identifier* $\xrightarrow{m}$ *Typed*)

fields-of-variants (*variants*) ≙
  ⋃ {*fields-of-variant* (*variant*) | 327
      variant ∈ elems *variants*}

annotations Construct a mapping that relates the identifiers belonging to every variant component of a record type with their declared types.

fields-of-variant : *Variant-structure* → (*Identifier* $\xrightarrow{m}$ *Typed*)

fields-of-variant (*variant*) ≙
  let *mk-Variant-structure* (-, *fields*) = *variant* in
  fields-of-fields-list (*fields*) 327

annotations Construct a mapping that relates the field identifiers of a particular variant component of a record type with their declared types.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.11.5.4 Auxiliary Functions for Arrays

is-zero-array-type : *Typed* → *Environment* → $\mathbb{B}$

is-zero-array-type (*type*)ρ $\triangleq$
  is-array-type (*type*)ρ $\wedge$ 321
  let *itype* = *index-type-of* (*type*)ρ in 326
  host-type-of (*itype*)ρ = UNSIGNED-TYPE $\wedge$ 325
  minimum (*itype*)ρ = 0 329

annotations Check that a type is an array type with signed index starting at zero.

lower-bound-of : *Typed* → *Environment* → *Ordinal-value*

lower-bound-of (*type*)ρ $\triangleq$
  let *itype* = *index-type-of* (*type*)ρ in 326
  minimum (*itype*)ρ 329
pre *is-array-type* (*type*)ρ 321

annotations The result is the lower bound of an array.

upper-bound-of : *Typed* → *Environment* → *Ordinal-value*

upper-bound-of (*type*)ρ $\triangleq$
  let *itype* = *index-type-of* (*type*)ρ in 326
  maximum (*itype*)ρ 328
pre *is-array-type* (*type*)ρ 321

annotations The result is the upper bound of an array.

size-of-array : *Typed* → *Environment* → $\mathbb{N}$

size-of-array (*type*)ρ $\triangleq$
  upper-bound-of (*type*)ρ − *lower-bound-of* (*type*)ρ + 1 328 328
pre *is-array-type* (*type*)ρ 321

6.11.5.5 Auxiliary Functions for Open Arrays

is-open-array : *Expression-typed* → *Environment* → $\mathbb{B}$

is-open-array (*type*)ρ $\triangleq$
  is-Open-array-typed(*type*)

annotations Check that a type denotes an open array formal type.

6.11.5.6 Auxiliary Functions for Opaque Types

opaque-types-of : *Environment* → *Identifier*-set

opaque-types-of (ρ) $\triangleq$
  let *type* = ρ.*types* in
  $\{id \mid id \in \text{dom } type \cdot is\text{-}opaque\text{-}type\ (id)\rho\}$ 322

annotations The opaque types of an environment are those types whose structure is opaque.

6.11.6 Values Associated with a Type

The values associated with a type are the smallest and largest values that a value of that type may assume.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

maximum : *Typed* → *Environment* → *Value*

maximum (*type*)ρ $\triangleq$
  (*is-Basic-type*(*type*) → *maximum-of-basic-type* (*type*), 276
   is-Type-name(*type*) → let *struc* = *structure-of* (*type*)ρ in 315
                  if *is-Enumeration-structure*(*struc*)
                  then *maximum-of-enumeration* (*struc*) 329
                  else *maximum-of-subrange* (*struc*)) 329
pre *is-scalar-type* (*type*)ρ 322

annotations Return the largest value of a type.

maximum-of-enumeration : *Enumeration-structure* → *Value*

maximum-of-enumeration (*struc*) $\triangleq$
  let *mk-Enumeration-structure* (*values*) = *struc* in
  mk-Enumeration-value (*values* (*maxs* (inds *values*)), *values*) 614

annotations Return the largest value of an enumeration type. This is denoted by the last identifier that occurs in the sequence of identifiers of the enumeration type.

maximum-of-subrange : *Subrange-structure* → *Value*

maximum-of-subrange (*struc*) $\triangleq$
  let *mk-Subrange-structure* (-, *range*) = *struc* in
  let *mk-Set-value* (*value*) = *range* in
  maxs (*value*) 614

annotations Return the largest value of a subrange type.

minimum : *Typed* → *Environment* → *Value*

minimum (*type*)ρ $\triangleq$
  (*is-Basic-type*(*type*) → *minimum-of-basic-type* (*type*), 276
   is-Type-name(*type*) → let *struc* = *structure-of* (*type*)ρ in 315
                  if *is-Enumeration-structure*(*struc*)
                  then *minimum-of-enumeration* (*struc*) 329
                  else *minimum-of-subrange* (*struc*)) 329
pre *is-scalar-type* (*typed*)ρ 322

annotations Return the smallest value of a type.

minimum-of-enumeration : *Enumeration-structure* → *Value*

minimum-of-enumeration (*struc*) $\triangleq$
  let *mk-Enumeration-structure* (*values*) = *struc* in
  mk-Enumeration-value (hd *values*, *values*)

annotations Return the smallest value of an enumeration type. This is denoted by the first identifier that occurs in the sequence of identifiers of the enumeration type.

minimum-of-subrange : *Subrange-structure* → *Value*

minimum-of-subrange (*struc*) $\triangleq$
  let *mk-Subrange-structure* (-, *range*) = *struc* in
  let *mk-Set-value* (*value*) = *range* in
  mins (*value*) 614

annotations Return the smallest value of a subrange type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

values-of : *Typed* → *Environment* → *Value*-set

values-of (*type*)ρ ≜
  (*is-Basic-type*(*type*) → *values-of-basic-type* (*type*), 277
   is-Type-name(*type*) → let *struc* = *structure-of* (*type*) in 315
                  if *is-Enumeration-structure*(*struc*)
                  then *values-of-enumeration* (*struc*) 330
                  else *values-of-subrange* (*struc*)) 330
pre *is-scalar-type* (*type*)ρ 322

annotations Return the set of values associated with a type.

values-of-enumeration : *Enumeration-structure* → *Value*-set

values-of-enumeration (*struc*) ≜
  let *mk-Enumeration-structure* (*values*) = *struc* in
  {*mk-Enumeration-value* (*values*(*i*), *values*) | *i* ∈ inds *values*}

annotations Return the set of value associated with an enumeration type. This is the set of values containing the identifiers that occurs in the sequence of identifiers of the enumeration type.

val-enumeration : $\mathbb{N}$ × *Identifier*$^+$ → *Enumeration-value*

val-enumeration (*n*, *ids*) ≜
  mk-Enumeration-value (*ids*(*n* + 1), *ids*)
pre *n* + 1 ∈ inds *ids*

annotations Convert a natural number into an Enumeration-value.

values-of-subrange : *Subrange-structure* → *Value*-set

values-of-subrange (*struc*) ≜
  let *mk-Subrange-structure* (-, *range*) = *struc* in
  let *mk-Set-value* (*value*) = *range* in
  value

annotations Return the set of value associated with an subrange type.

ordered-values-of : *Typed* → *Environment* → *Value**

ordered-values-of (*type*)ρ ≜
  let *sv* : *Value** be st *i* < *j* ⇒ *sv* (*i*) < *sv* (*j*) ∧
                inds *sv* = {1, . . . , card *values-of* (*type*)ρ} ∧
                elems *sv* = *values-of* (*type*)ρ in 329
  sv

annotations Return a sequence of values associated with a type. This sequence is ordered by the values from which it is constructed.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.12 The Storage Model and Auxiliary Formal Definitions

6.12.1 The Program State

state *Program-state* of
  storage : *Coroutine-storage*
  machine-storage : *Value-store*
  aliases : *Alias-relation*
  dens : [*Proc-dens*]
  termination : *Termination-stack*
  halting : $\mathbb{B}$
  exception-sources : *Exception-sources*
  handlers : *Handlers*

  init *mk-State* (*storage*, *machine-storage*, *aliases*, *dens*, *termination*, *halting*, *exception-sources*, *handlers*) $\triangleq$
    storage = {$\mapsto$} $\wedge$
    machine-storage = {$\mapsto$} $\wedge$
    aliases = { } $\wedge$
    dens = {$\mapsto$} $\wedge$
    termination = [ ] $\wedge$
    halting = false $\wedge$
    exception-sources = { } $\wedge$
    handlers = {$\mapsto$}
end

6.12.2 The External Program State

types

  External-program-state :: *program-storage-pool* : [*Storage-pool*]
                              external-store : *Value-store*

  inv *mk-External-program-state* (-, -) $\triangleq$
    true

annotations The *program-storage-pool* component of an *External-program-state* is an implementation-dependent model of storage available to a Modula-2 program that consists of a set of *Locs* that denotes the initial free storage available to a program. The *external-store* component of an *External-program-state* is a model of the storage locations that a Modula-2 program can have access to via an address that is calculated from the machine address component of a variable declaration (see 6.12.8.2.1).

6.12.2.1 Operations to Initialize and Terminate the Program State

read-external-program-state : *External-program-state* $\xrightarrow{o}$ ()

read-external-program-state (*estate*) $\triangleq$
  let *mk-External-program-state* (*psp*, *es*) = *estate* in
  let *st* = *mk-Store* (*psp*, {$\mapsto$}) in
  let *cst* = *mk-Coroutine-store* (*st*, false, [ ], [NORMAL-EXECUTION], *initprot*) in
  (*storage* := {PROGRAM $\mapsto$ *cst*};
   machine-storage := *es*)

annotations Initializes the state with the external program state. As part of the call of *read-external-program-state*, the changes to the library state of abstract implementation modules, as defined by the operation *read-external-* < *libraryname* > *-state*, should be carried out.

write-external-program-state : () $\xrightarrow{o}$ *External-program-state*

write-external-program-state () $\triangleq$
  def *result* = *machine-storage* in

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

(*storage* := {↦};
machine-storage := {↦};
aliases := { };
dens := {↦};
termination := [];
halting := true;
exception-sources := { };
handlers := {↦};
return *mk-External-program-state* (nil , *result*))

annotations This operation returns the final state of any external storage and de-allocates all storage in the storage model. As part of the call of *write-external-program-state*, the extraction of changes to the library state of abstract implementation modules, as defined by the operation *write-external-* < *libraryname* > *-state*, should be carried out.

6.12.3 The Store

6.12.3.1 Storage

types

Coroutine-storage = *Coroutine* $\xrightarrow{m}$ *Coroutine-store*;

Coroutine-store :: *store* : *Store*
terminating : $\mathbb{B}$
terminations : *Termination-stack*
exceptions : *Exception-context-stack*
protection : *Protection*;

Store :: *pool* : *Storage-pool*
values : *Value-store*

inv *mk-Store* (*pool*, *values*) $\triangleq$
pool-locs (*pool*) ∩ dom *values* = { } 338

annotations The invariant requires that the storage locations allocated to a coroutine are disjoint from the storage locations used by the coroutine. The operation *check-changes* requires that once storage has been allocated to a coroutine, it must not be used by the coroutine that was originally allocated the storage. Taken together, this means that an implementation *may* re-use the storage passed to a coroutine for the workspace of that coroutine.

types

Storage-pool :: *allocated* : *Address-value*
rest : implementation-dependent

annotations The *rest* component is an implementation-dependent type consisting of a set of *Loc*, posssibly derived from the *allocated* component, from which storage can be allocated.

types

Value-store = *Loc* $\xrightarrow{m}$ *Storable-value*;

Storable-value = *Value* | UNDEFINED;

Loc-pair :: *first* : *Loc*
second : *Loc*;

Alias-relation = *Loc-pair*-set

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.12.3.2 Values

types

Value = *Elementary-value* | *Array-value* | *Record-value*;

Elementary-value = *Basic-value*
| *Enumeration-value*
| *Set-value*
| *Pointer-value*
| *Procedure-id*
| *Protection*
| *System-value*;

Basic-value = *Number* | *String-value* | $\mathbb{B}$ | char | NIL-VALUE;

Number = $\mathbb{N}$ | $\mathbb{Z}$ | $\mathbb{R}$ | $\mathbb{C}$

annotations Strictly speaking $\mathbb{N} \subset \mathbb{Z} \subset \mathbb{R} \subset \mathbb{C}$, however, they are shown as separate sets for clarity.

types

String-value = *Char**;

Ordinal-value = $\mathbb{Z}$ | $\mathbb{B}$ | *Char* | *Enumeration-value*;

Enumeration-value :: *value* : *Identifier*
order : *Identifier**

inv *mk-Enumeration-value* (*value*, *order*) $\triangleq$ *value* $\in$ elems *order* ;

Set-value :: *value* : *Ordinal-value*-set;

Pointer-value = *Address-value* | NIL-VALUE;

Address-value :: *value* : *Address*
type : *Typed*
size : $\mathbb{N}$;

Address = *Variable*;

Protection = INTERRUPTIBLE | UNINTERRUPTIBLE | *Other-protection*;

Other-protection = implementation-defined

values

initprot : *Protection* = implementation-defined — initial protection value

properties

Other-protection is an implementation-defined type that defines a set of values, that together with the two values INTERRUPTIBLE and UNINTERUPTIBLE has a partial order. The partial order relation shall be such that:

$$\forall p : Protection \cdot \text{INTERRUPTIBLE} \leq p \leq \text{UNINTERRUPTIBLE}$$

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

types

System-value = *Coroutine* | *Loc* | *Exception-source*;

Coroutine = PROGRAM | token$_{coroutine\text{-}id}$;

Loc = token$_{Loc}$;

Exception-source = token$_{Exception\text{-}source}$;

Array-value = *Ordinal-value* $\xrightarrow{m}$ *Value*;

Record-value = *Identifier* $\xrightarrow{m}$ *Value*

6.12.4 Procedure Denotations

types

Procdens = *Procedure-id* $\xrightarrow{m}$ *Program-cont*

6.12.5 Termination

types

Termination-stack = *Module-finalization**;

Module-finalization = *Sourced-module-finalization* | *Abstract-module-finalization*;

Sourced-module-finalization :: *final* : *Block-body*
env : *Environment*;

Abstract-module-finalization :: *final* : *Termination-actions*
env : *Environment*;

Initialization-actions = *Program-cont*;

Termination-actions = *Program-cont*

6.12.6 Exceptions

types

Exception-sources = *Exception-source*-set;

Exception-context-stack = *Exception-context**;

Exception-context = NORMAL-EXECUTION | *Raised-exception*;

Raised-exception = *Language-exception* | *User-exception*;

Language-exception :: *exception* : *Exception-value*
message : *Message*;

User-exception :: *source* : *Exception-source*
exception : $\mathbb{N}$
message : *Message*;

Exception-value = *Enumeration-value*;

Message = *Char**

annotations Each coroutine has a non-empty stack of exception contexts associated with it. The top context on the stack indicates whether the coroutine is executing in the state of normal execution, exceptional execution due to the raising of a language exception, or exceptional execution due to the raising of a user exception. The stack is initialized to indicate normal execution; a further normal execution context is pushed on the stack on the execution of a block body that has an exceptional part. If an exception is raised, the top exception context is replaced by a context that holds information about the exception.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.12.7 Interrupt Handlers

types

$Handlers = Interrupt\text{-}source \xrightarrow{m} Handler;$

$Interrupt\text{-}source =$ implementation-defined

annotations *Interrupt-source* is an implementation-defined set of values

types

$Handler :: id : Coroutine$
$\quad from : [Loc]$

6.12.8 Operations on Storage

6.12.8.1 Operations Accessing Storage

$merged\text{-}value\text{-}store : Coroutine\text{-}storage \to Value\text{-}store$

$merged\text{-}value\text{-}store\,(storage) \triangleq$
  merge $\{st.store.values \mid mk\text{-}Coroutine\text{-}store\,(st, -, -, -, -) \in \text{rng } storage\}$

$exists\text{-}loc : Loc \xrightarrow{o} \mathbb{B}$

$exists\text{-}loc\,(var) \triangleq$
  let $store = merged\text{-}value\text{-}store\,(storage)$ in 335
  return $var \in \text{dom } store$

annotations Check the existence of a storage location.

$contents : Loc \xrightarrow{o} Value$

$contents\,(loc) \triangleq$
  let $store = merged\text{-}value\text{-}store\,(storage)$ in 335
  return $store(loc)$
pre $loc \in \text{dom } merged\text{-}value\text{-}store\,(storage)$

annotations Access the value of a storage location associated with a variable.

$change\text{-}value : Loc \times Storable\text{-}value \xrightarrow{o} ()$

$change\text{-}value\,(var, val) \triangleq$
  let $aliased = \{other \mid mk\text{-}Loc\text{-}pair(var, other) \in aliases\}$ in
  let $changes = \{loc \mapsto side\text{-}effect\text{-}of\text{-}change\,(var, val, loc) \mid loc \in aliased\} \dagger \{var \mapsto val\}$ in 336
  if $check\text{-}changes\,(changes, storage)$ 335
  then for all $loc \in$ dom $changes$
    do let $cid \in$ dom $storage$ be st $loc \in$ dom $storage(cid).store.values$ in
      $storage(cid).store.values(loc) := changes(loc)$
  else $non\text{-}mandatory\text{-}detection$(STORAGE-NOT-OWNED) 397
pre $var \in \text{dom } merged\text{-}value\text{-}store\,(storage)$

annotations Change the value of a storage location. That storage location could be aliased with other storage locations caused by passing a variable using a variable parameter of a system storage formal type; this will cause an implementation-defined change to these storage locations.

$all\text{-}storage\text{-}pools : Coroutine\text{-}storage \to Storage\text{-}pool\text{-set}$

$all\text{-}storage\text{-}pools\,(storage) \triangleq$
  $\{st.store.pool \mid mk\text{-}Coroutine\text{-}store\,(st, -, -, -, -) \in \text{rng } storage\}$

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

check-changes : *Value-store* × *Coroutine-storage* → 𝔹

check-changes (*changes*, *storage*) ≙
  let *locs* = dom *changes* in
  ∀ *mk-Storage-pool* (*allocated*, -) ∈ *all-storage-pools* (*storage*) · 335
    let *mk-Address-value* (*addr*, -, -) = *allocated* in
    locs ∩ *locs-of* (*addr*) = { } 338

annotations Check that the locations denoted by the first parameter are not contained in storage that has been allocated to a coroutine.

6.12.8.2 Storage Aliasing

side-effect-of-change : *Variable* × *Value* × *Loc* → *Elementary-loc-value*

side-effect-of-change (*var*, *val*, *loc*) ≙
  implementation-defined

annotations The storage location denoted by *loc* is aliased with (part of) the storage location(s) denoted by *var*. Thus a change to the contents of the storage location modelled by *var* when the value *val* is assigned to it may change the contents of the storage location modelled by *loc* — return an implementation-defined value that describes any changes to the contents of the storage location *loc* because of the assignment.

alias-set : *Variable* × *Variable* → *Loc-pair*-set

alias-set (*vara*, *varb*) ≙
  let *locsa* = *locs-of* (*vara*) in
  let *locsb* = *locs-of* (*varb*) in
  {*mk-Loc-pair*(*fst*, *snd*) | *fst*, *snd* ∈ *locsa* ∪ *locsb* · *fst* ≠ *snd*}

annotations Return the set of alias pairs that could occur if two variables share storage.

set-aliases : *Variable* × *Variable* $\xrightarrow{o}$ *Value*

set-aliases (*vara*, *varb*) ≙
  aliases := *alias-set* (*vara*, *varb*) ∪ *aliases* 336

annotations Update the alias information with the set of alias pairs that could occur if two variables share storage.

remove-aliases : *Loc*-set $\xrightarrow{o}$ ()

remove-aliases (*locs*) ≙
  aliases := {*pair* | *pair* ∈ *aliases* · *first*(*pair*) ∉ *locs* ∨ *second*(*pair*) ∉ *locs*}

annotations Remove alias pairs from storage.

6.12.8.2.1 Aliasing Machine Addresses

alias-machine-address : *Loc* × *Value* $\xrightarrow{o}$ ()

alias-machine-address (*loc*, *val*) ≙
  let *maddr* = *calculate-address* (*val*) in 336
  aliases := {*mk-Loc-pair*(*loc*, *maddr*)} ∪ {*mk-Loc-pair*(*maddr*, *loc*)} ∪ *aliases*

annotations Alias a machine address with a location.

calculate-address : *Value* → *Loc*

calculate-address (*val*) ≙
  implementation-defined

annotations Convert a machine address to a location. The value of type *Loc* returned by this function must be in the domain of the *machine-storage* component of the state.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.12.8.3 Operations to Allocate and Deallocate Storage

allocate-program-storage : *Typed* → *Environment* $\xrightarrow{o}$ *Variable*

allocate-program-storage (*type*)ρ $\triangleq$
  def *store* = *get-coroutine-storage*() in 339
  let *mk-*(*var*, *nstore*) = *a-typed* (*type*, *store*) in 114
  if *can-allocate-storage* (*store*, *nstore*) 346
  then (*put-coroutine-storage*(*nstore*) ; 339
    return *var*)
  else let *handler* = *current-exception-handler* (ρ) in 318
    mandatory-detection(WORKSPACE-OVERFLOW) *handler* 396

annotations An exception is raised if there is not enough storage available.

deallocate-program-storage : *Variable* $\xrightarrow{o}$ ()

deallocate-program-storage (*var*) $\triangleq$
  deallocate-variables({*var*}) 338

allocate-abstract-storage : () $\xrightarrow{o}$ *Loc*

allocate-abstract-storage () $\triangleq$
  def *store* = *get-coroutine-storage*() in 339
  let *mk-*(*loc*, *nstore*) = *generate-loc* (*store*) in 338
  (*put-coroutine-storage*(*nstore*) ; 339
   return *loc*)

deallocate-abstract-storage : *Loc* $\xrightarrow{o}$ ()

deallocate-abstract-storage (*loc*) $\triangleq$
  def *store* = *get-coroutine-storage*() in 339
  let *nstore* = *deallocate-locs* ({*loc*}) in 338
  put-coroutine-storage(*nstore*) ; 339

deallocate-storage : *Identifier*-set → *Environment* $\xrightarrow{o}$ ()

deallocate-storage (*ids*)ρ $\triangleq$
  let *denv* = *restrict-environment* (*ids*)ρ in 320
  deallocate-storage-of-environment(*denv*) 337

annotations Deallocate the storage associated with a set of identifiers.

deallocate-storage-of-environment : *Environment* $\xrightarrow{o}$ ()

deallocate-storage-of-environment ρ $\triangleq$
  let *mk-Environment*(-, -, -, *vars*, -, *mods*, -, -, -, -, -) = ρ in
  (*deallocate-modules*(*mods*) ; 337
   deallocate-variables(rng *vars*)) 338

deallocate-modules : *Module-environment* $\xrightarrow{o}$ ()

deallocate-modules (*menv*) $\triangleq$
  let *env* = *get-module-environment* (*menv*) in 317
  deallocate-module-env(*env*(*name*)) ; 338

annotations De-allocate the storage of the modules in the environment.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
deallocate-module-env : Environment ~o~> ()
deallocate-module-env (env) ≜
  if env = {↦}
  then skip
  else let name ∈ dom env in
       (deallocate-storage-of-environment(env(name)) ;                    337
        deallocate-modules({name} ⩤ env) )                                337
```

```
deallocate-variables : Variable-set ~o~> ()
deallocate-variables (vars) ≜
  let locs = ⋃{locs-of(var) | var ∈ vars} in
  def store = get-coroutine-storage() in                                   339
  let nstore = deallocate-locs (locs, store) in                            338
  (put-coroutine-storage(nstore) ;                                         339
   remove-aliases(locs) )                                                  336
```

annotations Deallocate the storage associated with a set of variables.

```
locs-of : Variable → Loc-set
locs-of (v) ≜
  cases v :
    mk-Elementary-variable(loc, -)      → {loc},
    mk-Array-variable(vars)             → ⋃{locs-of(var) | var ∈ rng vars},
    mk-Record-variable(fields)          → ⋃{locs-of(field) | field ∈ rng fields},
    mk-Tag-variable(var, -, -)          → locs-of(var),
    mk-Variant-variable(var, tagloc, -) → locs-of(var) ∪ {tagloc},
    mk-Tagged-variable(var, tagloc, -)  → locs-of(var) ∪ {tagloc}
  end
```

annotations Return the set of locations associated with a variable.

6.12.8.4 Operations to Allocate and Deallocate Storage Locations

```
generate-loc (st : Store) r : Loc × Store
post let r : Loc be st r ∉ get-locs (st) in                                338
     let mk-Store (pool, values) = st in
     let nvalues = values ∪ {r ↦ UNDEFINED} in
     mk-Store (pool, nvalues)
```

annotations Generate a new storage location (one currently not in the domain of the storage map) and set its value to undefined. The choice of storage location is non-deterministic.

```
get-locs : Store → Loc-set
get-locs (store) ≜
  let mk-Store (pool, values) = store in
  pool-locs (pool) − dom values                                            338
```

```
pool-locs : Storage-pool → Loc-set
pool-locs (pool) ≜
  implementation-dependent
```

annotations Return an implementation-dependent set of *Locs* derived from the storage pool.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

deallocate-locs : *Loc*-set × *Store* → *Store*

deallocate-locs (*locs*, *st*) ≙
  let *mk-Store* (*pool*, *values*) = *st* in
  let *nvalue* = *locs* ⩤ *values* in
  mk-Store (*pool*, *nvalues*)

annotations Deallocate storage defined by a set of locations — return the set of storage locations (currently in the domain of the storage map) and return them to the storage pool.

6.12.8.5 Access Coroutine Storage

get-coroutine-storage : () $\xrightarrow{o}$ *Store*

get-coroutine-storage () ≙
  def *cid* = *current-coroutine*() in
  return *storage*(*cid*).*store*

346

put-coroutine-storage : *Store* $\xrightarrow{o}$ ()

put-coroutine-storage (*st*) ≙
  def *cid* = *current-coroutine*() in
  storage(*cid*) := *μ* (*storage*(*cid*), *store* ↦ *st*)

346

6.12.9 Accessing the Procedure Denotation Component of the State

make-proper-procden : (*Arguments* → *Exception-handler* $\xrightarrow{o}$ ()) → *Proper-procedure-value*)

make-proper-procden (*proc*) ≙
  let *p* = *λ args* : *Arguments* ·
    λ handler : *Exception-handler* ·
      proc(*args*) *handler* in
  mk-Proper-Procedure-value (*p*)

make-function-procden : (*Arguments* → *Exception-handler* $\xrightarrow{o}$ *Value*) → *Function-procedure-value*)

make-function-procden (*fun*) ≙
  let *f* = *λ args* : *Arguments* ·
    λ handler : *Exception-handler* ·
      fun(*args*) *handler* in
  mk-Function-Procedure-value (*f*)

store-denotation : *Program-cont* $\xrightarrow{o}$ *Procedure-id*

store-denotation (*pcont*) ≙
  def *pid* = *allocate-procedure-id*() in
  (*dens* := *dens* † {*pid* ↦ *pcont*};
   return *pid*)

340

call-proper-procedure-value : *Procedure-id* → *Args* → *Exception-handler* $\xrightarrow{o}$ ()

call-proper-procedure-value (*pid*)*args handler* ≙
  def *proc* = *dens*(*pid*) in
  proc(*args*) *handler*

call-function-procedure-value : *Procedure-id* → *Args* → *Exception-handler* $\xrightarrow{o}$ *Value*

call-function-procedure-value (*pid*)*args handler* ≙
  def *fun* = *dens*(*pid*) in

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

fun(*args*) *handler*

store-procedure-dens : *Proc-dens* $\xrightarrow{o}$ ()

store-procedure-dens (*adens*) $\triangleq$
dens := *dens* $\cup$ *adens*

annotations The well-formed conditions for abstract modules require that all procedure identifiers used in the program be distinct.

allocate-procedure-id : () $\rightarrow$ *Procedure-id*

allocate-procedure-id () $\triangleq$
let *pid* : *Procedure-id* be st *pid* $\notin$ dom *dens* in
return *pid*

6.12.10 Program Initialization and Program Termination

6.12.10.1 Program Initialization

set-initializing : () $\xrightarrow{o}$ ()

set-initializing () $\triangleq$
(*halting* := false;
storage(PROGRAM).*terminating* := false)

annotations Set the program termination flag to signal that initialization is occurring. The main program and each coroutine will be able to enquire if it is terminating.

6.12.10.2 Program Termination

set-terminating : () $\xrightarrow{o}$ ()

set-terminating () $\triangleq$
(def *caller* = *current-coroutine*() in 346
storage(*caller*).*terminating* := true;
storage(*caller*).*terminations* := *terminations*)

annotations Set this coroutine as terminating.

is-terminating : () $\xrightarrow{o}$ $\mathbb{B}$

is-terminating () $\triangleq$
def *caller* = *current-coroutine*() in 346
return *storage*(*caller*).*terminating*

annotations Check if this coroutine is terminating.

is-program-terminating : () $\xrightarrow{o}$ $\mathbb{B}$

is-program-terminating () $\triangleq$
def *cstore* = *storage* in
return *check-for-termination* (*cstore*) 340

annotations Check if any coroutine is terminating.

check-for-termination : *Coroutine-storage* $\rightarrow$ $\mathbb{B}$

check-for-termination (*cs*) $\triangleq$
$\exists$ *cr* $\in$ dom *cs* $\cdot$ *cs*(*cr*).*terminating*

annotations Check if any coroutine is terminating.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

check-termination : *Coroutine* $\xrightarrow{o}$ $\mathbb{B}$

check-termination (*cid*) $\triangleq$
  return *storage*(*cid*).*terminating*

annotations Check if the coroutine with the identifier *cid* is terminating.

queue-termination : *Module-termination* $\xrightarrow{o}$ ()

queue-termination (*final*) $\triangleq$
  terminations := [*final*] $\frown$ *terminations*

next-module : () $\xrightarrow{o}$ [*Module-finalization*]

next-module () $\triangleq$
  def *caller* = *current-coroutine*() in
  if *storage*(*caller*).*terminations* = []
  then return nil
  else def [*next*] $\frown$ *rest* = *storage*(*caller*).*terminations* in
    (*storage*(*caller*).*terminations* := *rest*;
     return *next*)

346

annotations Return the next module finalization code if one exists, otherwise return nil .

termination-actions : () $\xrightarrow{o}$ *External-program-state*

termination-actions () $\triangleq$
  (implementation-dependent actions ;
   write-external-program-state())

331

annotations Implementation-dependent housekeeping to complete execution of a program.

6.12.10.3 Program Halting

note-HALT : () $\xrightarrow{o}$ *External-program-state*

note-HALT () $\triangleq$
  (*halting* := true;
   termination() ;
   termination-actions())

41
341

annotations Set the program termination flag to signal that termination is occuring.

check-halting : () $\xrightarrow{o}$ $\mathbb{B}$

check-halting () $\triangleq$
  return *halting*

annotations Check if the program is halting.

6.12.11 Exception Handling

exception-actions : () $\xrightarrow{o}$ ()

exception-actions () $\triangleq$
  (implementation-dependent actions ;
   termination-actions())

341

annotations An exception has no exception handler — these are implementation-dependent actions that are to be performed when the program is terminated exceptionally.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

push-exception-context : *Exception-context* $\xrightarrow{o}$ ()

push-exception-context (*context*) $\triangleq$
  def *caller* = *current-coroutine*() in
  storage(*caller*).*exceptions* := [*context*] $\curvearrowright$ *storage*(*caller*).*exceptions* 346

annotations Push an exception context.

pop-exception-context : () $\xrightarrow{o}$ ()

pop-exception-context () $\triangleq$
  def *caller* = *current-coroutine*() in
  storage(*caller*).*exceptions* := tl *storage*(*caller*).*exceptions* 346

annotations Pop an exception context.

top-exception-context : () $\xrightarrow{o}$ *Exception-context*

top-exception-context () $\triangleq$
  def *caller* = *current-coroutine*() in
  return hd *storage*(*caller*).*exceptions* 346

annotations Read the current exception context.

swop-exception-context : *Exception-context* $\xrightarrow{o}$ ()

swop-exception-context (*context*) $\triangleq$
  (*pop-exception-context*() ; 342
   push-exception-context(*context*)) 342

annotations Read the current exception context.

is-exceptional : *Exception-context* $\xrightarrow{o}$ $\mathbb{B}$

is-exceptional (*context*) $\triangleq$
  context $\neq$ NORMAL-EXECUTION

annotations Return *true* iff the given exception context indicates the state of exceptional execution.

raise-exception : *Raised-exception* $\rightarrow$ *Exception-handler* $\xrightarrow{o}$ ()

raise-exception (*context*)*handler* $\triangleq$
  (*swop-exception-context*(*context*) ; 342
   execute(*handler*)) 345

annotations Raise an exception under the current exception handler with the exception context.

6.12.11.1 Allocation of Exception Sources

allocate-source : () $\xrightarrow{o}$ *Exception-source*

allocate-source () $\triangleq$
  (
  — an implementation-dependent choice of:
  — either
  def *allocated* = *exception-sources* in
  let *unused* : *Exception-source* be st *unused* $\notin$ *allocated* in
  (*exception-sources* := *exception-sources* $\cup$ {*unused*};
   return *unused*
   — or
   return nil))

annotations Allocate a source, or return nil if no more *Exception-source* values are availablle.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.12.12 Interrupt Handlers

attach : *Interrupt-source* $\xrightarrow{o}$ ()

attach (*source*) $\triangleq$
  def *caller* = *current-coroutine*() in
  let *new-handler* = *mk-Handler* (*caller*, nil) in
  handler := *handler* † {*source* ↦ *new-handler*}

346

annotations Attach the calling coroutine to a source of interrupts.

detach : *Interrupt-source* $\xrightarrow{o}$ ()

detach (*source*) $\triangleq$
  def *caller* = *current-coroutine*() in
  handler := *remove* (*source*, *caller*, *handlers*)

346
343

annotations Detach the calling coroutine from a source of interrupts.

remove : *Interrupt-source* × *Coroutine* × *Handlers* → *Handlers*

remove (*source*, *caller*, *handlers*) $\triangleq$
  if *source* ∈ dom *handlers* ∧
    let *hand* = *handlers*(*source*) in *hand.id* = *caller*
  then {*source*} ⩤ *handlers*
  else *handlers*

annotations Return the identity of the currently executing coroutine.

target-for-interrupted-coroutine : *Loc* $\xrightarrow{o}$ ()

target-for-interrupted-coroutine (*loc*) $\triangleq$
  def *caller* = *current-coroutine*() in
  let *source* ∈ dom *handlers* be st *handlers*(*source*).*id* = *caller* in
  let *new-handler* = *mk-Handler* (*caller*, *loc*) in
  handlers(*source*) := *new-handler*

346

annotations Set a handler for an interrupt.

is-attached : () $\xrightarrow{o}$ $\mathbb{B}$

is-attached () $\triangleq$
  def *caller* = *current-coroutine*() in
  return ∃ *source* ∈ dom *handlers* ·
    handlers(*source*) = *caller*

346

annotations Check that the currently executing coroutine is attached to a source of interrupts.

transfer-to-handler : *Interrupt-source* $\xrightarrow{o}$ ()

transfer-to-handler (*source*) $\triangleq$
  def *caller* = *current-coroutine*() in
  let *mk-Handler* (*id*, *from*) = *handlers*(*source*) in
  (*assign*(*from*, *caller*) ;
   resume(*id*))

346
150
345

annotations An event connected to the coroutine calling `IOTRANSFER` will cause the coroutine to resume at the statement following the call to `IOTRANSFER` and place the identity of the coroutine that was interrupted in the first argument.

program-handlers : () $\xrightarrow{o}$ *Handlers*

program-handlers () $\triangleq$
  return *handlers*

annotations Return the handlers of the program from the program state.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.12.13 Accessing the Protection Component of the State

switch-protection : *Protection* → *Environment* $\xrightarrow{o}$ *Environment*

switch-protection (*val*)ρ $\triangleq$
  def *oldprot* = *get-protection*() in 344
  (*set-protection*(*val*) ; 344
   return *oldprot*)

get-protection : () $\xrightarrow{o}$ *Protection*

get-protection () $\triangleq$
  def *caller* = *current-coroutine*() in 346
  return *storage*(*caller*).*protection*

annotations Read the state and return the current protection of the caller.

is-less-restrictive-protection : *Protection* $\xrightarrow{o}$ $\mathbb{B}$

is-less-restrictive-protection (*val*) $\triangleq$
  def *caller* = *current-coroutine*() in 346
  let *protection* = *storage*(*caller*).*protection* in
  def *comp* = *is-protection-less-than-or-equal*(*protection*, *val*) in 217
  return $\neg$ *comp*

annotations Read the state and return *true* iff the parameter is less restrictive than the current protection of the caller.

set-protection : *Protection* $\xrightarrow{o}$ ()

set-protection (*val*) $\triangleq$
  def *caller* = *current-coroutine*() in 346
  storage(*caller*).*protection* := *val*

annotations Add the new protection to the state.

6.12.14 Coroutine and Continuations

6.12.14.1 The Coroutine Environment

types

Coroutine-env :: *mcont*: *Coroutine* $\xrightarrow{m}$ *Program-cont*
  caller : *Coroutine*

6.12.14.2 The Continuations

types

Exception-handler = *Program-cont*;

Program-cont = *Proper-procedure-cont* | *Function-procedure-cont*;

Proper-procedure-cont = *Cmdcont* → *Coroutine-env* → *Statecont*;

Function-procedure-cont = *Value* → *Cmdcont* → *Coroutine-env* → *Statecont*;

Cmdcont = *Coroutine-env* → *Statecont*;

Statecont = *State* → *Answer*;

Answer = *External-state*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.12.14.3 Operations on Continuations

$c\text{-}exit : Exit \to Environment \xrightarrow{o} ()$

$c\text{-}exit\,(ex)\rho \triangleq$
  return $\rho.conts\,(ex)$

$c\text{-}tixe : Continuation \to Environment \xrightarrow{o} Environment$

$c\text{-}tixe\,(cont)\rho \triangleq$
  let $oldcont = \rho.conts$ in
  let $newcont = oldcont \dagger cont$ in
  let $\rho_{new} = \mu\,(\rho, conts \mapsto newcont)$ in
  return ρ_{new}

$execute : Program\text{-}cont \xrightarrow{o} ()$

$execute\,(cont)\rho \triangleq$
  $cont()$

annotations Continue with the continuation *cont*.

values

$stop = \lambda\, c \cdot \lambda\, \sigma \cdot \text{STOP} \circ \lambda\, c \cdot \lambda\, \sigma \cdot terminate\text{-}actions;$

$cenv = mk\text{-}Coroutine\text{-}env(\{\}, \text{PROGRAM})$

$initialize : Coroutine \times Program\text{-}cont \xrightarrow{o} ()$

$initialize\,(id, proc) \triangleq$
  $\lambda\, c \cdot \lambda\, e \cdot$ let $mk\text{-}Coroutine\text{-}env(map, caller) = e$ in
    $mk\text{-}Coroutine\text{-}env(map \dagger \{id \mapsto proc\}, caller)$

annotations Initialize a coroutine by adding to the coroutine environment.

$resume : Identifier \xrightarrow{o} ()$

$resume\,(id) \triangleq$
  $\lambda\, c \cdot \lambda\, e \cdot$ let $mk\text{-}Coroutine\text{-}env(map, caller) = e$ in
    let $nmap = map \dagger \{caller \mapsto c\}$ in
    $nmap(id)(nmap, id)$

annotations Resume a coroutine by saving the current execution of the coroutine that is currently being executed in the environment, and continuing the execution of the coroutine to be resumed.

$allocate\text{-}coroutine\text{-}id : () \xrightarrow{o} Coroutine$

$allocate\text{-}coroutine\text{-}id \triangleq$
  $\lambda\, c \cdot \lambda\, e \cdot$ let $mk\text{-}Coroutine\text{-}env(map, caller) = e$ in
    let $id : Coroutine\text{-}Id$ be st $id \notin \text{dom}\ map$ in
    id

annotations Allocate an unused coroutine identifier.

$reserve\text{-}coroutine\text{-}workspace : Coroutine \times Loc \times \mathbb{N} \times Protection \xrightarrow{o} ()$

$reserve\text{-}coroutine\text{-}workspace\,(cid, space, size, prio) \triangleq$
  def $store = coroutine\text{-}locs(space, size)$ in
  let $cloc = mk\text{-}Coroutine\text{-}store\,(store, \text{false}, [\text{NORMAL-EXECUTION}], [\,], prio)$ in
  $add\text{-}to\text{-}store(cid, cloc)$

annotations Construct coroutine storage and add it to the state. The storage consists of an empty protection stack and the storage for the variables and control information for a coroutine.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

coroutine-locs : *Address-value* × $\mathbb{N} \xrightarrow{o}$ *Storage-pool*

coroutine-locs (*space*, *size*) $\triangleq$
implementation-dependent

can-allocate-storage : *Store* × *Store* $\xrightarrow{o} \mathbb{B}$

can-allocate-storage (*space*, *size*) $\triangleq$
implementation-dependent
— an implementation-dependent function that checks
— that adequate space for storage can be allocated

add-to-store : *Coroutine* × *Coroutine-storage* $\xrightarrow{o}$ ()

add-to-store (*cid*, *cloc*) $\triangleq$
storage := *storage* † {*cid* ↦ *cloc*}

annotations Add the storage for a coroutine to the state.

current-coroutine : () $\xrightarrow{o}$ *Coroutine*

current-coroutine () $\triangleq$
$\lambda\, c \cdot \lambda\, e \cdot \lambda\, \sigma \cdot$ let *mk-Coroutine-env* (-, *caller*) = *e* in
return *caller*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Section 7 : System Modules

System modules are not compilation modules, in the sense that they do not have an actual definition or implementation provided in a library, or by the user. They encapsulate predefined types, constants and procedures, for which the predefined identifiers are not pervasive.

For information only, the interfaces to system modules are shown informally in this part of ISO/IEC 10514 as definition modules. The required semantics cannot be obtained by interpreting these system definition modules according to the rules for separate modules as given in section 6. In particular, as predefined entities, the procedures and functions of a system module are not assignment-compatible with any procedure type. Also, types appearing as opaque types are system basic types and cannot be used to implement user-defined opaque types.

This part of ISO/IEC 10514 defines five system modules, namely `SYSTEM`, `COROUTINES`, `TERMINATION`, `EXCEPTIONS`, and `M2EXCEPTION`.

CHANGE — In *Programming in Modula-2* the only system module is `SYSTEM` and this is described as a *pseudo-module*. Coroutine facilities have been moved from `SYSTEM` to the system module `COROUTINES` in this part of ISO/IEC 10514.

Declaration Semantics

values

$$\rho_{sys\text{-}modules} = \text{let } sysmods = \{\text{SYSTEM} \mapsto \rho_{SYSTEM},\ \text{COROUTINES} \mapsto \rho_{COROUTINES},\ \text{EXCEPTIONS} \mapsto \rho_{EXCEPTIONS},\ \text{M2EXCEPTIONS} \mapsto \rho_{M2EXCEPTIONS},\ \text{TERMINATION} \mapsto \rho_{TERMINATION}\} \text{ in } \mu\,(\rho_{empty}, mods \mapsto sysmods)$$

System Module Proper Procedures

Abstract Syntax

types

$$\begin{aligned} &\textit{System-procedure-designator} = \textit{COROUTINES-procedure-designator} \\ &\qquad\qquad\qquad\qquad \mid \textit{EXCEPTIONS-procedure-designator} \end{aligned}$$

Static Semantics

$$\begin{aligned} &\textit{wf-system-procedure-designator} : \\ &\quad \textit{System-procedure-designator} \to \textit{Arguments} \to \textit{Environment} \to \mathbb{B} \\ &\textit{wf-system-procedure-designator}\,(\textit{desig})(\textit{aps})\rho \triangleq \\ &\quad (\textit{is-COROUTINES-procedure-designator}(\textit{desig}) \to \textit{wf-COROUTINES-procedure-designator}\,(\textit{desig})(\textit{aps})\rho, \quad {}^{370} \\ &\quad\ \textit{is-EXCEPTIONS-procedure-designator}(\textit{desig}) \to \textit{wf-EXCEPTIONS-procedure-designator}\,(\textit{desig})(\textit{aps})\rho) \quad {}^{382} \end{aligned}$$

Dynamic Semantics

$$\begin{aligned} &\textit{m-system-procedure-designator} : \\ &\quad \textit{System-procedure-designator} \to \textit{Arguments} \to \textit{Environment} \xrightarrow{o} () \\ &\textit{m-system-procedure-designator}\,(\textit{desig})(\textit{args})\rho \triangleq \\ &\quad (\ \textit{is-COROUTINES-procedure-designator}(\textit{desig}) \to \textit{m-COROUTINES-procedure-designator}(\textit{desig})\,(\textit{args})\,\rho, \quad {}^{370} \\ &\quad\ \textit{is-EXCEPTIONS-procedure-designator}(\textit{desig}) \to \textit{m-EXCEPTIONS-procedure-designator}(\textit{desig})\,(\textit{args})\,\rho) \quad {}^{382} \end{aligned}$$

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

System Module Function Procedures

Abstract Syntax

types

$$\begin{aligned} \mathit{System\text{-}function\text{-}designator} = {} & \mathit{SYSTEM\text{-}function\text{-}designator} \\ & \mid \mathit{COROUTINES\text{-}function\text{-}designator} \\ & \mid \mathit{TERMINATION\text{-}function\text{-}designator} \\ & \mid \mathit{EXCEPTIONS\text{-}function\text{-}designator} \\ & \mid \mathit{M2EXCEPTION\text{-}function\text{-}designator} \end{aligned}$$

Static Semantics

$$\mathit{wf\text{-}system\text{-}function\text{-}designator} :$$
$$\mathit{System\text{-}function\text{-}designator} \rightarrow \mathit{Arguments} \rightarrow \mathit{Environment} \rightarrow \mathbb{B}$$

$$\begin{aligned} &\mathit{wf\text{-}system\text{-}function\text{-}designator}\,(\mathit{desig})(\mathit{aps})\rho \triangleq \\ &\quad (\mathit{is\text{-}SYSTEM\text{-}function\text{-}designator}(\mathit{desig}) && \rightarrow \mathit{wf\text{-}SYSTEM\text{-}function\text{-}designator}\,(\mathit{desig})(\mathit{aps})\rho, && 354 \\ &\quad\ \mathit{is\text{-}COROUTINES\text{-}function\text{-}designator}(\mathit{desig}) && \rightarrow \mathit{wf\text{-}COROUTINES\text{-}function\text{-}designator}\,(\mathit{desig})(\mathit{aps})\rho, && 377 \\ &\quad\ \mathit{is\text{-}TERMINATION\text{-}function\text{-}designator}(\mathit{desig}) && \rightarrow \mathit{wf\text{-}TERMINATION\text{-}function\text{-}designator}\,(\mathit{desig})(\mathit{aps})\rho, && 390 \\ &\quad\ \mathit{is\text{-}EXCEPTIONS\text{-}function\text{-}designator}(\mathit{desig}) && \rightarrow \mathit{wf\text{-}EXCEPTIONS\text{-}function\text{-}designator}\,(\mathit{desig})(\mathit{aps})\rho, && 385 \\ &\quad\ \mathit{is\text{-}M2EXCEPTION\text{-}function\text{-}designator}(\mathit{desig}) && \rightarrow \mathit{wf\text{-}M2EXCEPTION\text{-}function\text{-}designator}\,(\mathit{desig})(\mathit{aps})\rho) && 394 \end{aligned}$$

$$\mathit{t\text{-}system\text{-}function\text{-}designator} :$$
$$\mathit{System\text{-}function\text{-}designator} \rightarrow \mathit{Arguments} \rightarrow \mathit{Environment} \rightarrow \mathit{Expression\text{-}typed}$$

$$\begin{aligned} &\mathit{t\text{-}system\text{-}function\text{-}designator}\,(\mathit{desig})(\mathit{aps})\rho \triangleq \\ &\quad (\mathit{is\text{-}SYSTEM\text{-}function\text{-}designator}(\mathit{desig}) && \rightarrow \mathit{t\text{-}SYSTEM\text{-}function\text{-}designator}\,(\mathit{desig})(\mathit{aps})\rho, && 354 \\ &\quad\ \mathit{is\text{-}COROUTINES\text{-}function\text{-}designator}(\mathit{desig}) && \rightarrow \mathit{t\text{-}COROUTINES\text{-}function\text{-}designator}\,(\mathit{desig})(\mathit{aps})\rho, && 377 \\ &\quad\ \mathit{is\text{-}TERMINATION\text{-}function\text{-}designator}(\mathit{desig}) && \rightarrow \mathit{t\text{-}TERMINATION\text{-}function\text{-}designator}\,(\mathit{desig})(\mathit{aps})\rho, && 390 \\ &\quad\ \mathit{is\text{-}EXCEPTIONS\text{-}function\text{-}designator}(\mathit{desig}) && \rightarrow \mathit{t\text{-}EXCEPTIONS\text{-}function\text{-}designator}\,(\mathit{desig})(\mathit{aps})\rho, && 385 \\ &\quad\ \mathit{is\text{-}M2EXCEPTION\text{-}function\text{-}designator}(\mathit{desig}) && \rightarrow \mathit{t\text{-}M2EXCEPTION\text{-}function\text{-}designator}\,(\mathit{desig})(\mathit{aps})\rho) && 394 \end{aligned}$$

Dynamic Semantics

$$\mathit{m\text{-}system\text{-}function\text{-}designator} :$$
$$\mathit{System\text{-}function\text{-}designator} \rightarrow \mathit{Arguments} \rightarrow \mathit{Environment} \xrightarrow{o} \mathit{Value}$$

$$\begin{aligned} &\mathit{m\text{-}system\text{-}function\text{-}designator}\,(\mathit{desig})(\mathit{args})\rho \triangleq \\ &\quad (\ \mathit{is\text{-}SYSTEM\text{-}function\text{-}designator}(\mathit{desig}) && \rightarrow \mathit{m\text{-}SYSTEM\text{-}function\text{-}designator}(\mathit{desig})\,(\mathit{args})\,\rho, && 354 \\ &\quad\ \mathit{is\text{-}COROUTINES\text{-}function\text{-}designator}(\mathit{desig}) && \rightarrow \mathit{m\text{-}COROUTINES\text{-}function\text{-}designator}(\mathit{desig})\,(\mathit{args})\,\rho, && 377 \\ &\quad\ \mathit{is\text{-}TERMINATION\text{-}function\text{-}designator}(\mathit{desig}) && \rightarrow \mathit{m\text{-}TERMINATION\text{-}function\text{-}designator}(\mathit{desig})\,(\mathit{args})\,\rho, && 390 \\ &\quad\ \mathit{is\text{-}EXCEPTIONS\text{-}function\text{-}designator}(\mathit{desig}) && \rightarrow \mathit{m\text{-}EXCEPTIONS\text{-}function\text{-}designator}(\mathit{desig})\,(\mathit{args})\,\rho, && 385 \\ &\quad\ \mathit{is\text{-}M2EXCEPTION\text{-}function\text{-}designator}(\mathit{desig}) && \rightarrow \mathit{m\text{-}M2EXCEPTION\text{-}function\text{-}designator}(\mathit{desig})\,(\mathit{args})\,\rho) && 394 \end{aligned}$$

7.1 The Module SYSTEM

The system module SYSTEM provides low-level facilities for gaining access to the addresses and the underlying storage of variables, for performing address arithmetic, and for manipulating the representation of values. Programs that use these facilities may be non-portable.

7.1.1 The Interface to SYSTEM

The interface to SYSTEM behaves as if the following were its definition module:

```
DEFINITION MODULE SYSTEM;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  (* Gives access to system programming facilities that are probably non portable. *)

  (* The constants and types define underlying properties of storage *)

CONST
  BITSPERLOC    = <implementation-defined constant> ;
  LOCSPERWORD   = <implementation-defined constant> ;

TYPE
  LOC; (* A system basic type. Values are the uninterpreted contents of the smallest
          addressable unit of storage *)
  ADDRESS = POINTER TO LOC;
  WORD = ARRAY [0 .. LOCSPERWORD-1] OF LOC;

  (* BYTE and LOCSPERBYTE are provided if appropriate for machine *)
CONST
  LOCSPERBYTE = <implementation-defined constant> ;

TYPE
  BYTE = ARRAY [0 .. LOCSPERBYTE-1] OF LOC;

PROCEDURE ADDADR (addr: ADDRESS; offset: CARDINAL): ADDRESS;
  (* Returns address given by (addr + offset), or may raise an exception if this
     address is not valid.
  *)

PROCEDURE SUBADR (addr: ADDRESS; offset: CARDINAL): ADDRESS;
  (* Returns address given by (addr - offset), or may raise an exception if this address
     is not valid.
  *)

PROCEDURE DIFADR (addr1, addr2: ADDRESS): INTEGER;
  (* Returns the difference between addresses (addr1 - addr2), or may raise an exception
     if the arguments are invalid or if the address space is non-contiguous.
  *)

PROCEDURE MAKEADR (val: <some type>; ... ): ADDRESS;
  (* Returns an address constructed from a list of values whose types are
     implementation-defined, or may raise an exception if this address is not valid.
  *)

PROCEDURE ADR (VAR v: <anytype>): ADDRESS;
  (* Returns the address of variable v. *)

PROCEDURE ROTATE (val: <a packedset type>; num: INTEGER): <type of first parameter>;
  (* Returns a bit sequence obtained from val by rotating up or down (left or right) by
     the absolute value of num.  The direction is down if the sign of num is negative,
     otherwise the direction is up.
  *)

PROCEDURE SHIFT (val: <a packedset type>; num: INTEGER): <type of first parameter>;
  (* Returns a bit sequence obtained from val by shifting up or down (left or right) by
     the absolute value of num, introducing zeros as necessary.  The direction is down
     if the sign of num is negative, otherwise the direction is up.
  *)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
PROCEDURE CAST (<targettype>; val: <anytype>): <targettype>;
  (* CAST is a type transfer function.  Given the expression denoted by val, it returns
     a value of the type <targettype>.  An invalid value for the target value or a
     physical address alignment problem may raise an exception.
  *)

PROCEDURE TSIZE (<type>; ... ): CARDINAL;
  (* Returns the number of LOCS used to store a value of the specified <type>.   The
     extra parameters, if present, are used to distinguish variants in a variant record.
  *)

END SYSTEM.
```

Declaration Semantics

The identifiers available for import from the module SYSTEM shall include the constant identifiers BITSPERLOC, and LOCSPERWORD, the type identifiers LOC, ADDRESS, and WORD, and the procedure identifiers ADDADR, SUBADR, DIFADR, MAKEADR, ADR, ROTATE, SHIFT, CAST, and TSIZE.

In addition, where appropriate for the target hardware architecture, the identifiers available for import from the module SYSTEM shall include the constant identifier LOCSPERBYTE, and the type identifier BYTE.

NOTE — An implementation may extend the module SYSTEM by including additional types, procedures, variables and constants appropriate to the facilities of the target hardware. The identifiers BYTE and LOCSPERBYTE are, however, reserved for use with appropriate target hardware architectures, and may not be used in SYSTEM with other meanings — see 4.4.

CHANGES

1 The type LOC has been introduced as a mechanism to resolve the problem that, in many implementations of Modula-2, both BYTE and WORD occur in SYSTEM. Its introduction allows a consistent handling of both of these types, and also enables further WORD-like types to be introduced in an implementation, for example, where TSIZE(CARDINAL)≠TSIZE(INTEGER).

2 The constants BITSPERLOC, LOCSPERWORD and LOCSPERBYTE have been introduced to denote the number of bits in a LOC, the number of LOCs in a word and the number of LOCs in a BYTE respectively.

3 Variables of type ADDRESS are no longer expression-compatible with CARDINAL, i.e. they cannot directly occur in expressions that include arithmetic operators.

4 The function MAKEADR has been introduced as a mechanism for constructing a value of type ADDRESS, and new functions ADDADR, SUBADR, and DIFADR have been introduced for address arithmetic, thereby obviating the necessity for assignment compatibility between ADDRESS and a whole number type.

5 New functions ROTATE and SHIFT have been introduced to extend the bit-level manipulation operations on packedset values.

6 The function CAST has been introduced for type transfer.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

values

$BYTE\text{-}is\text{-}implemented : \mathbb{B} =$ implementation-defined ;

$\rho_{SYSTEM} =$ let $BYTE\text{-}constants = \{$LOCSPERBYTE $\mapsto$ $locsperbyte\text{-}constant\text{-}value\}$,
$constants = \{$LOCSPERWORD $\mapsto$ $locsperword\text{-}constant\text{-}value$,
BITSPERLOC $\mapsto$ $bitsperloc\text{-}constant\text{-}value\}$,
$BYTE\text{-}types =$ if $locsperbyte > 1$
then {BYTE $\mapsto$ BYTE-TYPE}
else {BYTE $\mapsto$ LOC-TYPE},
$types = \{$WORD $\mapsto$ WORD-TYPE,
LOC $\mapsto$ LOC-TYPE,
ADDRESS $\mapsto$ ADDRESS-TYPE$\}$,
$structures = \{$BYTE-TYPE $\mapsto$ $byte\text{-}array\text{-}structure$,
BYTE-RANGE-TYPE $\mapsto$ $byte\text{-}range\text{-}structure$,
WORD-TYPE $\mapsto$ $word\text{-}array\text{-}structure$,
WORD-RANGE-TYPE $\mapsto$ $word\text{-}range\text{-}structure$,
ADDRESS-TYPE $\mapsto$ $address\text{-}type\text{-}structure\}$,
$variables = \{\mapsto\}$,
$procedures = \{$ADR $\mapsto$ SYSTEM-FUNCTION-PROCEDURE,
ADDADR $\mapsto$ SYSTEM-FUNCTION-PROCEDURE,
MAKEADR $\mapsto$ SYSTEM-FUNCTION-PROCEDURE,
CAST $\mapsto$ SYSTEM-FUNCTION-PROCEDURE,
DIFADR $\mapsto$ SYSTEM-FUNCTION-PROCEDURE,
ROTATE $\mapsto$ SYSTEM-FUNCTION-PROCEDURE,
SHIFT $\mapsto$ SYSTEM-FUNCTION-PROCEDURE,
SUBADR $\mapsto$ SYSTEM-FUNCTION-PROCEDURE,
TSIZE $\mapsto$ SYSTEM-FUNCTION-PROCEDURE$\}$,
$modules = \{\mapsto\}$,
$defmods = \{\mapsto\}$,
$level = 0$,
$local\text{-}continuations = \{\mapsto\}$,
$protection\text{-}domain =$ nil ,
$protected\text{-}procedures = \{\}$ in
if $BYTE\text{-}is\text{-}implemented$
then $mk\text{-}Qualified\text{-}environment\,(constants \cup BYTE\text{-}constants, types \cup BYTE\text{-}types,$
$structures, variables, procedures, modules, defmods, level,$
$local\text{-}continuations, protection\text{-}domain, protected)$
else $mk\text{-}Qualified\text{-}environment\,(constants, types,$
$structures, variables, procedures, modules, defmods, level,$
$local\text{-}continuations, protection\text{-}domain, protected\text{-}procedures)$

7.1.2 The Constants of SYSTEM

The module SYSTEM provides the constants BITSPERLOC and LOCSPERWORD. Additionally, if appropriate for the target hardware architecture, the constant LOCSPERBYTE is provided.

Declaration Semantics

The constant identifier BITSPERLOC shall denote the whole number literal value that is the number of bits used in the representation of values of the location type and other types occupying a single location.

The constant identifier LOCSPERWORD shall denote the whole number literal value that is the number of locations used in the representation of values of the type denoted by WORD and other types occupying a single word.

The constant identifier LOCSPERBYTE, if provided, shall denote the whole number literal value that is the number of

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

locations used in the representation of values of the type denoted by BYTE and other types occupying a single byte.

values

locsperbyte : $\mathbb{N}_1$ = implementation-defined;

locsperword : $\mathbb{N}_1$ = implementation-defined;

bitsperloc : $\mathbb{N}_1$ = implementation-defined;

locsperbyte-constant-value = *mk-Constant-value* (ℤ-TYPE, *mk-Whole-number-literal* (*locsperbyte*));

locsperword-constant-value = *mk-Constant-value* (ℤ-TYPE, *mk-Whole-number-literal* (*locsperword*));

bitsperloc-constant-value = *mk-Constant-value* (ℤ-TYPE, *mk-Whole-number-literal* (*bitsperloc*))

7.1.3 The Types of SYSTEM

The module SYSTEM provides system storage types to represent uninterpreted storage, and a type to represent logical addresses.

7.1.3.1 System Storage Types

System storage types include the location type, and types based on the location type.

Declaration Semantics

The type identifier LOC shall denote the location type. A value of this type shall correspond to the uninterpreted contents of the smallest addressable unit of storage in an implementation; the value of the call TSIZE(LOC) shall therefore be one.

NOTES

1 The only operation directly defined on the location type is assignment.

2 There are special rules affecting parameter compatibility for system storage types; where a formal parameter is expressed in terms of system storage types, the corresponding actual parameter may be of any type that is represented in a corresponding number of storage locations — see 6.9.3.3.

The type denoted by WORD shall be defined in terms of LOC as follows:

```
LOCSPERWORD = <implementation-defined constant> ;
WORD = ARRAY [0 .. LOCSPERWORD-1] OF LOC;
```

and the value of the call TSIZE(WORD) shall be LOCSPERWORD.

The type denoted by BYTE, if provided, shall be defined in terms of LOC as follows:
if TSIZE(BYTE) = 1 then

```
LOCSPERBYTE = 1;
BYTE = LOC;
```

and if TSIZE(BYTE) > 1 then

```
LOCSPERBYTE = <implementation-defined constant> ;
BYTE = ARRAY [0 .. LOCSPERBYTE-1] OF LOC;
```

and the value of the call TSIZE(BYTE) shall be LOCSPERBYTE.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

values

byte-array-structure = *mk-Array-structure* (BYTE-RANGE-TYPE, LOC-TYPE);

byte-range-structure = *mk-Subrange-structure* (UNSIGNED-TYPE, $\{0, \ldots, locsperbyte - 1\}$);

word-array-structure = *mk-Array-structure* (WORD-RANGE-TYPE, LOC-TYPE);

word-range-structure = *mk-Subrange-structure* (UNSIGNED-TYPE, $\{0, \ldots, locsperword - 1\}$)

7.1.3.2 The Address Type

Declaration Semantics

The declaration:

```
TYPE ADDRESS = POINTER TO LOC;
```

shall define the address type. Values of this type shall correspond to the logical addresses of storage locations.

NOTES

1 The term 'logical addresses' is used since it is required that the address type is assignment-compatible with all pointer types and vice-versa — see 6.4.2.

2 A formal variable parameter of the address type is parameter-compatible with an actual variable parameter of any pointer type — see 6.9.3.2.

3 As well as the operations that are normally applicable to values of pointer types, the system function procedures **ADDADR**, **SUBADR**, and **DIFADR** allow address arithmetic to be applied to all pointer types. In the case of **ADDADR** and **SUBADR** the result type is the address type; in the case of **DIFADR** the result type is the signed type.

4 Values of the address type may be used in the optional expression that follows the identifier in a variable declaration to fix the storage location of that variable; they may be constructed with the function **MAKEADR** — see 7.1.6.1.

values

address-type-structure = *mk-Pointer-structure* (LOC-TYPE)

7.1.4 The Functions of SYSTEM

The module SYSTEM provides function procedures for address arithmetic (ADDADR, DIFADR, and SUBADDR), for address value generation (MAKEADR and ADR), for packedset operations (ROTATE and SHIFT), for type transfer (CAST), and for determining the storage size of values of a given type (TSIZE).

Abstract Syntax

types

SYSTEM-function-designator :: *ident* : *Identifier*

inv *ident* $\triangleq$
ident $\in$ {ADDADR, MAKEADR, ADR, CAST, DIFADR, ROTATE, SHIFT, SUBADR, TSIZE}

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Static Semantics

wf-SYSTEM-function-designator :
SYSTEM-function-designator → *Actual-parameters* → *Environment* → *Typed*

wf-SYSTEM-function-designator (*fd*)(*aps*)ρ $\triangleq$
access-environment (*desig*)ρ = SYSTEM-FUNCTION-PROCEDURE $\wedge$ 319
cases *fd* :
ADDADR → *wf-ADDADR* (*aps*)ρ, 355
MAKEADR → *wf-MAKEADR* (*aps*)ρ, 359
ADR → *wf-ADR* (*aps*)ρ, 360
CAST → *wf-CAST* (*aps*)ρ, 364
DIFADR → *wf-DIFADR* (*aps*)ρ, 357
ROTATE → *wf-ROTATE* (*aps*)ρ, 362
SHIFT → *wf-SHIFT* (*aps*)ρ, 363
SUBADR → *wf-SUBADR* (*aps*)ρ, 356
TSIZE → *wf-TSIZE* (*aps*)ρ 365
end

t-SYSTEM-function-designator :
SYSTEM-function-designator → *Actual-parameters* → *Environment* → *Typed*

t-SYSTEM-function-designator (*fd*)(*aps*)ρ $\triangleq$
cases *fd* :
ADDADR → *t-ADDADR* (*aps*)ρ, 355
MAKEADR → *t-MAKEADR* (*aps*)ρ, 359
ADR → *t-ADR* (*aps*)ρ, 360
CAST → *t-CAST* (*aps*)ρ, 364
DIFADR → *t-DIFADR* (*aps*)ρ, 357
ROTATE → *t-ROTATE* (*aps*)ρ, 362
SHIFT → *t-SHIFT* (*aps*)ρ, 363
SUBADR → *t-SUBADR* (*aps*)ρ, 356
TSIZE → *t-TSIZE* (*aps*)ρ 366
end

Dynamic Semantics

m-SYSTEM-function-designator :
SYSTEM-function-designator → *Arguments* → *Environment* $\xrightarrow{o}$ *Value*

m-SYSTEM-function-designator (*fd*)(*args*)ρ $\triangleq$
let *handler* = *current-exception-handler* (ρ) in 318
cases *fd*:
ADDADR → *m-ADDADR*(*args*) *handler*, 355
MAKEADR → *m-MAKEADR*(*args*) *handler*, 359
ADR → *m-ADR*(*args*) ρ, 360
CAST → *m-CAST*(*args*) *handler*, 364
DIFADR → *m-DIFADR*(*args*) *handler*, 358
ROTATE → *m-ROTATE*(*args*) ρ, 362
SHIFT → *m-SHIFT*(*args*) ρ, 363
SUBADR → *m-SUBADR*(*args*) *handler*, 356
TSIZE → *m-TSIZE*(*args*) ρ 366
end

7.1.5 Address Arithmetic Functions

The function procedures ADDADR, SUBADDR and DIFADR are provided for address arithmetic operations on values of pointer types.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

7.1.5.1 The Function ADDADR

The function ADDADR can be used to construct a value of the address type by adding an offset to the value of a pointer type.

Static Semantics

A call of ADDADR shall have two actual parameters. The first parameter shall be an expression that is of a pointer type, and the second parameter shall be an expression of a type with which the unsigned type is value parameter compatible. The type of a call of ADDADR shall be the address type.

wf-ADDADR : *Actual-parameters* → *Environment* → 𝔹

wf-ADDADR(*aps*)ρ ≙
  len *aps* = 2 ∧
  let [*addr*, *offset*] = *aps* in
  is-Expression(*addr*) ∧
  let *exptype* = *t-expression* (*addr*)ρ in 190
  is-pointer-type (*exptype*)ρ ∧ 322
  is-Expression(*offset*) ∧
  is-value-parameter-compatible (UNSIGNED-TYPE, *t-expression* (*offset*)ρ)ρ 262 190

t-ADDADR : *Actual-parameters* → *Environment* → *Typed*

t-ADDADR(*aps*)ρ ≙
  ADDRESS-TYPE

Dynamic Semantics

The value of the call ADDADR(addr, offset) shall be an address calculated by adding offset to addr in an implementation-defined manner that satisfies the properties given in 7.1.5.4.

An exception shall be raised if addr is the nil value.

An exception may occur and may be raised if the ADDADR function increments an address out of address range.

NOTES

1 An exception may occur, in particular, where the address space is non-contiguous.

2 The subsequent use of a calculated address may raise an exception.

3 Adding N to a pointer which points to a type whose size is greater than one LOC will not add N*TSIZE(typePointedTo), but will add N.

m-ADDADR : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Value*

m-ADDADR(*args*)*handler* ≙
  let [*addr*, *offset*] = *args* in
  cases *addr*:
    mk-Address-value (-, -, -) → def *ok* = *can-add*(*addr*, *offset*) in 356
                                  if *ok*
                                  then *add-offset*(*addr*, *offset*) 356
                                  else *non-mandatory-detection*(ADDRESS-ARITHMETIC) *handler* , 397
    NIL-VALUE → *mandatory-detection*(ADDRESS-ARITHMETIC-ON-NIL) *handler* 396
  end

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Auxiliaries

can-add : *Address-value* × N $\xrightarrow{o}$ 𝔹

can-add (*val*, *offset*) $\triangleq$
implementation-defined — check if the result would be in address range

add-offset : *Address-value* × N $\xrightarrow{o}$ *Value*

add-offset (*val*, *offset*) $\triangleq$
implementation-defined — to satisfy the required properties for address arithmetic

7.1.5.2 The Function SUBADR

The function SUBADR can be used to construct a value of the address type by subtracting an offset from the value of a pointer type.

Static Semantics

A call of SUBADR shall have two actual parameters. The first parameter shall be an expression that is of a pointer type, and the second parameter shall be an expression of a type with which the unsigned type is value parameter compatible. The type of a call of SUBADR shall be the address type.

wf-SUBADR : *Actual-parameters* → *Environment* → 𝔹

wf-SUBADR (*aps*)ρ $\triangleq$
len *aps* = 2 ∧
let [*addr*, *offset*] = *aps* in
is-Expression(*addr*) ∧
let *exptype* = *t-expression* (*addr*)ρ in 190
is-pointer-type (*exptype*)ρ ∧ 322
is-Expression(*offset*) ∧
is-value-parameter-compatible (UNSIGNED-TYPE, *t-expression* (*offset*)ρ)ρ 262 190

t-SUBADR : *Actual-parameters* → *Environment* → *Typed*

t-SUBADR (*aps*)ρ $\triangleq$
ADDRESS-TYPE

Dynamic Semantics

The value of the call `SUBADR(addr, offset)` shall be an address calculated by subtracting `offset` from `addr` in an implementation-defined manner that satisfies the properties given in 7.1.5.4.

An exception shall be raised if `addr` is the nil value.

An exception may occur and may be raised if the SUBADR function decrements an address out of address range.

NOTES

1 An exception may occur, in particular, where the address space is non-contiguous.

2 The subsequent use of a calculated address may raise an exception.

3 Subtracting `N` from a pointer which points to a type whose size is greater than one `LOC` will not subtract `N*TSIZE(typePointedTo)`, but will subtract `N`.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

m-SUBADR : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Value*

m-SUBADR(*args*)ρ $\triangleq$
  let [*addr*, *offset*] = *args* in
   cases *addr*:
    mk-Address-value (-, -, -) → def *ok* = *can-subtract*(*addr*, *offset*) ρ in 357
                                  if *ok*
                                  then *subtract-offset*(*addr*, *offset*) ρ 357
                                  else *non-mandatory-detection*(ADDRESS-ARITHMETIC) *handler* , 397
    NIL-VALUE → *mandatory-detection*(ADDRESS-ARITHMETIC-ON-NIL) *handler* 396
  end

Auxiliaries

can-subtract : *Address-value* × **N** $\xrightarrow{o}$ **B**

can-subtract (*val*, *offset*) $\triangleq$
  implementation-defined — check if the result would be in address range

subtract-offset : *Address-value* × **N** $\xrightarrow{o}$ *Value*

subtract-offset (*val*, *offset*) $\triangleq$
  let *nval* : *Address-value* be st
        def *aval* = *add-offset*(*nval*, *offset*) in *aval* = *val* in 356
  return *nval*

7.1.5.3 The Function DIFADR

The function DIFADR can be used to calculate the offset between two addresses, by subtracting the second address from the first.

Static Semantics

A call of DIFADR shall have two actual parameters. Both parameters shall be expressions of a pointer type. The type of a call of DIFADR shall be the signed type.

NOTE — The two parameters need not be expressions of the same pointer type, since the address type is compatible with all pointer types.

wf-DIFADR : *Actual-parameters* → *Environment* → **B**

wf-DIFADR(*aps*)ρ $\triangleq$
  len *aps* = 2 ∧
  let [*addr*1, *addr*2] = *aps* in
  is-Expression(*addr*1) ∧
  let *exp*1*type* = *t-expression* (*addr*1)ρ in 190
  is-pointer-type (*exp*1*type*)ρ ∧ 322
  is-Expression(*addr*2) ∧
  let *exp*2*type* = *t-expression* (*addr*2)ρ in 190
  is-pointer-type (*exp*2*type*)ρ 322

t-DIFADR : *Actual-parameters* → *Environment* → *Typed*

t-DIFADR(*aps*)ρ $\triangleq$
  SIGNED-TYPE

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Dynamic Semantics

The value of the call `DIFADR(addr1, addr2)` shall be a signed number calculated by subtracting `addr2` from `addr1` in an implementation-defined manner that satisfies the properties given in 7.1.5.4.

An exception shall occur if `addr1` is the nil value or if `addr2` is the nil value.

An exception may occur and may be raised if the result of the `DIFADR` function is out of the range of the signed type, or if the result is meaningless because the address space is non-contiguous.

$m\text{-}DIFADR : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} Value$

$m\text{-}DIFADR\,(args)handler \triangleq$
  let $[addr1, addr2] = args$ in
  if $addr1 = \text{NIL-VALUE} \vee addr2 = \text{NIL-VALUE}$
  then $mandatory\text{-}detection(\text{ADDRESS-ARITHMETIC-ON-NIL})\ handler$ 396
  else def $ok = is\text{-}comparable\text{-}addresses(addr1, addr2)$ in 358
    if ok
    then def $offset = address\text{-}difference(addr1, addr2)$ in 358
      let $handler = current\text{-}exception\text{-}handler\ (\rho)$ in 318
      $get\text{-}whole\text{-}result(\text{SIGNED-TYPE}, offset)\ handler$ 199
    else $non\text{-}mandatory\text{-}detection(\text{ADDRESS-ARITHMETIC})\ handler$ 397

Auxiliaries

$is\text{-}comparable\text{-}addresses : Address\text{-}value \times Address\text{-}value \xrightarrow{o} \mathbb{B}$

$is\text{-}comparable\text{-}addresses\ (fval, sval)\rho \triangleq$
  return $\exists\, n \in \mathbb{N} \cdot$ def $ok = can\text{-}add(sval, n)$, 356
    $nval = add\text{-}offset(sval, n)$ in $ok \wedge nval = fval \vee$ 356
    $\exists\, n \in \mathbb{N} \cdot$ def $ok = can\text{-}add(sval, n)$, 356
    $nval = add\text{-}offset(fval, n)$ in $ok \wedge nval = sval$ 356

$address\text{-}difference : Address\text{-}value \times Address\text{-}value \xrightarrow{o} \mathbb{Z}$

$address\text{-}difference\ (fval, sval) \triangleq$
  if $\exists\, n \in \mathbb{N} \cdot$ def $nval = add\text{-}offset(sval, n)$ in $nval = fval$ 356
  then let $offset : \mathbb{N}$ be st
    def $nval = add\text{-}offset(sval, offset)$ in $nval = fval$ in 356
    return $offset$
  else let $offset : \mathbb{N}$ be st
    def $nval = add\text{-}offset(fval, offset)$ in $nval = sval$ in 356
    return $-\,offset$

7.1.5.4 Properties of the address arithmetic functions

Given the declarations `VAR adr: ADDRESS; n: CARDINAL;` the following shall be true of the address arithmetic function procedures if an exception is not raised:

```
SUBADR(ADDADR(adr,n), n)   = adr
ADDADR(SUBADR(adr,n), n)   = adr
DIFADR(ADDADR(adr,n), adr) = n
DIFADR(adr, SUBADR(adr,n)) = n
ADDADR(adr, 0)             = adr
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

7.1.6 Address Construction and Enquiry Functions

The function procedures `MAKEADR` and `ADR` are provided for the construction and determination of address values.

7.1.6.1 The Function MAKEADR

The function `MAKEADR` can be used to construct a value of the address type from one or more values of implementation-defined types. The actual number and types of the parameters are not defined in this part of ISO/IEC 10514 in order that implementations may offer one or more alternatives appropriate to the target architecture. On some implementations, one of the parameters might be used to distinguish between the generation of logical addresses and the generation of hardware addresses, such as apply to memory-mapped I/O.

A value constructed by `MAKEADR` from constant parameters can be used in the optional expression that follows the identifier in a variable declaration to fix the address of storage used for that variable — see 6.2.10.

Static Semantics

A call of `MAKEADR` shall have an implementation-defined number of actual parameters of implementation-defined types. The type of a call of `MAKEADR` shall be the address type.

wf-MAKEADR : *Actual-parameters* → *Environment* → $\mathbb{B}$

wf-MAKEADR(*aps*)ρ $\triangleq$
  implementation-defined

annotations Check if the call is well-formed, as defined for the target architecture.

t-MAKEADR : *Actual-parameters* → *Environment* → *Typed*

t-MAKEADR(*aps*)ρ $\triangleq$
  ADDRESS-TYPE

Dynamic Semantics

The value of the call `MAKEADR(val1, ... valN)` shall be an implementation-defined conversion of `val1, ... valN`. An exception may occur and may be raised if the result is not in address range.

NOTE — The subsequent use of an address constructed in this way may raise an exception.

m-MAKEADR : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Value*

m-MAKEADR(*args*)*handler* $\triangleq$
  def *ok* = *can-make*(*args*) in 359
  if *ok*
  then implementation-defined — conversion of arguments to an address value
  else *non-mandatory-detection*(ADDRESS-CONSTRUCTION) *handler* 397

Auxiliaries

can-make : *Arguments* $\xrightarrow{o}$ $\mathbb{B}$

can-make(*args*) $\triangleq$
  implementation-defined — check if the result of conversion would be in address range

7.1.6.2 The Function ADR

The function `ADR` can be used to obtain the address of a variable.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Static Semantics

A call of ADR shall have one actual parameter that shall be a variable designator. The type of a call of ADR shall be the address type.

wf-ADR : *Actual-parameters* → *Environment* → $\mathbb{B}$
wf-ADR (*aps*)ρ $\triangleq$
  len *aps* = 1 ∧
  let [*var*] = *aps* in
  is-Typed-variable-designator(*var*) ∧
  let *mk-Typed-variable-designator* (*desig*, *type*) = *var* in
  type = *t-variable-designator* (*desig*)ρ 182

t-ADR : *Actual-parameters* → *Environment* → *Typed*
t-ADR (*aps*)ρ $\triangleq$
  ADDRESS-TYPE

Dynamic Semantics

The value of the call ADR(v) shall be the address of the variable v such that if

```
TYPE
  Val    = ... ;
  PtoVal = POINTER TO Val;
VAR
  v : Val;
  p : PtoVal;
...
p := ADR(v)
```

then p^ and v shall designate the same variable.

m-ADR : *Arguments* → *Environment* $\xrightarrow{o}$ *Value*
m-ADR (*args*) $\triangleq$
  let [*mk-Typed-variable* (*var*, *type*)] = *args* in
  address-of (*var*, *type*) ρ 360

Auxiliaries

address-of : *Variable* × *Variable-typed* → *Environment* $\xrightarrow{o}$ *Value*
address-of (*arg*, *type*)ρ $\triangleq$
  let *ssize* = *tsize* (*type*, [])ρ in 366
  let *handler* = *current-exception-handler* (ρ) in 318
  let *size* = *get-whole-result*($\mathbb{Z}$-TYPE, *ssize*) *handler* in 199
  let *atype* = *mk-Ptype-constraint* (*type*, ρ) in
  return *mk-Address-value* (*arg*, *atype*, *size*)

7.1.7 Packedset Functions

The functions ROTATE and SHIFT augment the facilities provided by packedset infix operations and packedset relational operations for the bit-level manipulation of values of packedset types.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

7.1.7.1 Packedset mapping

Values of a packedset type shall be considered as sequences of bits, numbered according to the ordinal values of the members of the the host type of the base type of the packedset. Bit number n is set (i.e. has the value 1) if n is an element of the packedset, and bit number n is unset or clear (has the value 0) if n is not an element of the packedset.

The bit-numbering shall be such that bit number 0 is the least significant bit in a machine register. Bit number 1 is the next least significant bit *etc.* Adjacent bit numbers shall be adjacent bits in the machine register.

The ordinal values of the lower and upper bounds of the interval used to define a packedset define a bit field within which bit-level operations are to be applied. An implementation may restrict the lower and upper bounds of the interval used to define a packedset — see 4.11.

In shift and rotate operations on bit sequences, a move 'up' by a number of bit positions will move the bit values within the bit field in the direction of increasing bit numbers; this is also described as a 'left' move since bit positions of increasing significance are often shown as being laid out from right to left. Similarly, a move 'down', or 'right', will move the bit values within the bit field in the direction of decreasing bit numbers. In the case of shift operations, 0 bit values replace those moved away from a boundary of a bit field; in the case of rotate operations, bit values moved out of the bit field at one boundary are moved into the bit field at the other boundary.

NOTES

1 For example, the lower bound may be restricted to 0 and the upper bound to a value that is a multiple of `BITSPERLOC`.

2 The restriction on the upper and lower bounds of the defining interval might apply to packedsets passed as parameters to `SHIFT` and `ROTATE` if such manipulation of only part of a machine word might be impossible.

3 Set operations applied to packedsets are equivalent to the following bit-level operations within bit fields:

`+`	union	bitwise `inclusive-or` operation on two bit sequences
`-`	set difference	the bits of the first sequence that are not in the second
`*`	intersection	bitwise `and` operation on two bit sequences
`/`	symmetric set difference	bitwise `exclusive-or` operation on two bit sequences
`INCL(s,x)`		set bit `x` (to 1)
`EXCL(s,x)`		clear bit `x` (to 0)
`=`	equality	are the two bit sequences equal?
`<>`	inequality	are the two bit sequences different?
`<=`	subset	is the first bit sequence included in the second?
`>=`	superset	does the first bit sequence include the second?
`x IN s`	membership	is bit number `x` set to 1?

4 It is intended that packed sets be mapped on to machine register bits in such a way that in an implementation having a binary representation for `CARDINAL` and where, in addition, `TSIZE(CARDINAL)=TSIZE(BITSET)`, then the following would be true.

```
CAST(CARDINAL,BITSET{0}) = VAL(CARDINAL,1)
```

and if `v` is a variable of type `CARDINAL`, then

```
CAST(CARDINAL,SHIFT(CAST(BITSET,v),-1)) = v DIV 2
```

and provided `v < = MAX(CARDINAL) DIV 2`, then

```
CAST(CARDINAL,SHIFT(CAST(BITSET,v),1)) = v * 2
```

7.1.7.2 The Function ROTATE

The function `ROTATE` can be used to rotate a bit sequence up or down (left or right) by a specified number of positions within a bit field. The bit sequence is defined by the value of a packedset type, and the direction of rotation is down (right) if the specified number is negative; otherwise the direction is up (left).

Static Semantics

A call of `ROTATE` shall have two actual parameters. The first parameter shall be an expression that is of a packedset type. The second parameter shall be an expression of a type with which the signed type is value parameter compatible.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The type of a call of ROTATE shall be the type of the first parameter.

An implementation may impose restrictions on the types that are allowed as the first parameter, save that ROTATE will at least accept a value of the type BITSET — see 4.11 and 6.10.2.9.

wf-ROTATE : *Actual-parameters* → *Environment* → $\mathbb{B}$

wf-ROTATE (*aps*)ρ $\triangleq$
  len *aps* = 2 ∧
  let [*tpset*, *rotnum*] = *aps* in
  is-Typed-expression(*tpset*) ∧ let *mk-Typed-expression* (*pset*, *type*) = *tpset* in
  is-Expression(*pset*) ∧ *type* = *t-expression* (*pset*)ρ ∧ 190
  is-Packedset-type(*type*)ρ ∧
  is-Expression(*rotnum*) ∧
  is-value-parameter-compatible (SIGNED-TYPE, *t-expression* (*rotnum*)ρ)ρ 262 190

t-ROTATE : *Actual-parameters* → *Environment* → *Typed*

t-ROTATE (*aps*)ρ $\triangleq$
  let [*tpset*, -] = *aps* in
  let *mk-Typed-expression* (-, *type*) = *tpset* in
  host-type-of (*type*)ρ 325

Dynamic Semantics

The value of the call ROTATE(val, num) shall be a packedset whose elements are such that the offset of their ordinal values from the ordinal value of the minimum value of the base type of the packedset, is given by the offset of the ordinal values of the elements of val increased by num if it is positive, or decreased by the absolute value of num if it is negative, and scaled by the number of values in the base type.

NOTE — With the interpretation of a value of a packedset type as a sequence of bits, the rotate operation can be considered as a rotation of the bits upwards (leftwards), or a rotation of the bits downwards (rightwards), according to the value of the second operand, and within the bit field defined by the base type of the packedset.

m-ROTATE : *Arguments* → *Environment* $\xrightarrow{o}$ *Value*

m-ROTATE (*args*)ρ $\triangleq$
  let [*tpsetval*, *num-to-rot*] = *args* in
  let *mk-Typed-value* (*psetval*, *type*) = *tpsetval* in
  let *btype* = *base-type-of* (*type*)ρ in 325
  let *min* = *scalar-value* (*minimum* (*btype*)ρ) in 210 329
  let *max* = *scalar-value* (*maximum* (*btype*)ρ) in 210 328
  let *result* be st {*scalar-value* (*x*) | *x* ∈ *result*} = *rotate* (*min*, *max*, *psetval.value*, *num-to-rot*) ∧ 210 362
    ∀ *x* ∈ *result* · *x* ∈ *values-of* (*btype*) in 329
  return *mk-Set-value* (*result*)

Auxiliaries

rotate : $\mathbb{N} \times \mathbb{N} \times$ *Ordinal-value*-set $\times \mathbb{Z} \rightarrow$ $\mathbb{N}$-set

rotate (*min*, *max*, *val*, *nrot*) $\triangleq$
  let *d* = *max* − *min* + 1 in
  {((*scalar-value* (*x*) + *nrot* − *min*) mod *d*) + *min* | *x* ∈ *val*} 210

7.1.7.3 The Function SHIFT

The function SHIFT can be used to shift a bit sequence up or down (left or right) by a specified number of positions within a bit field. The bit sequence is defined by the value of a packedset type, and the direction of shift is down (right) if the specified number is negative; otherwise the direction is up (left).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Static Semantics

A call of SHIFT shall have two actual parameters. The first parameter shall be an expression that is of a packedset type. The second parameter shall be an expression of a type with which the signed type is value parameter compatible.

The type of a call of SHIFT shall be the type of the first parameter.

An implementation may impose restrictions on the types that are allowed as the first parameter, save that SHIFT will at least accept a value of the type BITSET — see 4.11 and 6.10.2.9.

wf-SHIFT : *Actual-parameters* → *Environment* → $\mathbb{B}$

wf-SHIFT (*aps*)ρ $\triangleq$
  len *aps* = 2 ∧
  let [*tpset*, *shiftnum*] = *aps* in
  is-Typed-expression(*tpset*) ∧ let *mk-Typed-expression* (*pset*, *type*) = *tpset* in
  is-Expression(*pset*) ∧ *type* = *t-expression* (*pset*)ρ ∧ 190
  is-Packedset-type(*type*)ρ ∧
  is-Expression(*shiftnum*) ∧
  is-value-parameter-compatible (SIGNED-TYPE, *t-expression* (*shiftnum*)ρ)ρ 262 190

t-SHIFT : *Actual-parameters* → *Environment* → *Typed*

t-SHIFT (*aps*)ρ $\triangleq$
  let [*tpset*, -] = *aps* in
  let *mk-Typed-expression* (-, *type*) = *tpset* in
  host-type-of (*type*)ρ 325

Dynamic Semantics

The value of the call SHIFT(val, num) shall be a packedset whose elements are such that the offset of their ordinal values from the ordinal value of the minimum value of the base type of the packedset, is given by the offset of the ordinal values of the elements of val increased by num if it is positive, or decreased by the absolute value of num if it is negative, provided that offset is not negative and is less than the number of values of the base type.

NOTE — With the interpretation of a value of type packedset as a sequence of bits, the shift operation can be considered as a zero-propagating logical-shift-up (left) operation or a logical-shift-down (right) operation, according to the value of the second operand, and within the bit field defined by the base type of the packedset.

m-SHIFT : *Arguments* → *Environment* $\xrightarrow{o}$ *Value*

m-SHIFT (*args*)ρ $\triangleq$
  let [*tpsetval*, *num-to-shift*] = *args* in
  let *mk-Typed-value* (*psetval*, *type*) = *tpsetval* in
  let *btype* = *base-type-of* (*type*)ρ in 325
  let *min* = *scalar-value* (*minimum* (*btype*)ρ) in 210 329
  let *max* = *scalar-value* (*maximum* (*btype*)ρ) in 210 328
  let *result* be st {*scalar-value* (*x*) | *x* ∈ *result*} = *shift* (*min*, *max*, *psetval.value*, *num-to-shift*) ∧ 210 363
    ∀ *x* ∈ *result* · *x* ∈ *values-of* (*btype*) in 329
  return *mk-Set-value* (*result*)

Auxiliaries

shift : $\mathbb{N} \times \mathbb{N} \times$ *Ordinal-value*-set $\times \mathbb{Z} \to \mathbb{N}$-set

shift (*min*, *max*, *val*, *nshift*) $\triangleq$
  {*scalar-value* (*x*) + *nshift* | 210
    x ∈ *val* ·
      min ≤ *scalar-value* (*x*) + *nshift* ∧ *scalar-value* (*x*) + *nshift* ≤ *max*} 210 210

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

7.1.8 The Function CAST

The function CAST can be used (as a type transfer function) to interpret a value of any type other than a numeric literal value as a value of another type.

CHANGE — In *Programming in Modula-2*, type transfer is effected by using a type identifier in the role of a function name, as exemplified by `CARDINAL(IntExpr)`.

Static Semantics

A call of CAST shall have two actual parameters. The first parameter shall be a qualified identifier that denotes a type. The second shall be an expression whose type is not that of a numeric literal.

NOTE 1 — The type of the expression that is the second parameter cannot be a numeric literal type, since such types do not have a defined storage size.

wf-CAST : *Actual-parameters* → *Environment* → $\mathbb{B}$

wf-CAST (*aps*)ρ $\triangleq$
  len *aps* = 2 ∧
  let [*qid*, *texpr*] = *aps* in
  is-type (*qid*)ρ ∧ 314
  is-Typed-expression(*texpr*) ∧
  let *mk-Typed-expression* (*expr*, *stype*) = *texpr* in
  is-Expression(*expr*) ∧
  t-expression (*expr*)ρ = *stype* ∧ 190
  t-expression (*expr*)ρ $\notin$ {$\mathbb{Z}$-TYPE, $\mathbb{R}$-TYPE, $\mathbb{C}$-TYPE} 190

t-CAST : *Actual-parameters* → *Environment* → *Typed*

t-CAST (*aps*)ρ $\triangleq$
  let [*qid*, -] = *aps* in
  let *ttype* = *type-of* (*qid*)ρ in 314
  host-type-of (*ttype*)ρ 325

Dynamic Semantics

The value of the call `CAST(Type, val)` shall be given by a possibly unchecked conversion of `val` to the type `Type` in an implementation-defined manner satisfying the properties given below.

If the number of storage units that `val` occupies is the same as the number of storage units used by a variable of the type `Type`, the bit pattern representation of the result shall be the same as the bit pattern representation of `val`; otherwise the result and the value of `val` shall have the same bit pattern representation for a size that is equal to the smaller of the number of storage units.

An exception may occur and may be raised if a call of CAST causes an implementation-dependent alignment problem to occur, or if the resulting bit pattern is not a valid representation for a value of the target type.

m-CAST : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Value*

m-CAST (*args*)*handler* $\triangleq$
  let [*ttype*, *val*] = *args* in
  return *cast-value*(*ttype*, *val*) 365

properties

Let `ARRAY [0 .. t] OF LOC` correspond to `Type` where `t = TSIZE(Type) - 1`

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Let `ARRAY [0 .. v] OF LOC` correspond to `val` where `v = TSIZE(`*type of* `SourceExpression) - 1`

The value of the call `CAST(Type, val)` shall be a value of type `Type` that when interpreted as an `ARRAY [0 .. t] OF LOC` has, for its first `min` elements, values that are the same as the first `min` elements of the value of `val` when interpreted as an `ARRAY [0 .. v] OF LOC`, where `min` is the minimum of `t` and `v`.

NOTE 2 — If `t` > `v`, the values of remaining elements of the array from `v+1` to `t` are not defined.

If a call of `CAST` causes an implementation-dependent alignment problem to occur, the following may result:
non-mandatory-detection(INVALID-ALIGNMENT) *handler* 365 397

If the resulting bit pattern is not a valid representation for a value of the target type, the following may result:
non-mandatory-detection(INVALID-VALUE) *handler* 365 397 397

Auxiliaries

$cast\text{-}value : Typed \times Value \xrightarrow{o} Value$

$cast\text{-}value\,(type, val) \triangleq$
implementation-defined — to satisfy the required properties for `CAST`.

7.1.9 The Function TSIZE

The function `TSIZE` can be used to ascertain the amount of storage that is used for a variable of a particular type. There are two versions of this function. The first has a type identifier as a parameter. It returns the size occupied by values of the type in terms of the smallest addressable unit of storage. This version is closely equivalent to the standard function `SIZE` — see 6.10.4.17. The second version has a record type identifier as the first parameter and one or more constant expressions as the remaining parameters; the values of these constant expressions must be associated with tags of variant components of that record type. It returns the size of the variant record in terms of the smallest addressable unit of storage that would be needed to store a value of the record type, with the tag fields having the values supplied in the call.

Static Semantics

The first, and possibly only parameter, shall be a qualified identifier denoting a type.

If the second and subsequent parameters are present, they shall be constant expressions, and the first parameter shall denote a record type with variant fields. The types of the constant expressions shall correspond to the types of the tag fields contained in the record type.

NOTE — Values may not be given for the tag fields of any record type embedded within the record type specified as the first parameter.

The type of a call of `TSIZE` shall be the $\mathbb{Z}$-type.

$wf\text{-}TSIZE : Actual\text{-}parameters \to Environment \to \mathbb{B}$

$wf\text{-}TSIZE\,(aps)\rho \triangleq$
len $aps \geq 1 \wedge$
let $[qid] \curvearrowright exprs = aps$ in
$is\text{-}type\,(qid)\rho \wedge$ 314
$(exprs = [] \vee$
(let $rs = structure\text{-}of\,(qid)\rho$ in 315
$is\text{-}Record\text{-}structure(rs) \wedge$
len $exprs \geq 1 \wedge$
let $mk\text{-}Record\text{-}structure\,(fields) = rs$ in
$is\text{-}tags\text{-}in\text{-}field\text{-}list\,(fields, exprs)\rho \wedge$ 366
$\forall\, expr \in exprs \cdot is\text{-}constant\text{-}expression\,(expr)\rho))$ 254

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

t-TSIZE : *Actual-parameters* → *Environment* → *Typed*

t-TSIZE (*aps*)ρ $\triangleq$
  $\mathbb{Z}$-TYPE

Dynamic Semantics

The value of the call `TSIZE(Type)` shall be the size occupied by values of the type `Type` in terms of the smallest addressable unit of storage.

The value of the call `TSIZE(Type, val ... )` shall be the size of the variant record in terms of the smallest addressable unit of storage that would be needed to store values of the record with the tag fields having the values `val ...`.

m-TSIZE : *Arguments* → *Environment* $\xrightarrow{o}$ *Value*

m-TSIZE (*args*)ρ $\triangleq$
  let [*type*] $\curvearrowright$ *vals* = *args* in
  let *val* = *tsize* (*type*, *vals*)ρ in 366
  let *handler* = *current-exception-handler* (ρ) in 318
  get-whole-result($\mathbb{Z}$-TYPE, *val*) *handler* 199

tsize : *Variable-typed* × *Value** → *Environment* → $\mathbb{N}$

tsize (*type*, *vals*)ρ $\triangleq$
  cases *type* :
    LOC-TYPE → 1,
    BYTE-TYPE → *locsperbyte*,
    WORD-TYPE → *locsperword*,
    others → implementation-defined
  end

Auxiliaries

is-tags-in-field-list : *Field-list-structure* × *Expression** → *Environment* → $\mathbb{B}$

is-tags-in-field-list (*fsl*, *exprs*)ρ $\triangleq$
  if *exprs* = [ ]
  then true
  elseif *fsl* = [ ]
  then *exprs* = [ ]
  else *is-tags-in-fields* (hd *fsl*, *exprs*)ρ ∧ 366
    let *size* = *number-of-tags-in-fields* (hd *fsl*, *exprs*)ρ in 367
    let *rexprs* = *rest* (*exprs*, *size* + 1) in 614
    is-tags-in-field-list (tl *fsl*, *rexprs*)ρ 366

annotations Check that the tag fields contained in a fields-list match the values given by an expression sequence.

is-tags-in-fields : *Fields-structure-structure* × *Expression** → *Environment* → $\mathbb{B}$

is-tags-in-fields (*fs*, *exprs*)ρ $\triangleq$
  (*is-Fixed-fields-structure*(*fs*) → true,
   is-Variant-fields-structure(*fs*) → *is-tags-in-variant-fields* (*fs*, *exprs*)ρ) 366

annotations Check that the tag fields contained in a fields match the values given by an expression sequence.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

is-tags-in-variant-fields : *Variant-fields-structure* × *Expression** → *Environment* → $\mathbb{B}$

is-tags-in-variant-fields (*vfs*, *exprs*)ρ ≙
  let *vfs* = *mk-Variant-fields-structure*(-, *tagt*, *variants*, *other*) in
  t-expression (hd *exprs*)ρ = *host-type-of* (*tagt*)ρ ∧ 190 325
  let *tagv* = *evaluate-constant-expression* (hd *exprs*)ρ in 257
  if ∃ *i* ∈ inds *variants* · *tagv* ∈ *variants*(*i*).*labels*.*value*
  then let *i* ∈ inds *variants* be st *tagv* ∈ *variants*(*i*).*labels*.*value* in
    is-tags-in-variant (*variants*(*i*), tl *exprs*)ρ 367
  elseif *other* ≠ NIL
  then *is-tags-in-field-list* (*other*, tl *exprs*)ρ 366
  else false

annotations Check that the type of the expression in the sequence of tag values is assignment-compatible with the type of the corresponding tag component and the expression that corresponds to any tag field is a constant expression.

is-tags-in-variant : *Variant-structure* × *Expression** → *Environment* → $\mathbb{B}$

is-tags-in-variant (*variant*, *exprs*) ≙
  let *mk-Variant-structure* (-, *fields*) = *variant* in
  is-tags-in-field-list (*fields*, *exprs*)ρ 366

annotations Check the tag values in a sequence of values correspond to any tag fields in a variant component.

number-of-tags-in-field-list : *Fields-list-structure* × *Expression** → *Environment* → $\mathbb{N}$

number-of-tags-in-field-list (*fsl*, *exprs*)ρ ≙
  if *fsl* = [] ∨ *exprs* = []
  then 0
  else let *size* = *number-of-tags-in-fields* (hd *fsl*, *exprs*)ρ in 367
    let *rexprs* = *rest* (*exprs*, *size* + 1) in 614
    size + *number-of-tags-in-field-list* (tl *fsl*, *rexprs*)ρ 367

annotations Calculate the number of tags in a fields-list.

number-of-tags-in-fields : *Fields-structure* × *Expression** → *Environment* → $\mathbb{N}$

number-of-tags-in-fields (*fs*, *exprs*)ρ ≙
  (*is-Fixed-fields-structure*(*fs*) → 0,
   is-Variant-fields-structure(*fs*) → *number-of-tags-in-variant-fields* (*fs*, *exprs*)ρ) 367

annotations Calculate the number of tags in a fields.

number-of-tags-in-variant-fields : *Variant-fields-structure* × *Expression** → *Environment* → $\mathbb{N}$

number-of-tags-in-variant-fields (*vstruc*, *exprs*)ρ ≙
  let *mk-Variant-fields-structure* (-, -, *variants*, *other*) = *vstruc* in
  let *tagval* = *evaluate-constant-expression* (hd *exprs*)ρ in 257
  if ∃ *i* ∈ inds *variants* · *tagval* ∈ *variants*(*i*).*labels*.*value*
  then let *i* ∈ inds *variants* be st *tagval* ∈ *variants*(*i*).*labels*.*value* in
    number-of-tags-in-variant (*variants*(*i*), tl *exprs*)ρ + 1 367
  elseif *other* ≠ nil
  then *number-of-tags-in-field-list* (*other*, tl *exprs*)ρ + 1 367
  else 0

annotations Calculate the number of tags in a variant component of a record structure, selected by the value of the tag field.

number-of-tags-in-variant : *Variant-structure* × *Expression** → *Environment* → $\mathbb{N}$

number-of-tags-in-variant (*vstruc*, *exprs*)ρ ≙
  let *mk-Variant-structure*(-, *fields*) = *vstruc* in
  number-of-tags-in-field-list (*fields*, *exprs*)ρ 367

annotations Calculates the number of tags in a variant component of a record structure.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

7.2 The Module COROUTINES

The system module COROUTINES provides facilities for creating coroutines, for the explicit transfer of control between coroutines, and for the handling of interrupts by coroutine transfer. Programs that use the interrupt handling facilities may be non-portable since the type used to identify sources of interrupts is implementation-defined.

7.2.1 The Interface to COROUTINES

The interface to COROUTINES behaves as if the following were its definition module.

```
DEFINITION MODULE COROUTINES;

(* Facilities for coroutines and the handling of interrupts *)

IMPORT SYSTEM;

TYPE
  COROUTINE; (* Values of this type are created dynamically by NEWCOROUTINE
                and identify the coroutine in subsequent operations *)
  INTERRUPTSOURCE = <implementation-defined>;

PROCEDURE NEWCOROUTINE (procBody: PROC; workspace: SYSTEM.ADDRESS; size: CARDINAL;
                        VAR cr: COROUTINE[; initProtection: PROTECTION]);
  (* Creates a new coroutine whose body is given by procBody, and returns the identity
     of the coroutine in cr.  workspace is a pointer to the work space allocated to
     the coroutine; size specifies the size of this workspace in terms of SYSTEM.LOC.
     initProtection is an optional parameter that specifies the initial protection
     level of the coroutine.
  *)

PROCEDURE TRANSFER (VAR from: COROUTINE; to: COROUTINE);
  (* Returns the identity of the calling coroutine in from, and transfers control to
     the coroutine specified by to.
  *)

PROCEDURE IOTRANSFER (VAR from: COROUTINE; to: COROUTINE);
  (* Returns the identity of the calling coroutine in from and transfers control to
     the coroutine specified by to.  On occurrence of an interrupt, associated with the
     caller, control is transferred back to the caller, and the identity of the
     interrupted coroutine is returned in from.  The calling coroutine must be
     associated with a source of interrupts.
  *)

PROCEDURE ATTACH (source: INTERRUPTSOURCE);
  (* Associates the specified source of interrupts with the calling coroutine. *)

PROCEDURE DETACH (source: INTERRUPTSOURCE);
  (* Dissociates the specified source of interrupts from the calling coroutine. *)

PROCEDURE IsATTACHED (source: INTERRUPTSOURCE): BOOLEAN;
  (* Returns TRUE if and only if the specified source of interrupts is currently
     associated with a coroutine; otherwise returns FALSE.
  *)

PROCEDURE HANDLER (source: INTERRUPTSOURCE): COROUTINE;
  (* Returns the coroutine, if any, that is associated with the source of interrupts.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
    The result is undefined if IsATTACHED(source) = FALSE.
  *)

PROCEDURE CURRENT (): COROUTINE;
  (* Returns the identity of the calling coroutine. *)

PROCEDURE LISTEN (p: PROTECTION);
  (* Momentarily changes the protection of the calling coroutine to p. *)

PROCEDURE PROT (): PROTECTION;
  (* Returns the protection of the calling coroutine. *)

END COROUTINES.
```

Declaration Semantics

The identifiers available for import from the module COROUTINES shall be the type identifiers COROUTINE and INTERRUPTSOURCE, and the procedure identifiers NEWCOROUTINE, TRANSFER, IOTRANSFER, ATTACH, DETACH, IsATTACHED, HANDLER, CURRENT, LISTEN, and PROT.

values

$$
\begin{aligned}
\rho_{COROUTINES} = \text{let } & \mathit{constants} = \{\mapsto\},\\
& \mathit{types} = \{\texttt{COROUTINE} \mapsto \text{COROUTINE-TYPE},\\
& \qquad \texttt{INTERRUPTSOURCE} \mapsto \text{INTERRUPTSOURCE-TYPE}\},\\
& \mathit{structures} = \{\mapsto\},\\
& \mathit{variables} = \{\mapsto\},\\
& \mathit{procedures} = \{\texttt{ATTACH} \mapsto \text{COROUTINE-PROPER-PROCEDURE},\\
& \qquad \texttt{DETACH} \mapsto \text{COROUTINE-PROPER-PROCEDURE},\\
& \qquad \texttt{IOTRANSFER} \mapsto \text{COROUTINE-PROPER-PROCEDURE},\\
& \qquad \texttt{LISTEN} \mapsto \text{COROUTINE-PROPER-PROCEDURE},\\
& \qquad \texttt{NEWCOROUTINE} \mapsto \text{COROUTINE-PROPER-PROCEDURE},\\
& \qquad \texttt{TRANSFER} \mapsto \text{COROUTINE-PROPER-PROCEDURE},\\
& \qquad \texttt{HANDLER} \mapsto \text{COROUTINE-FUNCTION-PROCEDURE},\\
& \qquad \texttt{IsATTACHED} \mapsto \text{COROUTINE-FUNCTION-PROCEDURE},\\
& \qquad \texttt{CURRENT} \mapsto \text{COROUTINE-FUNCTION-PROCEDURE},\\
& \qquad \texttt{PROT} \mapsto \text{COROUTINE-FUNCTION-PROCEDURE}\},\\
& \mathit{modules} = \{\mapsto\},\\
& \mathit{defmods} = \{\mapsto\},\\
& \mathit{level} = 0,\\
& \mathit{local\text{-}continuations} = \{\mapsto\},\\
& \mathit{protection\text{-}domain} = \text{nil},\\
& \mathit{protected\text{-}procedures} = \{\ \} \text{ in}\\
& \mathit{mk\text{-}Qualified\text{-}environment}(\mathit{constants}, \mathit{types}, \mathit{structures}, \mathit{variables}, \mathit{procedures}, \mathit{modules}, \mathit{defmods},\\
& \qquad \mathit{level}, \mathit{local\text{-}continuations}, \mathit{protection\text{-}domain}, \mathit{protected\text{-}procedures})
\end{aligned}
$$

7.2.2 The Types of COROUTINES

The module COROUTINES provides the coroutine type (denoted by COROUTINE) in terms of which coroutines are identified, and the implementation-defined interrupt source type (denoted by INTERRUPTSOURCE) in terms of which sources of interrupts are identified.

CHANGES

1 The type PROCESS is called COROUTINE in this part of ISO/IEC 10514. The change from PROCESS to ADDRESS which occurred in the third edition of *Programming in Modula-2* has not been adopted.

2 A particular coroutine is identified by the same value of the coroutine type throughout the lifetime of that coroutine.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

3 The interrupt source type denoted by INTERRUPTSOURCE is not described in *Programming in Modula-2*.

Declaration Semantics

The type identifier COROUTINE shall denote the coroutine type, and the type identifier INTERRUPTSOURCE shall denote the implementation-defined interrupt source type.

The coroutine type shall be a system basic type, distinct from any other type, whose values identify a coroutine throughout the lifetime of that coroutine. The interrupt source type may be identical to a basic type or may be a predefined constructed type.

NOTE — In addition to use with the COROUTINES procedures NEWCOROUTINE, TRANSFER, IOTRANSFER, and CURRENT, as a system basic type, the operations of assignment and equality testing are applicable to the coroutine type — see 6.8.2.7.7.

annotations Values of the coroutine type are represented by values of the type *Coroutine* in the formal model — see 6.12.1. The INTERRUPTSOURCE-TYPE is an implementation-defined type — see 6.12.7. Values of the interrupt source type are represented by values of the type *Interrupt-source* in the formal model — see 6.12.7.

7.2.3 The Procedures of COROUTINES

The module COROUTINES provides procedures for creating coroutines (NEWCOROUTINE), transferring between coroutines (TRANSFER and IOTRANSFER), for associating and dissociating interrupt sources and coroutines (ATTACH and DETACH), and for unmasking interrupts momentarily (LISTEN).

Abstract Syntax

types

COROUTINES-procedure-designator :: *ident* : *Identifier*

inv *ident* $\triangleq$
ident $\in$ {ATTACH, DETACH, IOTRANSFER, LISTEN, NEWCOROUTINE, TRANSFER}

Static Semantics

wf-COROUTINES-procedure-designator :
COROUTINES-procedure-designator $\rightarrow$ *Actual-parameters* $\rightarrow$ *Environment* $\rightarrow \mathbb{B}$

wf-COROUTINES-procedure-designator (*pd*)(*aps*)ρ $\triangleq$
access-environment (*desig*)ρ = COROUTINE-PROPER-PROCEDURE $\wedge$ 319
cases *pd* :
ATTACH $\rightarrow$ *wf-ATTACH* (*aps*)ρ, 371
DETACH $\rightarrow$ *wf-DETACH* (*aps*)ρ, 371
IOTRANSFER $\rightarrow$ *wf-IOTRANSFER* (*aps*)ρ, 372
LISTEN $\rightarrow$ *wf-LISTEN* (*aps*)ρ, 373
NEWCOROUTINE $\rightarrow$ *wf-NEWCOROUTINE* (*aps*)ρ, 374
TRANSFER $\rightarrow$ *wf-TRANSFER* (*aps*)ρ 376
end

Dynamic Semantics

m-COROUTINES-procedure-designator :
COROUTINES-procedure-designator $\rightarrow$ *Arguments* $\rightarrow$ *Environment* $\xrightarrow{o}$ ()

m-COROUTINES-procedure-designator (*pd*)(*args*)ρ $\triangleq$
let *handler* = *current-exception-handler* (ρ) in 318
cases *pd*:
ATTACH $\rightarrow$ *m-ATTACH*(*args*) , 371

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  DETACH          → m-DETACH(args),                                  372
  IOTRANSFER      → m-IOTRANSFER(args) handler,                      372
  LISTEN          → m-LISTEN(args),                                  373
  NEWCOROUTINE    → m-NEWCOROUTINE(args) handler,                    375
  TRANSFER        → m-TRANSFER(args) handler                         376
end
```

7.2.3.1 The Procedure ATTACH

The procedure ATTACH can be used to associate a coroutine with an interrupt source.

Static Semantics

A call of ATTACH shall have one actual parameter that shall be an expression that is of the interrupt source type.

$wf\text{-}ATTACH : Actual\text{-}parameters \to Environment \to \mathbb{B}$

$wf\text{-}ATTACH\,(aps)\rho \triangleq$
  len $aps = 1 \wedge$
  let $[source] = aps$ in
  $is\text{-}Expression(source) \wedge$
  $t\text{-}expression\,(source)\rho = \text{INTERRUPTSOURCE-TYPE}$ 190

Dynamic Semantics

The call ATTACH(source) shall associate the implementation-defined source of interrupts identified by source with the calling coroutine. If the source of interrupts is already associated with a coroutine, then that association shall first be broken.

NOTES

1 More than one source of interrupts may be associated with a single coroutine.

2 If interrupt requests are possible from the attached source, execution of the calling coroutine must first be protected from interrupts — see 6.1.11. The protection of the calling coroutine must continue as long as interrupt requests are possible.

$m\text{-}ATTACH : Arguments \xrightarrow{o} ()$

$m\text{-}ATTACH\,(arg) \triangleq$
  let $[source] = arg$ in
  $attach(source)$ 343

7.2.3.2 The Procedure DETACH

The procedure DETACH can be used to dissociate a coroutine from an interrupt source.

Static Semantics

A call of DETACH shall have one actual parameter that shall be an expression that is of the interrupt source type.

$wf\text{-}DETACH : Actual\text{-}parameters \to Environment \to \mathbb{B}$

$wf\text{-}DETACH\,(aps)\rho \triangleq$
  len $aps = 1 \wedge$
  let $[source] = aps$ in
  $is\text{-}Expression(source) \wedge$
  $t\text{-}expression\,(source)\rho = \text{INTERRUPTSOURCE-TYPE}$ 190

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Dynamic Semantics

If the calling coroutine is associated with the source of interrupts identified by `source`, the call `DETACH(source)` shall dissociate that coroutine from that source.

NOTE — The call has no effect if the coroutine is not associated with `source`.

m-DETACH : *Arguments* $\xrightarrow{o}$ ()

m-DETACH (*arg*) $\triangleq$
  let [*source*] = *arg* in
  detach(*source*) 343

7.2.3.3 The Procedure IOTRANSFER

The procedure `IOTRANSFER` can be used to transfer control from one coroutine to another, pending an interrupt.

Static Semantics

A call of `IOTRANSFER` shall have two actual parameters. The first actual parameter shall designate a variable that is of the coroutine type. The second actual parameter shall be an expression that is of the coroutine type.

CHANGE — In *Programming in Modula-2*, the association with the source of interrupts is made on each call of `IOTRANSFER`. In this part of ISO/IEC 10514, the association is made beforehand by a call or calls to `ATTACH`.

wf-IOTRANSFER : *Actual-parameters* → *Environment* → $\mathbb{B}$

wf-IOTRANSFER (*aps*)ρ $\triangleq$
  len *aps* = 2 ∧
  let [*from*, *to*] = *aps* in
  is-Variable-designator(*from*) ∧
  t-variable-designator (*from*)ρ = COROUTINE-TYPE ∧ 182
  is-Expression(*to*) ∧
  t-expression (*to*)ρ = COROUTINE-TYPE 190

Dynamic Semantics

Provided that the calling coroutine is associated with a source of interrupts, the call `IOTRANSFER(from,to)` shall assign the identity of that coroutine to the variable `from`, and shall transfer control to the coroutine identified by `to`. An exception shall be raised if the calling coroutine is not associated with a source of interrupts.

On the granting of a subsequent interrupt, following a request from one of the sources associated with the calling coroutine, the coroutine that is then executing shall be suspended, the identity of that interrupted coroutine shall be assigned to `from`, and the calling coroutine shall be resumed.

NOTE — The coroutine that is transferred to by a call of `IOTRANSFER`, or one that is subsequently transferred to, must execute without protection from interrupts in order for an interrupt request to be granted.

m-IOTRANSFER : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

m-IOTRANSFER (*args*)*handler* $\triangleq$
  let [*from*, *to*] = *args* in
  def *attached* = *is-attached*() in 343
  if *attached*
  then def *current* = *current-coroutine*() in 346
    (*assign*(*from*, *current*) *handler*; 150
     let *mk-Elementary-variable* (*loc*, -) = *from* in
     target-for-interrupted-coroutine(*loc*) ; 343
     resume(*to*)) 345

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

else *mandatory-detection*(NOT-ATTACHED) *handler* 396

interrupt-occurs : *Interrupt-source* $\xrightarrow{o}$ ()

interrupt-occurs (*source*) $\triangleq$
transfer-to-handler(*source*) 343

annotations This operation models the interruption of a coroutine following an interrupt request from *source*. The source will be associated with a coroutine that has called IOTRANSFER. The operation is called by the coroutine that is executing when the interrupt request is granted.

7.2.3.4 The Procedure LISTEN

The procedure LISTEN can be used to alter the protection momentarily (usually lowering it so as to allow interrupt requests to be granted).

Static Semantics

A call of LISTEN shall have one parameter that is an expression that is of the protection type.

wf-LISTEN : *Actual-parameters* → *Environment* → $\mathbb{B}$

wf-LISTEN (*aps*)ρ $\triangleq$
len *aps* = 1 $\wedge$
let [*p*] = *aps* in
t-expression (*p*)ρ = PROTECTION-TYPE 190

Dynamic Semantics

The call LISTEN(p) shall momentarily change the protection of the currently executing coroutine to p.

m-LISTEN : *Arguments* $\xrightarrow{o}$ ()

m-LISTEN (*args*) $\triangleq$
let [*p*] = *args* in
def *curr-prot* = *get-protection*() in 344
(*set-protection*(*p*) ; 344
— grant an interrupt request from a source not excluded by protection *p*
set-protection(*curr-prot*)) 344

annotations If an interrupt request is granted, the effect is for a call of *interrupt-occurs*to be made.

7.2.3.5 The Procedure NEWCOROUTINE

The procedure NEWCOROUTINE can be used to create a new coroutine. Control can later be passed to this coroutine by calling the procedures TRANSFER or IOTRANSFER.

Static Semantics

A call of NEWCOROUTINE shall have four or five actual parameters.

The first actual parameter shall be an expression of a type with which the proc type is value-parameter-compatible.

NOTE 1 — The value of the expression that is the first actual parameter in a call of NEWCOROUTINE must therefore be a parameterless procedure declared at declaration level 0.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The second actual parameter shall be an expression that is of a pointer type.

The third actual parameter shall be an expression of a type with which the unsigned type is value parameter compatible.

The fourth actual parameter shall designate a variable that is of the coroutine type.

The fifth actual parameter, if present, shall be an expression that is of the protection type.

CHANGE 1 — In *Programming in Modula-2*, no provision is made for the initial protection of a coroutine to be specified in a call of `NEWCOROUTINE`.

$$
\begin{array}{l}
\mathit{wf\text{-}NEWCOROUTINE} : \mathit{Actual\text{-}parameters} \rightarrow \mathit{Environment} \rightarrow \mathbb{B} \\
\mathit{wf\text{-}NEWCOROUTINE}\,(\mathit{aps})\rho \triangleq \\
\quad (\mathsf{len}\ \mathit{aps} = 4 \vee \mathsf{len}\ \mathit{aps} = 5) \wedge \\
\quad \mathsf{let}\ [\mathit{proc}, \mathit{space}, \mathit{size}, \mathit{cr}] \curvearrowright \mathit{prios} = \mathit{aps}\ \mathsf{in} \\
\quad \mathit{is\text{-}Expression}(\mathit{proc}) \wedge \\
\quad \mathsf{let}\ \mathit{ptype} = \mathit{t\text{-}expression}\,(\mathit{proc})\rho\ \mathsf{in} \qquad 190 \\
\quad \mathit{is\text{-}value\text{-}parameter\text{-}compatible}\,(\text{PROC-TYPE}, \mathit{ptype})\rho \wedge \qquad 262 \\
\quad \mathit{is\text{-}Expression}(\mathit{space}) \wedge \\
\quad \mathit{is\text{-}pointer\text{-}type}\,(\mathit{t\text{-}expression}\,(\mathit{space})\rho)\rho \wedge \qquad 322\ 190 \\
\quad \mathit{is\text{-}Expression}(\mathit{size}) \wedge \\
\quad \mathsf{let}\ \mathit{stype} = \mathit{t\text{-}expression}\,(\mathit{size})\rho\ \mathsf{in} \qquad 190 \\
\quad \mathit{is\text{-}value\text{-}parameter\text{-}compatible}\,(\text{UNSIGNED-TYPE}, \mathit{stype})\rho \wedge \qquad 262 \\
\quad \mathit{is\text{-}Variable\text{-}designator}(\mathit{cr}) \wedge \\
\quad \mathit{t\text{-}variable\text{-}designator}\,(\mathit{cr})\rho = \text{COROUTINE-TYPE}) \wedge \qquad 182 \\
\quad (\mathit{prios} = [\,] \vee \\
\quad \mathsf{let}\ [\mathit{prio}] = \mathit{prios}\ \mathsf{in} \\
\quad \mathit{is\text{-}Expression}(\mathit{prio}) \wedge \\
\quad \mathit{t\text{-}expression}\,(\mathit{prio})\rho = \text{PROTECTION-TYPE}) \qquad 190
\end{array}
$$

Dynamic Semantics

Provided that the value of `size` is not less than the minimum workspace size, the call

```
NEWCOROUTINE(procBody,workspace,size,cr,initProtection)
```

shall create a new coroutine and shall assign a value identifying the created coroutine to the variable `cr`. The initial protection of the coroutine shall be given by `initProtection`.

Provided that the value of `size` is not less than the minimum workspace size, the alternative call

```
NEWCOROUTINE(procBody,workspace,size,cr)
```

shall create a new coroutine and shall assign a value identifying the created coroutine to the variable `cr`. The initial protection of the coroutine shall be given by the current protection of the caller of `NEWCOROUTINE`.

The created coroutine shall be initialized in such a way that when control is first transferred to that coroutine, the procedure given by the value of `procBody` will be called as the body of that coroutine. The effect shall be for the caller of `procBody` to be in the state of normal execution with no block body as an exception handler. The effect shall also be for the caller of `procBody` to have a protection from interrupts given by the initial protection for the coroutine.

CHANGE 2 — *Programming in Modula-2* does not deal with the initial protection of a new coroutine. This part of ISO/IEC 10514 defines rules for determining that initial protection.

If a procedure called as the body of a coroutine attempts to return, an exception shall be raised.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NOTES

2 If the procedure has local dynamic modules, these are finalized before the exception is raised.

3 Since the effective caller has no exception handler, the raising of this exception is a termination event.

CLARIFICATION — *Programming in Modula-2* states that a program terminates when control reaches the end of a procedure which is the body of a coroutine. This part of ISO/IEC 10514 requires that an unhandled exception is raised on any attempt to return to the effective caller of the body of a dynamically created coroutine.

The storage for a variable occupying a number of locations given by the value of `size` and having an address given by the value of `workspace` shall be given to the coroutine as its workspace. The use made of this workspace shall be implementation-dependent.

The minimum workspace size shall be implementation-defined. An exception shall be raised if the value of `size` is smaller than the minimum workspace size.

NOTES

4 In addition to coroutines created by `NEWCOROUTINE`, there is an original coroutine, or main coroutine, by which the static modules of the program are initialized. The initial protection of the main coroutine is implementation-defined and the workspace is supplied by the implementation.

5 The creation of a coroutine does not cause a transfer of control.

6 To ensure that no interrupts are allowed on the initial transfer to a coroutine, it is not sufficient for the procedure given as the value of the `procBody` parameter in a call of `NEWCOROUTINE` to be a protected procedure with a protection of `UNINTERRUPTIBLE`. That protection will only be established when the procedure is called (automatically) in the new coroutine.

7 A call of `NEWCOROUTINE` does not check if the given workspace overlaps with that of other coroutines; provided that a variable remains in existence until the end of program execution, the storage for that variable may be given to a coroutine for use as its workspace.

8 The giving of coroutine workspace allows the possibility of aliasing. By accessing the workspace through the variable whose storage is given in the call of `NEWCOROUTINE` it may be possible for a program to alter the values of coroutine variables or to affect control information such as procedure return addresses. Two consequences that follow are:

a) Since the use of coroutine workspace is implementation-dependent, no meaning is given to such a program by this part of ISO/IEC 10514.

b) Depending on the implementation, it may not be possible to perform a static check that a for loop control variable in a coroutine is not threatened in this way.

9 Workspace overflow is checked during coroutine execution. The check may be carried out when a procedure is activated when storage is allocated for its local variables (see 6.12.8.3)

m-NEWCOROUTINE : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

m-NEWCOROUTINE (*args*)*handler* $\triangleq$
  let [*proc*, *space*, *size*, *cr*] $\curvearrowright$ *prios* = *args* in
  def *v* = *is-adequate-work-space*(*space*, *size*) in 376
  if *v*
  then let *cont* = *coroutine-continuation*(*proc*) in 375
    def *cid* = *allocate-coroutine-id*() in 345
    def *prio* = if *prios* = [] then *get-protection*() else hd *prios* in 344
    (*reserve-coroutine-workspace*(*cid*, *space*, *size*, *prio*) ; 345
     assign(*cr*, *cid*) *handler*; 150
     initialize(*cid*, *cont*)) 345
  else *mandatory-detection*(SMALL-WORKSPACE) *handler* 396

coroutine-continuation : *Procedure-id* → *Program-cont*

coroutine-continuation (*proc*) $\triangleq$
  let *handler* = (*termination*() ; *exception-actions*()) in 41 341
  (*call-proper-procedure-value*(*proc*) ([]) *handler* ; 339
   mandatory-detection(RETURN-FROM-COROUTINE) *handler*) 396

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Auxiliaries

is-adequate-work-space : *Address-value* × N → 𝔹

is-adequate-work-space (*addr*, *amount*) ≜
  implementation-defined

7.2.3.6 The Procedure TRANSFER

The procedure TRANSFER can be used to transfer control from one coroutine to another.

Static Semantics

A call of TRANSFER shall have two actual parameters. The first actual parameter shall designate a variable that is of the coroutine type. The second actual parameter shall be an expression that is of the coroutine type.

wf-TRANSFER : *Actual-parameters* → *Environment* → 𝔹

wf-TRANSFER (*aps*)ρ ≜
  len *aps* = 2 ∧
  let [*from*, *to*] = *aps* in
  is-Variable-designator(*from*) ∧
  t-variable-designator (*from*)ρ = COROUTINE-TYPE ∧ 182
  is-Expression(*to*) ∧
  t-expression (*to*)ρ = COROUTINE-TYPE 190

Dynamic Semantics

The call TRANSFER(from,to) shall assign the identity of the calling coroutine to the variable from, and shall transfer control to the coroutine identified by to.

NOTE — If at the end of its previous execution (if any) the coroutine identified by to had transferred control to another by a call of IOTRANSFER, it will appear as if an interrupt has occurred.

m-TRANSFER : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

m-TRANSFER (*arg*)*handler* ≜
  (let [*var*, *val*] = *arg* in
   def *from* = *current-coroutine*() in 346
   assign(*var*, *from*) *handler* ; 150
   resume(*val*)) 345

7.2.4 The Functions of COROUTINES

The module COROUTINES provides functions for determining the identity of the current coroutine (CURRENT), the protection of the current coroutine (PROT), and for determining whether and which interrupt sources are associated with coroutines (HANDLER and IsATTACHED).

Abstract Syntax

types

  COROUTINES-function-designator :: *ident* : *Identifier*

  inv *ident* ≜
    ident ∈ {HANDLER, IsATTACHED, CURRENT, PROT}

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Static Semantics

wf-COROUTINES-function-designator :
COROUTINES-function-designator → *Actual-parameters* → *Environment* → 𝔹

wf-COROUTINES-function-designator $(fd)(aps)\rho \triangleq$
access-environment $(desig)\rho$ = COROUTINE-FUNCTION-PROCEDURE ∧ 319
cases *fd* :
HANDLER → *wf-HANDLER* $(aps)\rho$, 377
IsATTACHED → *wf-IsATTACHED* $(aps)\rho$, 378
CURRENT → *wf-CURRENT* $(aps)\rho$, 379
PROT → *wf-PROT* $(aps)\rho$ 379
end

t-COROUTINES-function-designator :
COROUTINES-function-designator → *Actual-parameters* → *Environment* → *Typed*

t-COROUTINES-function-designator $(fd)(aps)\rho \triangleq$
cases *fd* :
HANDLER → *t-HANDLER* $(aps)\rho$, 377
IsATTACHED → *t-IsATTACHED* $(aps)\rho$, 378
CURRENT → *t-CURRENT* $(aps)\rho$, 379
PROT → *t-PROT* $(aps)\rho$ 379
end

Dynamic Semantics

m-COROUTINES-function-designator :
COROUTINES-function-designator → *Arguments* → *Environment* $\xrightarrow{o}$ *Value*

m-COROUTINES-function-designator $(fd)(args)\rho \triangleq$
let *handler* = *current-exception-handler* (ρ) in 318
cases *cfd*:
HANDLER → *m-HANDLER*(*args*) *handler*, 378
IsATTACHED → *m-IsATTACHED*(*args*) , 378
CURRENT → *m-CURRENT*(*args*) , 379
PROT → *m-PROT*(*args*) 379
end

7.2.4.1 The Function HANDLER

The function HANDLER can be used to identify which coroutine is attached to a specified interrupt source.

Static Semantics

A call of HANDLER shall have one actual parameter that is an expression that is of the interrupt source type. The type of a call of HANDLER shall be the coroutine type.

wf-HANDLER : *Actual-parameters* → *Environment* → 𝔹

wf-HANDLER $(aps)\rho \triangleq$
len *aps* = 1 ∧
let [*source*] = *aps* in
is-Expression(*source*) ∧
is-Interrupt-source-type(*t-expression* (*source*)ρ) 190

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

t-HANDLER : *Actual-parameters* → *Environment* → *Typed*

t-HANDLER (*aps*)ρ $\triangleq$
  COROUTINE-TYPE

Dynamic Semantics

The value of the call `HANDLER(source)` shall be the identity of the coroutine, if any, that is associated with the source of interrupts given by `source`. An exception shall occur and may be raised if that source is not currently associated with any coroutine.

m-HANDLER : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Coroutine*

m-HANDLER (*aps*)*handler* $\triangleq$
  let [*source*] = *aps* in
  def *handlers* = *program-handlers*() in 343
  if *source* ∈ dom *handlers*
  then let *mk-Handler* (*id*, -) = *handlers*(*source*) in
    return *id*
  else *non-mandatory-detection*(NO-HANDLER) *handler* 397

7.2.4.2 The Function IsATTACHED

The function `IsATTACHED` can be used to determine whether a coroutine is attached to a specified interrupt source.

Static Semantics

A call of `IsATTACHED` shall have one actual parameter that is an expression that is of the interrupt source type. The type of a call of `IsATTACHED` shall be the coroutine type.

wf-IsATTACHED : *Actual-parameters* → *Environment* → $\mathbb{B}$

wf-IsATTACHED (*aps*)ρ $\triangleq$
  len *aps* = 1 ∧
  let [*source*] = *aps* in
  is-Expression(*source*) ∧
  t-expression (*source*)ρ = INTERRUPTSOURCE-TYPE 190

t-IsATTACHED : *Actual-parameters* → *Environment* → *Typed*

t-IsATTACHED (*aps*)ρ $\triangleq$
  BOOLEAN-TYPE

Dynamic Semantics

The value of the call `IsATTACHED(source)` shall be *true* if any coroutine is associated with the source of interrupts denoted by `source`, and *false* otherwise.

m-IsATTACHED : *Arguments* $\xrightarrow{o}$ $\mathbb{B}$

m-IsATTACHED (*aps*) $\triangleq$
  let [*source*] = *aps* in
  def *handlers* = *program-handlers*() in 343
  return *source* ∈ dom *handlers*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

7.2.4.3 The Function CURRENT

The function CURRENT can be used to determine the identity of the currently executing coroutine.

Static Semantics

A call of CURRENT shall have no actual parameters. The type of a call of CURRENT shall be the coroutine type.

wf-CURRENT : *Actual-parameters* → *Environment* → $\mathbb{B}$
wf-CURRENT (*aps*)ρ $\triangleq$
  len *aps* = 0

t-CURRENT : *Actual-parameters* → *Environment* → *Typed*
t-CURRENT (*aps*)ρ $\triangleq$
  COROUTINE-TYPE

Dynamic Semantics

The value of the call CURRENT() shall be the identity of the calling coroutine.

m-CURRENT : *Arguments* $\xrightarrow{o}$ *Value*
m-CURRENT (*args*) $\triangleq$
  current-coroutine() 346

7.2.4.4 The Function PROT

The function PROT can be used to obtain the protection of the currently executing coroutine, that is, the extent to which interrupts are currently masked.

Static Semantics

A call of PROT shall have no actual parameters. The type of a call of PROT shall be the protection type.

wf-PROT : *Actual-parameters* → *Environment* → $\mathbb{B}$
wf-PROT (*aps*)ρ $\triangleq$
  len *aps* = 0

t-PROT : *Actual-parameters* → *Environment* → *Typed*
t-PROT (*aps*)ρ $\triangleq$
  PROTECTION-TYPE

Dynamic Semantics

The value of the call PROT() shall be the value of the current protection against interrupts.

m-PROT : *Arguments* $\xrightarrow{o}$ *Value*
m-PROT (*args*) $\triangleq$
  get-protection() 344

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

7.3 The Module EXCEPTIONS

The system module EXCEPTIONS provides low-level facilities for identifying the source of user-defined exceptions, for reporting their occurrence, and for making enquiries concerning the execution state of the current coroutine.

User-defined exceptions are identified uniquely by an exception source/number pair. When the source of a user-defined exception is a separate module, that module may encapsulate the exception source value, thus preventing the defined exceptions of the module from being raised directly by other sources. The exception number may also be mapped to values of an enumeration type defined by the module. An example of the use of these facilities is given in 7.3.4.4.

7.3.1 The Interface to EXCEPTIONS

The interface to EXCEPTIONS behaves as if the following were its definition module.

```
DEFINITION MODULE EXCEPTIONS;

(* Provides facilities for raising user exceptions
   and for making enquiries concerning the current execution state.
*)

TYPE
  ExceptionSource;                (* values of this type are used within library modules to
                                     identify the source of raised exceptions    *)
  ExceptionNumber = CARDINAL;

PROCEDURE AllocateSource (VAR newSource: ExceptionSource);
  (* Allocates a unique value of type ExceptionSource *)

PROCEDURE RAISE (source: ExceptionSource; number: ExceptionNumber; message: ARRAY OF CHAR);
  (* Associates the given values of source, number and message with the current context
     and raises an exception.
  *)

PROCEDURE CurrentNumber (source: ExceptionSource): ExceptionNumber;
  (* If the current coroutine is in the exceptional execution state because of the raising
     of an exception from source, returns the corresponding number, and otherwise
     raises an exception.
  *)

PROCEDURE GetMessage (VAR text: ARRAY OF CHAR);
  (* If the current coroutine is in the exceptional execution state, returns the possibly
     truncated string associated with the current context.
     Otherwise, in normal execution state, returns the empty string.
  *)

PROCEDURE IsCurrentSource (source: ExceptionSource): BOOLEAN;
  (* If the current coroutine is in the exceptional execution state because of the raising
     of an exception from source, returns TRUE, and otherwise returns FALSE.
  *)

PROCEDURE IsExceptionalExecution (): BOOLEAN;
  (* If the current coroutine is in the exceptional execution state because of the raising
     of an exception, returns TRUE, and otherwise returns FALSE.
  *)

END EXCEPTIONS.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Declaration Semantics

The identifiers available for import from the module `EXCEPTIONS` shall be the type identifiers `ExceptionSource` and `ExceptionNumber`, and the procedure identifiers `AllocateSource`, `RAISE`, `CurrentNumber`, `GetMessage`, `IsCurrentSource`, and `IsExceptionalExecution`.

values

$\rho_{EXCEPTIONS}$ = let *constants* = {↦},
types = {`ExceptionSource` ↦ EXCEPTIONSOURCE-TYPE,
`ExceptionNumber` ↦ UNSIGNED-TYPE},
structures = {↦},
variables = {↦},
procedures = {`AllocateSource` ↦ EXCEPTIONS-PROPER-PROCEDURE,
`RAISE` ↦ EXCEPTIONS-PROPER-PROCEDURE,
`CurrentNumber` ↦ EXCEPTIONS-FUNCTION-PROCEDURE,
`GetMessage` ↦ EXCEPTIONS-PROPER-PROCEDURE,
`IsCurrentSource` ↦ EXCEPTIONS-FUNCTION-PROCEDURE,
`IsExceptionalExecution` ↦ EXCEPTIONS-FUNCTION-PROCEDURE},
modules = {↦},
defmods = {↦},
level = 0,
local-continuations = {↦},
protection-domain = nil ,
protected-procedures = { } in
mk-Qualified-environment (*constants*, *types*, *structures*, *variables*, *procedures*, *modules*, *defmods*, *level*, *local-continuations*, *protection-domain*, *protected-procedures*)

7.3.2 The Types of EXCEPTIONS

The module `EXCEPTIONS` provides the exception source type (denoted by `ExceptionSource`) in terms of which the sources of user-defined exceptions are identified, and the exception number type (denoted by `ExceptionNumber`) in terms of which exceptions from a particular source are identified.

Declaration Semantics

The type identifier `ExceptionSource` shall denote the exception source type, and the type identifier `ExceptionNumber` shall denote the whole number type.

NOTE — In addition to use with the `EXCEPTIONS` procedures `AllocateSource`, `RAISE`, `CurrentNumber`, and `IsCurrentSource`, as a system basic type, the operations of assignment and equality testing are applicable to the exception source type — see 6.8.2.7.7.

annotations Values of the exception source type are represented by values of the type *Exception-source* in the formal model — see 6.12.6.

7.3.3 The Procedures of EXCEPTIONS

The module `EXCEPTIONS` provides procedures for allocating a unique value of the exception source type (`AllocateSource`), raising a user-defined exception (`RAISE`), and obtaining the message associated with the current exception context (`GetMessage`).

Abstract Syntax

types

EXCEPTIONS-procedure-designator :: *ident* : *Identifier*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

inv *ident* $\triangleq$
  ident $\in$ {`AllocateSource`, `RAISE`, `GetMessage`}

Static Semantics

wf-EXCEPTIONS-procedure-designator :
  EXCEPTIONS-procedure-designator $\rightarrow$ *Actual-parameters* $\rightarrow$ *Environment* $\rightarrow \mathbb{B}$

wf-EXCEPTIONS-procedure-designator (*pd*)(*aps*)ρ $\triangleq$
  access-environment (*desig*)ρ = EXCEPTIONS-PROPER-PROCEDURE $\wedge$ 319
  cases *pd* :
    `AllocateSource` $\rightarrow$ *wf-AllocateSource* (*aps*)ρ, 382
    `RAISE` $\rightarrow$ *wf-RAISE* (*aps*)ρ, 383
    `GetMessage` $\rightarrow$ *wf-GetMessage* (*aps*)ρ 384
  end

Dynamic Semantics

m-EXCEPTIONS-procedure-designator :
  EXCEPTIONS-procedure-designator $\rightarrow$ *Arguments* $\rightarrow$ *Environment* $\xrightarrow{o}$ ()

m-EXCEPTIONS-procedure-designator (*pd*)(*args*)ρ $\triangleq$
  let *handler* = *current-exception-handler* (ρ) in 318
   cases *pd*:
    `AllocateSource` $\rightarrow$ *m-AllocateSource*(*args*) *handler*, 382
    `RAISE` $\rightarrow$ *m-RAISE*(*args*) *handler*, 383
    `GetMessage` $\rightarrow$ *m-GetMessage*(*args*) *handler* 384
  end

7.3.3.1 The Procedure AllocateSource

The procedure `AllocateSource` can be used as a server for previously unallocated values of the type `ExceptionSource`. Values of this type are used to identify a source of exceptions, such as a library module, when an exception is raised with the procedure `RAISE`. The procedure is normally called once during initialization of a separate module, and the resulting value is then used in all calls of `RAISE` for exceptions that may be handled by clients of the module.

Static Semantics

A call of `AllocateSource` shall have one actual parameter which shall be a variable that is of the exception source type.

wf-AllocateSource : *Actual-parameters* $\rightarrow$ *Environment* $\rightarrow \mathbb{B}$

wf-AllocateSource (*aps*)ρ $\triangleq$
  len *aps* = 1 $\wedge$
  let [*newsource*] = *aps* in
  is-Variable-designator(*newsource*) $\wedge$
  let *stype* = *t-entire-designator* (*newsource*)ρ in 183
  stype = EXCEPTIONSOURCE-TYPE

Dynamic Semantics

The call `AllocateSource(newSource)` shall allocate a value of the exception source type that is unique within the program and shall assign this value to the variable `newSource`. If a unique value cannot be allocated, an exception shall be raised.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

m-AllocateSource : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

m-AllocateSource (*aps*)*handler* $\triangleq$
  let [*newsource*] = *aps* in
  def *source* = *allocate-source*() in 342
  if *source* ≠ nil
  then *assign*(*newsource*, *source*) *handler* 150
  else *mandatory-detection*(NO-EXCEPTION-SOURCE) *handler* 396

7.3.3.2 The Procedure RAISE

The procedure RAISE can be used by a specified source to raise a user-defined exception.

The effect of raising an exception is the same for exceptions raised by the language implementation as for exceptions raised by explicit use of the procedure RAISE — see 7.5.4. Thus the raising of a user-defined exception is a termination event (see 6.1.7) unless the exception is handled by an exception body (see 6.5.8).

Static Semantics

A call of RAISE shall have three actual parameters. The first actual parameter shall be an expression that is of the exception source type, the second actual parameter shall be an expression of a type with which the unsigned type is value parameter compatible, and the third actual parameter shall be an expression of a type with which an open array of CHAR is value parameter compatible.

wf-RAISE : *Actual-parameters* → *Environment* → $\mathbb{B}$

wf-RAISE (*aps*)ρ $\triangleq$
  len *aps* = 3 ∧
  let [*source*, *number*, *message*] = *aps* in
  is-Expression(*source*) ∧ *t-expression* (*source*)ρ = EXCEPTIONSOURCE-TYPE ∧ 190
  is-Expression(*number*) ∧
  let *tnumber* = *t-expression* (*number*)ρ in 190
  is-value-parameter-compatible (UNSIGNED-TYPE, *tnumber*)ρ ∧ 262
  is-Expression(*message*) ∧
  let *etype* = *t-expression* (*message*)ρ in 190
  is-String-type(*etype*)ρ ∨
  (*is-array-type* (*etype*)ρ ∧ 321
  component-type-of (*etype*)ρ = CHARACTER-TYPE) 325

Dynamic Semantics

The call of `RAISE(source, number, message)` shall set the current exception source to the value of `source`, the current exception number to the value of `number`, and the current exception message to the value of `message` interpreted as a character string and limited to an implementation-defined length.

m-RAISE : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

m-RAISE ((*aps*)*handler*) $\triangleq$
  let [*source*, *number*, *message*] = *aps* in
  raise-exception(*mk-User-exception-context* (*source*, *number*, *message*)) *handler* 342

7.3.3.3 The Procedure GetMessage

The procedure `GetMessage` can be used to obtain the message passed when an exception is raised. This may give further information about the nature of the exception for use by the program, or in the construction of other messages.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Static Semantics

A call of `GetMessage` shall have one actual parameter which shall be an array variable having components of the character type.

wf-GetMessage : *Actual-parameters* → *Environment* → 𝔹
wf-GetMessage (*aps*)ρ ≜
  len *aps* = 1 ∧
  let [*text*] = *aps* in
  is-Variable-designator(*text*) ∧
  let *dtype* = *t-variable-designator* (*text*)ρ in 182
  is-array-type (*dtype*)ρ ∧ 321
  component-type-of (*dtype*)ρ = CHARACTER-TYPE 325

Dynamic Semantics

The call `GetMessage(text)` shall assign a string value to the variable `text`. If the calling coroutine is in the state of exceptional execution, the string value shall be formed from the message stored when the corresponding exception was raised, but with a length limited by the capacity of `text`.

In the state of normal execution, the string assigned to `text` shall be the empty string.

m-GetMessage : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()
m-GetMessage (*aps*)*handler* ≜
  let [*text*] = *aps* in
  def *context* = *top-exception-context*() in 342
  let *message* = cases *context* :
      NORMAL-EXECUTION → [],
      mk-User-exception (-, -, *mesg*) → *mesg*,
      mk-Language-exception (-, *mesg*) → *mesg*
    end in
  assign(*text*, *message*) *handler* 150

7.3.4 The Functions of EXCEPTIONS

The module `EXCEPTIONS` provides functions for determining the exception number for the current user-defined exception (`CurrentNumber`), whether a user-defined exception has been raised by a given source (`IsCurrentSource`), and whether the execution state is that of exceptional execution (`IsExceptionalExecution`).

Abstract Syntax

types

EXCEPTIONS-function-designator :: *ident* : *Identifier*

inv *ident* ≜
  ident ∈ {`CurrentNumber`, `IsCurrentSource`, `IsExceptionalExecution`}

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Static Semantics

wf-EXCEPTIONS-function-designator :
EXCEPTIONS-function-designator → *Actual-parameters* → *Environment* → 𝔹

wf-EXCEPTIONS-function-designator (*fd*)(*aps*)ρ ≙
access-environment (*desig*)ρ = EXCEPTIONS-FUNCTION-PROCEDURE ∧ 319
cases *fd* :
`CurrentNumber` → *wf-CurrentNumber* (*aps*)ρ, 385
`IsCurrentSource` → *wf-IsCurrentSource* (*aps*)ρ, 386
`IsExceptionalExecution` → *wf-IsExceptionalExecution* (*aps*)ρ 387
end

t-EXCEPTIONS-function-designator :
EXCEPTIONS-function-designator → *Actual-parameters* → *Environment* → *Typed*

t-EXCEPTIONS-function-designator (*fd*)(*aps*)ρ ≙
cases *fd* :
`CurrentNumber` → *t-CurrentNumber* (*aps*)ρ, 385
`IsCurrentSource` → *t-IsCurrentSource* (*aps*)ρ, 386
`IsExceptionalExecution` → *t-IsExceptionalExecution* (*aps*)ρ 387
end

Dynamic Semantics

m-EXCEPTIONS-function-designator :
EXCEPTIONS-function-designator → *Arguments* → *Environment* $\xrightarrow{o}$ *Value*

m-EXCEPTIONS-function-designator (*fd*)(*args*)ρ ≙
let *handler* = *current-exception-handler* (ρ) in 318
cases *fd*:
`CurrentNumber` → *m-CurrentNumber*(*args*) *handler*, 386
`IsCurrentSource` → *m-IsCurrentSource*(*args*) , 386
`IsExceptionalExecution` → *m-IsExceptionalExecution*(*args*) 387
end

7.3.4.1 The Function CurrentNumber

The function `CurrentNumber` can be used to obtain the exception number for the current user-defined exception that has been raised.

Static Semantics

A call of `CurrentNumber` shall have one actual parameter which shall be an expression of the exception source type. The type of a call of `CurrentNumber` shall be the unsigned type.

wf-CurrentNumber : *Actual-parameters* → *Environment* → 𝔹

wf-CurrentNumber (*aps*)ρ ≙
len *aps* = 1 ∧
let [*source*] = *aps* in
is-Expression(*source*) ∧
t-expression (*source*)ρ = EXCEPTIONSOURCE-TYPE 190

t-CurrentNumber : *Actual-parameters* → *Environment* → *Typed*

t-CurrentNumber (*aps*)ρ ≙
UNSIGNED-TYPE

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Dynamic Semantics

If the calling coroutine is in the state of exceptional execution because of the raising of a user-defined exception from `source`, the value of the call `CurrentNumber(source)` shall be the number that is associated with the raised exception; otherwise an exception shall be raised.

m-CurrentNumber : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ℕ

m-CurrentNumber (*args*)*handler* ≙
  let [*source*] = *aps* in
  def *context* = *top-exception-context*() in 342
  def *number* =
    cases *context* :
      NORMAL-EXECUTION → *mandatory-detection*(NO-CURRENT-EXCEPTION) *handler*, 396
      mk-User-exception (*from*, *num*, -) → if *from* = *source*
        then *num*
        else *mandatory-detection*(NOT-CURRENT-SOURCE) *handler*, 396
      mk-Language-exception (-, -) → *mandatory-detection*(NOT-CURRENT-SOURCE) *handler* 396
    end in
  return *number*

7.3.4.2 The Function IsCurrentSource

The function `IsCurrentSource` can be used to test whether the calling coroutine is in the state of exceptional execution because of the raising of a user-defined exception from a specified source.

Static Semantics

A call of `IsCurrentSource` shall have one actual parameter which shall be an expression of the exception source type. The type of a call of `IsCurrentSource` shall be the Boolean type.

wf-IsCurrentSource : *Actual-parameters* → *Environment* → 𝔹

wf-IsCurrentSource (*aps*)ρ ≙
  len *aps* = 1 ∧
  let [*source*] = *aps* in
  is-Expression(*source*) ∧
  t-expression (*source*)ρ = EXCEPTIONSOURCE-TYPE 190

t-IsCurrentSource : *Actual-parameters* → *Environment* → *Typed*

t-IsCurrentSource (*aps*)ρ ≙
  BOOLEAN-TYPE

Dynamic Semantics

If the calling coroutine is in the state of exceptional execution because of the raising of a user-defined exception from `source`, the value of the call `IsCurrentSource(source)` shall be *true*; otherwise the value shall be *false*.

m-IsCurrentSource : *Arguments* $\xrightarrow{o}$ 𝔹

m-IsCurrentSource (*aps*) ≙
  let [*source*] = *aps* in
  def *context* = *top-exception-context*() in 342

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
let res = cases context :
            NORMAL-EXECUTION          → false,
            mk-User-exception (from, -, -) → source = from,
            mk-Language-exception (-, -)  → false
          end in
return res
```

7.3.4.3 The Function IsExceptionalExecution

The function `IsExceptionalExecution` can be used to test whether the calling coroutine is in the state of exceptional execution.

Static Semantics

A call of `IsExceptionalExecution` shall have no actual parameters. The type of a call of `IsExceptionalExecution` shall be the Boolean type.

```
wf-IsExceptionalExecution : Actual-parameters → Environment → 𝔹
wf-IsExceptionalExecution (aps)ρ ≜
  len aps = 0

t-IsExceptionalExecution : Actual-parameters → Environment → Typed
t-IsExceptionalExecution (aps)ρ ≜
  BOOLEAN-TYPE
```

Dynamic Semantics

If the calling coroutine is in the state of exceptional execution, the value of the call `IsExceptionalExecution()` shall be *true*; otherwise the value shall be *false*.

```
m-IsExceptionalExecution : Arguments →o 𝔹
m-IsExceptionalExecution (aps) ≜
  def context = top-exception-context() in                    342
  let res = cases context :
              NORMAL-EXECUTION          → false,
              mk-User-exception (-, -, -)   → true,
              mk-Language-exception (-, -) → true
            end in
  return res
```

7.3.4.4 The Use of EXCEPTIONS

The example given below shows the use of `EXCEPTIONS` in the implementation of a separate module that provides facilities to allow a general user-defined exception to be raised and identified.

EXAMPLE — A general user exception separate module.

```
DEFINITION MODULE GeneralUserExceptions;

(* Provides facilities for general user-defined exceptions *)

TYPE
  GeneralExceptions = (problem, disaster);

PROCEDURE RaiseGeneralException (exception: GeneralExceptions; text: ARRAY OF CHAR);
  (* Raises exception using text as the associated message *)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
PROCEDURE IsGeneralException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional execution state
     because of the raising of an exception from GeneralExceptions;
     otherwise returns FALSE.
  *)

PROCEDURE GeneralException(): GeneralExceptions;
  (* If the current coroutine is in the exceptional execution state because of the
     raising of an exception from GeneralExceptions, returns the corresponding
     enumeration value, and otherwise raises an exception.
  *)

END GeneralUserExceptions.


IMPLEMENTATION MODULE GeneralUserExceptions;

IMPORT EXCEPTIONS;

VAR
  generalSource: EXCEPTIONS.ExceptionSource;

PROCEDURE RaiseGeneralException (exception: GeneralExceptions; text: ARRAY OF CHAR);
  BEGIN
    EXCEPTIONS.RAISE(generalSource, ORD(exception), text)
  END RaiseGeneralException;

PROCEDURE IsGeneralException (): BOOLEAN;
  BEGIN
    RETURN EXCEPTIONS.IsCurrentSource(generalSource)
  END IsGeneralException;

PROCEDURE GeneralException(): GeneralExceptions;
  BEGIN
    RETURN VAL(GeneralExceptions, EXCEPTIONS.CurrentNumber(generalSource))
  END GeneralException;

BEGIN
  EXCEPTIONS.AllocateSource(generalSource)
END GeneralUserExceptions.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.4 The Module TERMINATION

The system module TERMINATION provides facilities for a program to enquire whether a termination event has occurred — see 6.1.7.

7.4.1 The Interface to TERMINATION

The interface to TERMINATION behaves as if the following were its definition module.

```
DEFINITION MODULE TERMINATION;

(* Provides facilities for enquiries concerning the occurrence of termination events. *)

PROCEDURE IsTerminating (): BOOLEAN ;
  (* Returns TRUE if any coroutine has started  program termination and FALSE otherwise. *)

PROCEDURE HasHalted (): BOOLEAN ;
  (* Returns TRUE if a call to HALT has been made and FALSE otherwise. *)

END TERMINATION.
```

Declaration Semantics

The identifiers available for import from the module TERMINATION shall be the procedure identifiers IsTerminating, and HasHalted.

values

$$
\begin{aligned}
\rho_{TERMINATION} = \mathsf{let}\ & \mathit{constants} = \{\mapsto\},\\
& \mathit{types} = \{\mapsto\},\\
& \mathit{structures} = \{\mapsto\},\\
& \mathit{variables} = \{\mapsto\},\\
& \mathit{procedures} = \{\texttt{IsTerminating} \mapsto \text{TERMINATION-FUNCTION-PROCEDURE},\\
& \qquad\qquad \texttt{HasHalted} \mapsto \text{TERMINATION-FUNCTION-PROCEDURE}\},\\
& \mathit{modules} = \{\mapsto\},\\
& \mathit{defmods} = \{\mapsto\},\\
& \mathit{level} = 0,\\
& \mathit{local\text{-}continuations} = \{\mapsto\},\\
& \mathit{protection\text{-}domain} = \mathsf{nil}\ ,\\
& \mathit{protected\text{-}procedures} = \{\ \}\ \mathsf{in}\\
& \mathit{mk\text{-}Qualified\text{-}environment}(\mathit{constants}, \mathit{types}, \mathit{structures}, \mathit{variables}, \mathit{procedures}, \mathit{modules}, \mathit{defmods},\\
& \qquad\qquad \mathit{level}, \mathit{local\text{-}continuations}, \mathit{protection\text{-}domain}, \mathit{protected\text{-}procedures})
\end{aligned}
$$

7.4.2 The Functions of TERMINATION

The module TERMINATION provides the function procedures IsTerminating and HasHalted for enquiries concerning the occurrence of termination events.

Abstract Syntax

types

$$\mathit{TERMINATION\text{-}function\text{-}designator} :: \mathit{ident} : \mathit{Identifier}$$

$$\mathsf{inv}\ \mathit{ident} \triangleq \mathit{ident} \in \{\texttt{IsTerminating}, \texttt{HasHalted}\}$$

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Static Semantics

wf-TERMINATION-function-designator :
TERMINATION-function-designator → *Actual-parameters* → *Environment* → $\mathbb{B}$

wf-TERMINATION-function-designator (*fd*)(*aps*)ρ $\triangleq$
access-environment (*desig*)ρ = TERMINATION-FUNCTION-PROCEDURE $\wedge$ 319
cases *fd* :
`IsTerminating` → *wf-IsTerminating* (*aps*)ρ, 390
`HasHalted` → *wf-HasHalted* (*aps*)ρ 391
end

t-TERMINATION-function-designator :
TERMINATION-function-designator → *Actual-parameters* → *Environment* → *Typed*

t-TERMINATION-function-designator (*fd*)(*aps*)ρ $\triangleq$
cases *fd* :
`IsTerminating` → *t-IsTerminating* (*aps*)ρ, 390
`HasHalted` → *t-HasHalted* (*aps*)ρ 391
end

Dynamic Semantics

m-TERMINATION-function-designator :
TERMINATION-function-designator → *Arguments* → *Environment* $\xrightarrow{o}$ *Value*

m-TERMINATION-function-designator (*fd*)(*args*)ρ $\triangleq$
cases *fd*:
`IsTerminating` → *m-IsTerminating*(*args*) , 390
`HasHalted` → *m-HasHalted*(*args*) 391
end

7.4.2.1 The Function IsTerminating

The function `IsTerminating` can be used to determine whether any coroutine has started program termination.

Static Semantics

A call of `IsTerminating` shall have no parameters. The type of a call of `IsTerminating` shall be the Boolean type.

wf-IsTerminating : *Actual-parameters* → *Environment* → $\mathbb{B}$

wf-IsTerminating (*aps*)ρ $\triangleq$
aps = []

t-IsTerminating : *Actual-parameters* → *Environment* → *Typed*

t-IsTerminating (*aps*)ρ $\triangleq$
BOOLEAN-TYPE

Dynamic Semantics

The value of the call `IsTerminating()` shall be *true* if any coroutine has started program termination and shall be *false* otherwise.

m-IsTerminating : () $\xrightarrow{o}$ $\mathbb{B}$

m-IsTerminating () $\triangleq$
is-program-terminating() 340

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

7.4.2.2 The Function HasHalted

The function `HasHalted` can be used to determine whether a call has been made to `HALT`.

Static Semantics

A call of `HasHalted` shall have no parameters. The type of a call of `HasHalted` shall be the Boolean type.

wf-HasHalted : *Actual-parameters* → *Environment* → $\mathbb{B}$

wf-HasHalted (*aps*)ρ $\triangleq$
  aps = [ ]

t-HasHalted : *Actual-parameters* → *Environment* → *Typed*

t-HasHalted (*aps*)ρ $\triangleq$
  BOOLEAN-TYPE

Dynamic Semantics

The value of the call `HasHalted()` shall be *true* if a call to `HALT` has been made and shall be *false* otherwise.

m-HasHalted : () $\xrightarrow{o}$ $\mathbb{B}$

m-HasHalted () $\triangleq$
  check-halting()

341

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.5 The Module M2EXCEPTION

The system module M2EXCEPTION provides facilities for identifying language exceptions that have been raised.

The language (which includes the system modules) is regarded as one source of exceptions. Similar facilities are defined by library modules that are sources of user-defined exceptions — see, for example, 8.1.

7.5.1 The Interface to M2EXCEPTION

The interface to M2EXCEPTION behaves as if the following were its definition module.

```
DEFINITION MODULE M2EXCEPTION;

(* Provides facilities for identifying language exceptions *)

TYPE
  M2Exceptions =
    (indexException,     rangeException,         caseSelectException,  invalidLocation,
     functionException,  wholeValueException,    wholeDivException,    realValueException,
     realDivException,   complexValueException,  complexDivException,  protException,
     sysException,       coException,            exException
    );

PROCEDURE M2Exception (): M2Exceptions;
  (* If the current coroutine is in the exceptional execution state because of the raising
     of a language exception, returns the corresponding enumeration value, and otherwise
     raises an exception.
  *)

PROCEDURE IsM2Exception (): BOOLEAN;
  (* If the current coroutine is in the exceptional execution state because of the raising
     of a language exception, returns TRUE, and otherwise returns FALSE.
  *)

END M2EXCEPTION.
```

Declaration Semantics

The identifiers available for import from the module M2EXCEPTION shall be the enumeration type identifier M2Exceptions, the corresponding enumeration value identifiers, and the procedure identifiers M2Exception and IsM2Exception.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

values

$\rho_{M2EXCEPTION}$ = let *constants* = {↦},
types = {M2Exceptions ↦ M2EXCEPTIONS-TYPE},
structures = {M2EXCEPTIONS-TYPE ↦ *mk-Enumeration-structure* (*m2exceptions*)},
variables = {↦},
procedures = {M2Exception ↦ M2EXCEPTION-FUNCTION-PROCEDURE,
IsM2Exception ↦ M2EXCEPTION-FUNCTION-PROCEDURE},
modules = {↦},
defmods = {↦},
level = 0,
local-continuations = {↦},
protection-domain = nil ,
protected = { } in
mk-Qualified-environment (*constants*, *types*, *structures*, *variables*, *procedures*, *modules*, *defmods*, *level*, *local-continuations*, *protection-domain*, *protected-procedures*)

7.5.2 The Types of M2EXCEPTION

The module M2EXCEPTION provides the m2exceptions enumeration type (denoted by M2Exceptions) in terms of which the language exceptions are raised. The enumeration values represent aggregations of the exceptions detected by the formal model — see 7.5.6.

Declaration Semantics

The type identifier M2Exceptions shall denote the m2exceptions type which shall be an enumeration type corresponding to the following enumeration:

```
(indexException,     rangeException,        caseSelectException,  invalidLocation,
 functionException,  wholeValueException,   wholeDivException,    realValueException,
 realDivException,   complexValueException, complexDivException,  protException,
 sysException,       coException,           exException
);
```

values

m2exceptions = [indexException, rangeException, caseSelectException, invalidLocation,
functionException, wholeValueException, wholeDivException, realValueException,
realDivException, complexValueException, complexDivException, protException,
sysException, coException, exException]

annotations Values of the m2exceptions type are represented by values of the type *Exception-value* in the formal model — see 6.12.6.

7.5.3 The Functions of M2EXCEPTION

The module M2EXCEPTION provides the function procedures M2Exception and IsM2Exception for enquiries concerning language exceptions.

Abstract Syntax

types

M2EXCEPTION-function-designator :: *ident* : *Identifier*

inv *ident* ≜
ident ∈ {M2Exception, IsM2Exception}

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Static Semantics

wf-M2EXCEPTION-function-designator :
M2EXCEPTION-function-designator → *Actual-parameters* → *Environment* → $\mathbb{B}$

wf-M2EXCEPTION-function-designator $(fd)(aps)\rho \triangleq$
access-environment $(desig)\rho$ = M2EXCEPTION-FUNCTION-PROCEDURE $\wedge$ 319
cases *fd* :
`M2Exception` → *wf-M2Exception* $(aps)\rho$, 394
`IsM2Exception` → *wf-IsM2Exception* $(aps)\rho$ 395
end

t-M2EXCEPTION-function-designator :
M2EXCEPTION-function-designator → *Actual-parameters* → *Environment* → *Typed*

t-M2EXCEPTION-function-designator $(fd)(aps)\rho \triangleq$
cases *fd* :
`M2Exception` → *t-M2Exception* $(aps)\rho$, 394
`IsM2Exception` → *t-IsM2Exception* $(aps)\rho$ 395
end

Dynamic Semantics

m-M2EXCEPTION-function-designator :
M2EXCEPTION-function-designator → *Arguments* → *Environment* $\xrightarrow{o}$ *Value*

m-M2EXCEPTION-function-designator $(fd)(args)\rho \triangleq$
let *handler* = *current-exception-handler* (ρ) in 318
cases *fd*:
`M2Exception` → *m-M2Exception*$(args)$ *handler*, 394
`IsM2Exception` → *m-IsM2Exception*$(args)$ 395
end

7.5.3.1 The Function M2Exception

The function `M2Exception` can be used to determine the language exception that has been raised.

Static Semantics

A call of `M2Exception` shall have no actual parameters. The type of a call of `M2Exception` shall be the m2exceptions type.

wf-M2Exception : *Actual-parameters* → *Environment* → $\mathbb{B}$

wf-M2Exception $(aps)\rho \triangleq$
aps = []

t-M2Exception : *Actual-parameters* → *Environment* → *Typed*

t-M2Exception $(aps)\rho \triangleq$
M2EXCEPTIONS-TYPE

Dynamic Semantics

If the calling coroutine is in the state of exceptional execution because of the raising of a language exception, the value of the call `M2Exception()` shall be the value of type `M2Exceptions` that is associated with the raised exception; otherwise an exception shall be raised.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

m-M2Exception : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Exception-value*

m-M2Exception (*args*)*handler* ≜
  def *context* = *top-exception-context*() in 342
  def *exvalue* =
    cases *context* :
      NORMAL-EXECUTION → *mandatory-detection*(NO-CURRENT-EXCEPTION) *handler*, 396
      mk-User-exception (-, -, -) → *mandatory-detection*(NOT-CURRENT-SOURCE) *handler*, 396
      mk-Language-exception (*ex*, -) → *ex*
    end in
  return *exvalue*

7.5.3.2 The Function IsM2Exception

The function `IsM2Exception` can be used to test whether the calling coroutine is in the state of exceptional execution because of the raising of a language exception.

Static Semantics

A call of `IsM2Exception` shall have no parameters. The type of a call of `IsM2Exception` shall be the Boolean type.

wf-IsM2Exception : *Actual-parameters* → *Environment* → 𝔹

wf-IsM2Exception (*aps*)ρ ≜
  len *aps* = 0

t-IsM2Exception : *Actual-parameters* → *Environment* → *Typed*

t-IsM2Exception (*aps*)ρ ≜
  BOOLEAN-TYPE

Dynamic Semantics

If the calling coroutine is in the state of exceptional execution because of the raising of a language exception, the value of the call `IsM2Exception()` shall be *true*; otherwise the value shall be *false*.

m-IsM2Exception : *Arguments* $\xrightarrow{o}$ 𝔹

m-IsM2Exception (*aps*) ≜
  def *context* = *top-exception-context*() in 342
  let *res* = cases *context* :
      NORMAL-EXECUTION → false,
      mk-User-exception (-, -, -) → false,
      mk-Language-exception (-, -) → true
    end in
  return *res*

7.5.4 Language Exceptions

The exceptions defined by the dynamic semantics of the language, which includes the system modules, are categorized into those which an implementation is required to detect and those which an implementation is not required to detect. Detected exceptions are raised by the implementation and constitute a termination event (see 6.1.7) unless the exception is handled by an exception body (see 6.5.8). Exceptions that are distinguished in the dynamic semantics are aggregated into a smaller number of identified exceptions for the purposes of exception handling.

An implementation must define which of the exceptions whose detection is not required are in fact detected in some or all circumstances. It need not define the circumstances in which such exceptions are detected. If such an exception

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

occurs, but is not detected, this part of ISO/IEC 10514 gives no meaning to the program — see 4.7.

Domains

types

Language-exceptions = *Detection-required* | *Detection-not-required*

7.5.4.1 Exceptions whose detection is required

Domains

types

Detection-required =

ADDRESS-ARITHMETIC-ON-NIL	— call of ADDADR, SUBADR, or DIFADR on NIL
| ASSIGN-RANGE	— assignment value out of range of target
| CASE-RANGE	— case selector out of range
| COMPLEX-OVERFLOW	— overflow in the evaluation of a COMPLEX expression
| COMPLEX-ZERO-DIVISION	— divide by zero in a COMPLEX expression
| CONSTRUCTOR-RANGE	— structure component value is out of range
| CONVERSIONERROR	— expression cannot be converted to the new type
| EXCLNOTBASE	— value to be excluded is not of the base type
| INCLNOTBASE	— value to be included is not of the base type
| INDEX-RANGE	— array indexed out of range
| LESSPROTECTION	— given protection less restrictive than current protection
| NIL-DEREFERENCE	— attempt to dereference NIL
| NO-CURRENT-EXCEPTION	— exception identity enquiry in normal execution
| NO-EXCEPTION-SOURCE	— no further exception source values can be allocated
| NORETURN	— no return statement before the end of a function
| NOT-ATTACHED	— the caller is not attached to a source of interrupts
| NOT-CURRENT-SOURCE	— exception identity enquiry to wrong source
| REAL-OVERFLOW	— overflow in the evaluation of a REAL expression
| REAL-ZERO-DIVISION	— divide by zero in the evaluation of a REAL expression
| RETURN-FROM-COROUTINE	— return from a coroutine other than the main coroutine
| RETURN-RANGE	— return value out of range
| SET-RANGE	— set value out of range
| SMALL-WORKSPACE	— size of the supplied workspace smaller than the minimum required
| TAG-RANGE	— tag value out of range
| WHOLE-NONPOS-DIV	— 2nd operand of DIV is negative or zero
| WHOLE-NONPOS-MOD	— 2nd operand of MOD is negative or zero
| WHOLE-ZERO-DIVISION	— divide by zero in the evaluation of a whole number expression
| WHOLE-ZERO-REMAINDER	— 2nd operand of REM is zero
| WORKSPACE-OVERFLOW	— coroutine workspace overflow

Dynamic Semantics

mandatory-detection : *Detection-required* → *Exception-handler* $\xrightarrow{o}$ ()

mandatory-detection (*exception*)*handler* $\triangleq$
  def *message* = *implementation-defined-exception-string*(*exception*) in 397
  build-and-raise-exception(*exception*, *message*) *handler* 398

annotations Construct an appropriate message based on the exception and the current state, and construct an exception context in which to raise the exception.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

7.5.4.2 Exceptions whose detection is not required

Domains

types

Detection-not-required =

ADDRESS-ARITHMETIC	— invalid use of DIFADR, SUBADR or ADDADR
| ADDRESS-CONSTRUCTION	— result of MAKEADR out of address range
| INACTIVE-VARIANT	— access to non-active variant
| INVALID-ALIGNMENT	— alignment problem with CAST
| INVALID-VALUE	— result of CAST not valid representation for target type
| NO-HANDLER	— source for HANDLER is not associated with a coroutine
| NONEXISTENT	— attempt to access a non-existing variable
| STORAGE-NOT-OWNED	— attempt to access storage of another coroutine
| UNDEFINED-VALUE	— undefined value
| WHOLE-OVERFLOW	— overflow in the evaluation of a whole number expression
| WRONG-TYPE	— a pointer points to the wrong type of variable

Dynamic Semantics

non-mandatory-detection : *Detection-not-required* → *Exception-handler* $\xrightarrow{o}$ ()

non-mandatory-detection (*exception*)*handler* $\triangleq$
  if *actually-detected* (*exception*) 397
  then def *message* = *implementation-defined-exception-string*(*exception*) in 397
    build-and-raise-exception(*exception*, *message*) *handler* 398
  else implementation-dependent

annotations If the exception is detected by the implementation, construct an appropriate message based on the exception, the environment and the current state, and construct an exception context in which to raise the exception. Otherwise, the behaviour is implementation-dependent.

actually-detected : *Detection-not-required* → $\mathbb{B}$

actually-detected (*exception*) $\triangleq$
  implementation-dependent

annotations The result is *true* if the exception is actually detected and otherwise *false*.

7.5.5 Messages Associated with Language Exceptions

Dynamic Semantics

The character string associated with each of the exceptions that may be raised as specified in sections 6 and 7 shall be implementation-defined.

NOTE — The character string associated with a raised exception may be accessed using the `GetMessage` procedure — see 7.3.3.3.

implementation-defined-exception-string : *Exception* $\xrightarrow{o}$ *Message*

implementation-defined-exception-string (*exception*) $\triangleq$
  implementation-defined

annotations Using the exception and the current state, construct an appropriate message for a language exception context.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

7.5.6 Aggregation and Raising of Language Exceptions

values

exception-value = {INDEX-RANGE ↦ *mk-Exception-value* (`indexException`, *m2exceptions*),
ASSIGN-RANGE ↦ *mk-Exception-value* (`rangeException`, *m2exceptions*),
CONSTRUCTOR-RANGE ↦ *mk-Exception-value* (`rangeException`, *m2exceptions*),
CONVERSIONERROR ↦ *mk-Exception-value* (`rangeException`, *m2exceptions*),
EXCLNOTBASE ↦ *mk-Exception-value* (`rangeException`, *m2exceptions*),
INCLNOTBASE ↦ *mk-Exception-value* (`rangeException`, *m2exceptions*),
RETURN-RANGE ↦ *mk-Exception-value* (`rangeException`, *m2exceptions*),
SET-RANGE ↦ *mk-Exception-value* (`rangeException`, *m2exceptions*),
TAG-RANGE ↦ *mk-Exception-value* (`rangeException`, *m2exceptions*),
CASE-RANGE ↦ *mk-Exception-value* (`caseSelectException`, *m2exceptions*),
ADDRESS-ARITHMETIC ↦ *mk-Exception-value* (`invalidLocation`, *m2exceptions*),
INACTIVE-VARIANT ↦ *mk-Exception-value* (`invalidLocation`, *m2exceptions*),
NIL-DEREFERENCE ↦ *mk-Exception-value* (`invalidLocation`, *m2exceptions*),
NONEXISTENT ↦ *mk-Exception-value* (`invalidLocation`, *m2exceptions*),
STORAGE-NOT-OWNED ↦ *mk-Exception-value* (`invalidLocation`, *m2exceptions*),
UNDEFINED-VALUE ↦ *mk-Exception-value* (`invalidLocation`, *m2exceptions*),
WRONG-TYPE ↦ *mk-Exception-value* (`invalidLocation`, *m2exceptions*)
NORETURN ↦ *mk-Exception-value* (`functionException`, *m2exceptions*),
WHOLE-OVERFLOW ↦ *mk-Exception-value* (`wholeValueException`, *m2exceptions*),
WHOLE-NONPOS-DIV ↦ *mk-Exception-value* (`wholeDivException`, *m2exceptions*),
WHOLE-NONPOS-MOD ↦ *mk-Exception-value* (`wholeDivException`, *m2exceptions*),
WHOLE-ZERO-DIVISION ↦ *mk-Exception-value* (`wholeDivException`, *m2exceptions*),
WHOLE-ZERO-REMAINDER ↦ *mk-Exception-value* (`wholeDivException`, *m2exceptions*),
LOWLONGEXCEPTION ↦ *mk-Exception-value* (`realValueException`, *m2exceptions*),
LOWREALEXCEPTION ↦ *mk-Exception-value* (`realValueException`, *m2exceptions*),
REAL-OVERFLOW ↦ *mk-Exception-value* (`realValueException`, *m2exceptions*),
REAL-ZERO-DIVISION ↦ *mk-Exception-value* (`realDivException`, *m2exceptions*),
COMPLEX-OVERFLOW ↦ *mk-Exception-value* (`complexValueException`, *m2exceptions*),
COMPLEX-ZERO-DIVISION ↦ *mk-Exception-value* (`complexDivException`, *m2exceptions*),
LESSPROTECTION ↦ *mk-Exception-value* (`protException`, *m2exceptions*),
PROTECTIONERROR ↦ *mk-Exception-value* (`protException`, *m2exceptions*),
ADDRESS-ARITHMETIC-ON-NIL ↦ *mk-Exception-value* (`sysException`, *m2exceptions*),
ADDRESS-CONSTRUCTION ↦ *mk-Exception-value* (`sysException`, *m2exceptions*),
INVALID-ALIGNMENT ↦ *mk-Exception-value* (`sysException`, *m2exceptions*),
INVALID-VALUE ↦ *mk-Exception-value* (`sysException`, *m2exceptions*),
COROUTINE-EXCEPTION ↦ *mk-Exception-value* (`coException`, *m2exceptions*),
NOT-ATTACHED ↦ *mk-Exception-value* (`coException`, *m2exceptions*),
SMALL-WORKSPACE ↦ *mk-Exception-value* (`coException`, *m2exceptions*),
WORKSPACE-OVERFLOW ↦ *mk-Exception-value* (`coException`, *m2exceptions*)
NO-CURRENT-EXCEPTION ↦ *mk-Exception-value* (`exException`, *m2exceptions*),
NO-EXCEPTION-SOURCE ↦ *mk-Exception-value* (`exException`, *m2exceptions*),
NOT-CURRENT-SOURCE ↦ *mk-Exception-value* (`exException`, *m2exceptions*)}

build-and-raise-exception : *Exception* × *Message* → *Exception-handler* $\xrightarrow{o}$ ()

build-and-raise-exception (*exception*, *message*)*handler* $\triangleq$
let *except-value* = *exception-value*(*exception*) in
raise-exception(*mk-Language-exception* (*except-value*, *message*)) *handler* 342

annotations Build a language exception context from: an enumeration value defined by the mapping *exception-value*; and the message. Raise an exception that has the exception context.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Section 8 : Required Library Modules

An implementation is required to provide the separate modules `Storage`, `LowReal`, `LowLong`, and `CharClass` — see 4.5.

8.1 The Module Storage

The module `Storage` provides facilities for dynamically allocating and deallocating storage for variables that are not declared in variable declarations. Variables with storage allocated in this way are designated by dereferenced variable designators — see 6.7.4. The facilities are often invoked by using the predefined procedures `NEW` and `DISPOSE` — see 6.10.3.8 and 6.10.3.3. The facilities can also be used to allocate storage to be used as coroutine workspace — see 7.2.

The semantics are described in terms of allocating storage for a variable since the allocator must take account of any address alignment requirements for the storage of variables of the given size.

8.1.1 The Definition Module of Storage

```
DEFINITION MODULE Storage;

  (* Facilities for dynamically allocating and deallocating storage *)

IMPORT SYSTEM;

PROCEDURE ALLOCATE (VAR addr: SYSTEM.ADDRESS; amount: CARDINAL);
  (* Allocates storage for a variable of size amount and assigns the address of this
     variable to addr. If there is insufficient unallocated storage to do this, the
     value NIL is assigned to addr.
  *)

PROCEDURE DEALLOCATE (VAR addr: SYSTEM.ADDRESS; amount: CARDINAL);
  (* Deallocates amount locations allocated by ALLOCATE for the storage of the variable
     addressed by addr and assigns the value NIL to addr.
  *)

TYPE
  StorageExceptions = (
    nilDeallocation,             (* first argument to DEALLOCATE is NIL *)
    pointerToUnallocatedStorage, (* storage to deallocate not allocated by ALLOCATE *)
    wrongStorageToUnallocate     (* amount to deallocate is not amount allocated *)
  );

PROCEDURE IsStorageException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional execution state
     because of the raising of an exception from StorageExceptions;
     otherwise returns FALSE.
  *)

PROCEDURE StorageException (): StorageExceptions;
  (* If the current coroutine is in the exceptional execution state because of the
     raising of an exception from StorageExceptions, returns the corresponding
     enumeration value, and otherwise raises an exception.
  *)

END Storage.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

CLARIFICATIONS

1 *Programming in Modula-2* adopts two approaches to handling situations where it is not possible to allocate sufficient storage; the procedure `Allocate` of Chapter 25 assigns `NIL` to the first parameter, whereas the procedure `ALLOCATE` of Appendix 2 causes the program to terminate. This part of ISO/IEC 10514 requires the procedure `ALLOCATE` to assign the value `NIL` to its first parameter in this situation.

2 Although the first parameter of the procedure `DEALLOCATE` given in Appendix 2 of *Programming in Modula-2* is a variable parameter, it is not stated whether `DEALLOCATE` assigns a value to its parameter. This part of ISO/IEC 10514 requires the procedure `DEALLOCATE` to assign the value `NIL` to its parameter.

8.1.2 The Procedures of Storage

The procedure `ALLOCATE`

The call `ALLOCATE(addr,amount)` shall allocate storage for a variable of size `amount`, and shall assign the address of this variable to `addr`. The allocated locations shall not be in use for the storage of any other variable. If it is not possible to allocate this storage, the nil value shall be assigned to `addr`.

NOTES

1 If an address passed back by a call of `ALLOCATE` is assigned to a pointer variable `vp` that is to be dereferenced to designate a variable of type `T`, the value for the second parameter to `ALLOCATE` may be obtained from evaluation of the expression `SIZE(T)`. An equivalent effect may be obtained as `NEW(vp)`.

2 If an address passed back by a call of `ALLOCATE` is to be given directly as the workspace address in a call of `COROUTINES.NEWCOROUTINE`, the value for the second parameter to `ALLOCATE` is the size of workspace required.

ALLOCATE : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

ALLOCATE (*args*)*handler* $\triangleq$
  let [*addr*, *amount*] = *args* in
  def *available* = *is-available*(*amount*) in 400
  if (*amount* = 0) ∨ (¬ *available*)
  then *assign*(*addr*, NIL-VALUE) *handler* 150
  else let *mk-Pointer-variable* (*loc*, *ptype*) = *addr* in
    def *var* = *allocate-heap-storage*(*ptype*) in 403
    let *aval* = *mk-Address-value* (*var*, *ptype*, *amount*) in
    assign(*addr*, *aval*) *handler* 150

annotations The *addr* argument to this procedure is modelled as a typed pointer. If enough storage is available, storage of the type of the pointer is allocated and the address of the storage is returned.

is-available : $\mathbb{N} \xrightarrow{o} \mathbb{B}$

is-available (*amount*) $\triangleq$
  implementation-dependent

The procedure `DEALLOCATE`

The call `DEALLOCATE(addr,amount)` shall deallocate `amount` locations for the storage of the variable addressed by addr and assign the nil value to addr.

The exception `nilDeallocation` shall be raised if the given value of `addr` is the nil value. The exception `pointerToUnallocatedStorage` shall be raised if the given value of `addr` is not the address of a variable whose storage has been allocated by `ALLOCATE`. The exception `wrongStorageToUnallocate` shall occur and may be raised if `amount` is not equal to the number of locations allocated for the storage of the variable addressed by `addr`.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NOTES

3 If an address passed to a call of **DEALLOCATE** is the value of a pointer variable vp that is dereferenced to designate a variable of type T, the value for the second parameter to **DEALLOCATE** may be obtained from evaluation of the expression **SIZE(T)**. An equivalent effect may be obtained as **DISPOSE(vp)**.

4 The variable whose storage is deallocated no longer exists and hence an exception occurs, which may be raised, if there is a subsequent attempt to access the variable through a dereferenced designator.

5 This part of ISO/IEC 10514 gives no meaning for a program that deallocates dynamic storage given as workspace in a call of **COROUTINES.NEWCOROUTINE** since the use made of coroutine workspace is implementation-dependent — see 7.2.3.5.

6 It need not be the case that storage locations deallocated by a call of **DEALLOCATE** are available for re-use by a subsequent call of **ALLOCATE**.

DEALLOCATE : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

DEALLOCATE (*args*)*handler* $\triangleq$
  let [*addr*, *amount*] = *args* in
  def *pval* = *value-of-a-variable*(*addr*) in 224
  if *pval* = NIL-VALUE
  then *storage-exception*(*nilDeallocation*) *handler* 402
  elseif ¬ *in-heap* (*loc*, *storage-heap*) 403
  then *storage-exception*(*pointerToUnallocatedStorage*) *handler* 402
  else let *mk-Address-value* (*avar*, *atype*, *size*) = *loc* in
    if *size* ≠ *amount*
    then *storage-exception*(*wrongStorageToUnallocate*) *handler* 402
    else (*deallocate-heap-storage*(*avar*) ; 403
      assign(*addr*, NIL-VALUE) *handler*) 150

The procedure IsStorageException

If the calling coroutine is in the state of exceptional execution because of the raising of a **Storage** exception, the value of the call **IsStorageException()** shall be *true*; otherwise the value shall be *false*.

IsStorageException : *Arguments* → *Exception-handler* $\xrightarrow{o}$ $\mathbb{B}$

IsStorageException (*args*)*handler* $\triangleq$
  def *source* = *storage-source* in
  def *res* = *m-IsCurrentSource*([*source*]) in 386
  return *res*

The procedure StorageException

If the calling coroutine is in the state of exceptional execution because of the raising of a **Storage** exception, the value of the call **StorageException()** shall be the value that identifies the raised exception; otherwise the language exception **exException** shall be raised.

StorageException : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Value*

StorageException (*args*)*handler* $\triangleq$
  def *source* = *storage-source* in
  def *n* = *m-CurrentNumber*([*source*]) *handler* in 386
  let *res* = *val-enumeration* (*n*, *storageexceptions*) in 330
  return *res*

annotations The language exception **exException** is raised by *m-CurrentNumber* if *storage-source* is not the current source or there is no current exception.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8.1.3 Exceptions Raised by Storage

The exceptions raised by `Storage` shall be identified by the values of the enumeration type `StorageExceptions`.

The detection of the exception `wrongStorageToUnallocate` shall be implementation-defined.

The character string associated with each of the exceptions raised by `Storage` shall be implementation-defined.

values

storageexceptions = [`nilDeallocation`, `pointerToUnallocatedStorage`, `wrongStorageToUnallocate`];

nilDeallocation = *mk-Enumeration-value* (`nilDeallocation`, *storageexceptions*);

pointerToUnallocatedStorage = *mk-Enumeration-value* (`pointerToUnallocatedStorage`, *storageexceptions*);

wrongStorageToUnallocate = *mk-Enumeration-value* (`wrongStorageToUnallocate`, *storageexceptions*)

storage-exception : *Enumeration-value* → *Exception-handler* $\xrightarrow{o}$ ()

storage-exception (*se*)*handler* $\triangleq$
  if *storage-exception-detected* (*se*)
  then *raise-storage-exception*(*se*) *handler* 402
  else implementation-dependent actions 402

storage-exception-detected : *Enumeration-value* → $\mathbb{B}$

storage-exception-detected (*se*) $\triangleq$
  cases *se* :
    nilDeallocation → true,
    pointerToUnallocatedStorage → true,
    wrongStorageToUnallocate → implementation-defined
  end

raise-storage-exception : *Enumeration-value* → *Exception-handler* $\xrightarrow{o}$ ()

raise-storage-exception (*se*)*handler* $\triangleq$
  def *source* = *storage-source* in
  def *message* = *storage-exception-string*(*se*) in 402
  let *number* ∈ inds *storageexceptions* be st
      se = *val-enumeration* (*number*, *storageexceptions*) in 330
  m-RAISE([*source*, *number*, *message*]) *handler* 383

storage-exception-string : *Enumeration-value* $\xrightarrow{o}$ *Message*

storage-exception-string (*se*) $\triangleq$
  implementation-defined

annotations The result is an implementation-defined string value.

8.1.4 The State for Storage

The state for the module `Storage` shall include the set of locations available to be allocated by `ALLOCATE` and a value of the exception source type identifying exceptions raised by the module.

NOTE — An implementation of the module `Storage` is free to augment the set of locations available to be allocated at any time, for example by calls to an underlying operating system.

values

HEAP-STORAGE-POOL : *Storage-pool* = implementation-dependent

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations *HEAP-STORAGE-POOL* is an implementation-dependent value consisting of a set of *Locs* that denotes the initial free storage available for allocation by the **ALLOCATE** procedure.

```
state Storage of
  storage-heap : Store
  storage-source : Exception-source

  init mk-Storage (heap, source) △
      heap = mk-Store (HEAP-STORAGE-POOL, {↦})
end
```

8.1.5 The Initialization of Storage

The initialization of the module **Storage** shall allocate a unique value of the exception source type to identify exceptions raised by the module.

```
storage-initialization : () → Exception-handler →o ()
storage-initialization ()handler △
  def source = allocate-source() in                                    342
  if source ≠ nil
  then storage-source := source
  else mandatory-detection(NO-EXCEPTION-SOURCE) handler                396
```

8.1.6 Auxiliary Operations

```
allocate-heap-storage : Ptype-constraint →o Variable
allocate-heap-storage (mk-Ptype-constraint (type, ρ)) △
  (def store = get-heap-storage() in                                   403
   let mk-(var, nstore) = a-typed (type, store)ρ in                    114
   put-heap-storage(nstore) ;                                          403
   return var )
```

annotations Allocate storage of a form that matches the type of the parameter and return the address of the storage.

```
deallocate-heap-storage : Variable →o ()
deallocate-heap-storage (var) △
  (let locs = locs-of (avar) in                                        338
   def store = get-heap-storage() in                                   403
   let nstore = deallocate-locs (locs, store) in                       338
   put-heap-storage(nstore) )                                          403
```

annotations Deallocate the storage associated with an address.

```
get-heap-storage : () →o Store
get-heap-storage () △
  def st = storage-heap in
  return st
```

annotations Return the value of *storage-heap*.

```
put-heap-storage : () →o Store
put-heap-storage (st) △
  storage-heap := st
```

annotations Change the storage in the heap — this will either allocate or deallocate storage.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

$in\text{-}heap : Variable \times Store \to \mathbb{B}$

$in\text{-}heap\,(var, store) \triangleq$
  let $locs = locs\text{-}of\,(var)$ in 338
  let $mk\text{-}Store\,(pool, values) = store$ in
  $locs \in pool\text{-}locs\,(pool) \wedge$ 338
  $locs \in \mathsf{dom}\ values$

annotations Check that the storage associated with an address is in the heap and allocated.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2 The Modules LowReal and LowLong

Two modules are provided to give access to the underlying properties of the types REAL and LONGREAL. The two modules share common concepts, functions and values, and hence both modules are considered together.

The module LowReal gives access to the underlying properties of the type REAL, while LowLong gives access to the same properties for the type LONGREAL.

Since real number types may be represented in fixed-point, floating-point, or some other form, the semantics of these modules cannot be defined to the precision of the main language. Hence VDM-SL is not used in this clause.

For implementations that conform to ISO/IEC 10967-1, a more precise specification is given in that International Standard.

8.2.1 The Definition Module of LowReal

```
DEFINITION MODULE LowReal;

  (* Access to underlying properties of the type REAL *)

CONST
  radix =      <implementation-defined whole number value>;
  places =     <implementation-defined whole number value>;
  expoMin =    <implementation-defined whole number value>;
  expoMax =    <implementation-defined whole number value>;
  large =      <implementation-defined real number value>;
  small =      <implementation-defined real number value>;
  IEC559 =     <implementation-defined BOOLEAN value>;
  LIA1 =       <implementation-defined BOOLEAN value>;
  rounds =     <implementation-defined BOOLEAN value>;
  gUnderflow = <implementation-defined BOOLEAN value>;
  exception =  <implementation-defined BOOLEAN value>;
  extend =     <implementation-defined BOOLEAN value>;
  nModes =     <implementation-defined whole number value>;

TYPE
  Modes = PACKEDSET OF [0 .. nModes-1];

PROCEDURE exponent (x: REAL): INTEGER;
  (* Returns the exponent value of x *)

PROCEDURE fraction (x: REAL): REAL;
  (* Returns the significand (or significant part) of x *)

PROCEDURE sign (x: REAL): REAL;
  (* Returns the signum of x *)

PROCEDURE succ (x: REAL): REAL;
  (* Returns the next value of the type REAL greater than x *)

PROCEDURE ulp (x: REAL): REAL;
  (* Returns the value of a unit in the last place of x *)

PROCEDURE pred (x: REAL): REAL;
  (* Returns the previous value of the type REAL less than x *)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
PROCEDURE intpart (x: REAL): REAL;
  (* Returns the integer part of x *)

PROCEDURE fractpart (x: REAL): REAL;
  (* Returns the fractional part of x *)

PROCEDURE scale (x: REAL; n: INTEGER): REAL;
  (* Returns the value of x * radix ** n *)

PROCEDURE trunc (x: REAL; n: INTEGER): REAL;
  (* Returns the value of the first n places of x *)

PROCEDURE round (x: REAL; n: INTEGER): REAL;
  (* Returns the value of x rounded to the first n places *)

PROCEDURE synthesize (expart: INTEGER; frapart: REAL): REAL;
  (* Returns a value of the type REAL constructed from the given expart and frapart *)

PROCEDURE setMode (m: Modes);
  (* Sets status flags appropriate to the underlying implementation of the type REAL *)

PROCEDURE currentMode (): Modes;
  (* Returns the current status flags in the form set by setMode *)

PROCEDURE IsLowException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional execution state
     because of the raising of the LowReal exception; otherwise returns FALSE.
  *)

END LowReal.
```

8.2.2 The Definition Module of LowLong

```
DEFINITION MODULE LowLong;

  (* Access to underlying properties of the type LONGREAL *)

CONST
  radix =      <implementation-defined whole number value>;
  places =     <implementation-defined whole number value>;
  expoMin =    <implementation-defined whole number value>;
  expoMax =    <implementation-defined whole number value>;
  large =      <implementation-defined real number value>;
  small =      <implementation-defined real number value>;
  IEC559 =     <implementation-defined BOOLEAN value>;
  LIA1 =       <implementation-defined BOOLEAN value>;
  rounds =     <implementation-defined BOOLEAN value>;
  gUnderflow = <implementation-defined BOOLEAN value>;
  exception =  <implementation-defined BOOLEAN value>;
  extend =     <implementation-defined BOOLEAN value>;
  nModes =     <implementation-defined whole number value>;

TYPE
  Modes = PACKEDSET OF [0 .. nModes-1];

PROCEDURE exponent (x: LONGREAL): INTEGER;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  (* Returns the exponent value of x *)

PROCEDURE fraction (x: LONGREAL): LONGREAL;
  (* Returns the significand (or significant part) of x *)

PROCEDURE sign (x: LONGREAL): LONGREAL;
  (* Returns the signum of x *)

PROCEDURE succ (x: LONGREAL): LONGREAL;
  (* Returns the next value of the type LONGREAL greater than x *)

PROCEDURE ulp (x: LONGREAL): LONGREAL;
  (* Returns the value of a unit in the last place of x *)

PROCEDURE pred (x: LONGREAL): LONGREAL;
  (* Returns the previous value of the type LONGREAL less than x *)

PROCEDURE intpart (x: LONGREAL): LONGREAL;
  (* Returns the integer part of x *)

PROCEDURE fractpart (x: LONGREAL): LONGREAL;
  (* Returns the fractional part of x *)

PROCEDURE scale (x: LONGREAL; n: INTEGER): LONGREAL;
  (* Returns the value of x * radix ** n *)

PROCEDURE trunc (x: LONGREAL; n: INTEGER): LONGREAL;
  (* Returns the value of the first n places of x *)

PROCEDURE round (x: LONGREAL; n: INTEGER): LONGREAL;
  (* Returns the value of x rounded to the first n places *)

PROCEDURE synthesize (expart: INTEGER; frapart: LONGREAL): LONGREAL;
  (* Returns a value of the type LONGREAL constructed from the given expart and frapart *)

PROCEDURE setMode (m: Modes);
  (* Sets status flags appropriate to the underlying implementation of the type LONGREAL *)

PROCEDURE currentMode (): Modes;
  (* Returns the current status flags in the form set by setMode *)

PROCEDURE IsLowException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional execution state
     because of the raising of the LowLong exception; otherwise returns FALSE.
  *)

END LowLong.
```

8.2.3 The Constants of LowReal and LowLong

The values denoted by the constant identifiers exported from `LowReal` and `LowLong` shall be the implementation-defined values specified below.

If an implementation provides facilities for dynamically changing the properties of the real number types, the constant values shall refer to the default properties.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The value of `places`, and the other facilities in these modules, shall refer only to the representation used to store values.

NOTE 1 — Some implementations may choose to compute expressions to greater precision than that used to store values.

If the implementation of the corresponding real number type conforms to ISO/IEC 10967-1, the following correspondences shall hold:

Modula-2	LIA-1
`radix`	r
`places`	p
`gUnderflow`	*denorm*
`small`	$fmin_N$
`expoMin`	*emin*
`expoMax`	*emax*

NOTE 2 — The value of the parameter *fmax*, required by ISO/IEC 10967-1, is given by the predefined function `MAX` when applied to the corresponding real number type.

`radix`: — The whole number value of the radix used to represent the corresponding real number values.

`places`: — The whole number value of the number of radix places used to store values of the corresponding real number type.

`expoMin`: — The whole number value of the exponent minimum.

`expoMax`: — The whole number value of the exponent maximum.

NOTE 3 — An implementation may choose values such that `expoMin` = `expoMax` (which will presumably be the case for a fixed point representation).

`large`: — The largest value of the corresponding real number type.

NOTE 4 — On some implementations this may be a machine representation of infinity.

`small`: — The smallest positive value of the corresponding real number type, represented to maximal precision.

NOTE 5 — If an implementation has stored values strictly between 0.0 and `small`, then presumably the implementation supports gradual underflow.

`IEC559`: — A Boolean value that is *true* if and only if the implementation of the corresponding real number type conforms to IEC 559:1989 (also known as IEEE 754:1987) in all regards.

NOTES

6 If `IEC559` is *true*, the value of `radix` is 2.

7 If `LowReal.IEC559` is *true*, the 32-bit format of IEC 559:1989 is used for the type `REAL`.

8 If `LowLong.IEC559` is *true*, the 64-bit format of IEC 559:1989 is used for the type `LONGREAL`.

`LIA1`: — A Boolean value that is *true* if and only if the implementation of the corresponding real number type conforms to ISO/IEC 10967-1 in all regards: parameters, arithmetic, exceptions, and notification.

NOTE 9 — For implementations not conforming to ISO/IEC 10967-1, the corresponding properties are implementation-defined — see 6.8.2.2.

`rounds`: — A Boolean value that is *true* if and only if each operation produces a result that is one of the values of the corresponding real number type nearest to the mathematical result.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NOTE 10 — If `rounds` is *true*, and the mathematical result lies mid-way between two values of the corresponding real number type, then the selection from the two possible values is implementation-dependent.

`gUnderflow`: — A Boolean value that is *true* if and only if there are values of the corresponding real number type between 0.0 and `small`.

`exception`: — A Boolean value that is *true* if and only if every operation that attempts to produce a real value out of range raises an exception.

`extend`: — A Boolean value that is *true* if and only if expressions of the corresponding real number type are computed to higher precision than the stored values.

NOTE 11 — If `extend` is *true*, then values greater than `large` can be computed in expressions, but cannot be stored in variables.

`nModes`: — The whole number value giving the number of bit positions needed for the status flags for mode control.

8.2.4 The Procedures of LowReal and LowLong

If the implementation of the corresponding real number type conforms to ISO/IEC 10967-1, functions of a similar name shall correspond to the operations required by that International Standard.

The procedure `exponent`

The value of the call `exponent(x)` shall be the exponent value of `x` that lies between `expoMin` and `expoMax`. An exception shall occur and may be raised if `x` is equal to 0.0.

The procedure `fraction`

The value of the call `fraction(x)` shall be the significand (or significant) part of `x`. Hence the following relationship shall hold: `x = scale(fraction(x), exponent(x))`.

The procedure `sign`

The value of the call `sign(x)` shall be 1.0 if `x` is greater than 0.0, or shall be −1.0 if `x` is less than 0.0, or shall be either 1.0 or −1.0 if `x` is equal to 0.0.

NOTE 1 — The uncertainty about the handling of 0.0 is to allow for systems that distinguish between +0.0 and −0.0 (such as IEEE 754 systems).

The procedure `succ`

The value of the call `succ(x)` shall be the next value of the corresponding real number type greater than `x`, if such a value exists; otherwise an exception shall occur and may be raised.

The procedure `ulp`

The value of the call `ulp(x)` shall be the value of the corresponding real number type equal to a unit in the last place of `x`, if such a value exists; otherwise an exception shall occur and may be raised.

NOTE 2 — Thus, when the value exists, either `ulp(x) = succ(x)-x` or `ulp(x) = x-pred(x)` or both shall be true.

The procedure `pred`

The value of the call `pred(x)` shall be the next value of the corresponding real number type less than `x`, if such a value exists; otherwise an exception occurs and may be raised.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The procedure intpart

The value of the call intpart(x) shall be the integral part of x. For negative values, this shall be -intpart(abs(x)).

The procedure fractpart

The value of the call fractpart(x) shall be the fractional part of x. This satisfies the relationship fractpart(x)+intpart(x)=x.

The procedure scale

The value of the call scale(x,n) shall be the value $x * radix^n$ if such a value exists; otherwise an exception shall occur and may be raised.

The procedure trunc

The value of the call trunc(x, n) shall be the value of the most significant n places of x. An exception shall occur and may be raised if n is less than or equal to zero.

The procedure round

The value of the call round(x,n) shall be the value of x rounded to the most significant n places. An exception shall occur and may be raised if such a value does not exist, or if n is less than or equal to zero.

The procedure synthesize

The value of the call synthesize(expart,frapart) shall be a value of the corresponding real number type constructed from the values of expart and frapart. This value shall satisfy the relationship synthesize(exponent(x),fraction(x))=x.

The procedure setMode

The call setMode(m) shall set status flags from the value of m, appropriate to the underlying implementation of the corresponding real number type.

NOTES

3 Many implementations of floating point provide options for setting status flags within the system which control details of the handling of the type. Although two procedures are provided, one for each real number type, the effect may be the same. Typical effects that can be obtained by this means are:

a) Ensuring that overflow will raise an exception;

b) Allowing underflow to raise an exception;

c) Controlling the rounding;

d) Allowing special values to be produced (e.g. NaNs in implementations conforming to IEC 559:1989 (IEEE 754:1987));

e) Ensuring that special value access will raise an exception;

Since these effects are so varied, the values of type Modes that may be used are not specified by this part of ISO/IEC 10514.

4 The effect of setMode on operations on values of the corresponding real number type in coroutines other than the calling coroutine is not defined. Implementations are not required to preserve the status flags (if any) with the coroutine state.

The procedure currentMode

The value of the call currentMode() shall be the current status flags (in the form set by setMode), or the default status flags (if setMode is not used).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NOTE 5 — The value of the call `currentMode()` is not necessarily the value set by `setMode`, since a call of `setMode` might attempt to set flags that cannot be set by the program.

The procedure `IsLowException`

If the calling coroutine is in the state of exceptional execution because of the raising of the `LowReal` exception, the value of the call `LowReal.IsLowException()` shall be *true*; otherwise the value shall be *false*.

If the calling coroutine is in the state of exceptional execution because of the raising of the `LowLong` exception, the value of the call `LowLong.IsLowException()` shall be *true*; otherwise the value shall be *false*.

8.2.5 Exceptions Raised by LowReal and LowLong

Following the detection of one of the exceptions defined for the modules `LowReal` and `LowLong`, an implementation-defined exception shall be raised which shall either be the `LowReal` exception (for the module `LowReal`) or the `LowLong` exception (for the module `LowLong`), or an implementation-defined language exception.

NOTES

1 The language exceptions are listed in 7.5.4; the exception could depend upon the context.

2 If an implementation chooses to rely on a language exception, this exception could fail to be detected (see the list in 7.5.4).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.3 The Module CharClass

The full set of values of the character type (the elementary type denoted by the pervasive identifier CHAR) is implementation-defined. The module CharClass allows a program to determine the classification of a given value of the character type of an implementation in a way that does not rely on there being a known literal representation of all members of the character set.

The module CharClass provides predicates to test if a given value of the character type in an implementation is classified as a numeric, a letter, an upper-case letter, a lower-case letter, a value representing a control function, or as white space.

8.3.1 The Definition Module of CharClass

```
DEFINITION MODULE CharClass;

  (* Classification of values of the type CHAR *)

PROCEDURE IsNumeric (ch: CHAR): BOOLEAN;
  (* Returns TRUE if and only if ch is classified as a numeric character *)

PROCEDURE IsLetter (ch: CHAR): BOOLEAN;
  (* Returns TRUE if and only if ch is classified as a letter *)

PROCEDURE IsUpper (ch: CHAR): BOOLEAN;
  (* Returns TRUE if and only if ch is classified as an upper case letter *)

PROCEDURE IsLower (ch: CHAR): BOOLEAN;
  (* Returns TRUE if and only if ch is classified as a lower case letter *)

PROCEDURE IsControl (ch: CHAR): BOOLEAN;
  (* Returns TRUE if and only if ch represents a control function *)

PROCEDURE IsWhiteSpace (ch: CHAR): BOOLEAN;
  (* Returns TRUE if and only if ch represents a space character or a format effector *)

END CharClass.
```

8.3.2 The Procedures of CharClass

The following definitions refer to sequences of character values defined in 6.10.2.5, or to sets of values built up from lexical definitions in 5.6. The latter are distinguished by a different font, eg. national numeric.

The procedure IsNumeric

The value of the call IsNumeric(ch) shall be *true* if ch is a member of an implementation-defined set of numeric characters that shall include the decimal digits, and shall be *false* otherwise.

$IsNumeric : Arguments \rightarrow \mathbb{B}$

$IsNumeric\,(arg) \triangleq$
  let $[ch] = arg$ in
  $ch \in$ elems $digits \cup \{c \mid c \in$ national numeric$\}$

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The procedure `IsLetter`

The value of the call `IsLetter(ch)` shall be *true* if `ch` is a member of an implementation-defined set of letters that shall include the required letters, and shall be *false* otherwise.

IsLetter : *Arguments* → 𝔹

IsLetter (*arg*) ≙
  let [*ch*] = *arg* in
  ch ∈ elems *required-lower-case-letters* ∪ elems *required-upper-case-letters* ∪ {*c* | *c* ∈ letter}

The procedure `IsUpper`

The value of the call `IsUpper(ch)` shall be *true* if `ch` is a member of an implementation-defined set of upper case letters that shall include the required upper case letters, and shall be *false* otherwise.

IsUpper : *Arguments* → 𝔹

IsUpper (*arg*) ≙
  let [*ch*] = *arg* in
  ch ∈ rng *capitalizations*

The procedure `IsLower`

The value of the call `IsLower(ch)` shall be *true* if `ch` is a member of an implementation-defined set of lower case letters that shall include the required lower case letters, and shall be *false* otherwise.

IsLower : *Arguments* → 𝔹

IsLower (*arg*) ≙
  let [*ch*] = *arg* in
  ch ∈ dom *capitalizations*

The procedure `IsControl`

The value of the call `IsControl(ch)` shall be *true* if `ch` is a member of an implementation-defined set of characters that represent control functions, and shall be *false* otherwise.

IsControl : *Arguments* → 𝔹

IsControl (*arg*) ≙
  let [*ch*] = *arg* in
  ch ∈ *controls*

values

  controls : char-set = implementation-defined

properties

controls ⊆ *implementation-other-characters*

The procedure `IsWhiteSpace`

The value of the call `IsWhiteSpace(ch)` shall be *true* if `ch` is either a space character or a member of an implementation-defined set of characters that represent format effectors, and shall be *false* otherwise.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

$IsWhiteSpace : Arguments \rightarrow \mathbb{B}$

$IsWhiteSpace\,(arg) \triangleq$
  let $[ch] = arg$ in
  $ch = space\text{-}character \vee$
  $ch \in implementation\text{-}whitespace$

values

$implementation\text{-}whitespace$: char-set = implementation-defined

properties

$implementation\text{-}whitespace \subseteq implementation\text{-}other\text{-}characters$

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Section 9 : Standard Library Modules

9.1 The Mathematical Libraries

The mathematical libraries provide the common mathematical functions and some constants.

The module `RealMath` provides the constants and functions for the type `REAL`, while the module `LongMath` provides similar constants and functions for the type `LONGREAL`.

The module `ComplexMath` provides the constants and functions for the type `COMPLEX`, while the module `LongComplexMath` provides similar functions for the type `LONGCOMPLEX`.

9.1.1 The Definition Module of RealMath

```
DEFINITION MODULE RealMath;

  (* Mathematical functions for the type REAL *)

CONST
  pi   = 3.14159265358979323846264338327950288419 72;
  exp1 = 2.71828182845904523536028747135266249775 72;

PROCEDURE sqrt (x: REAL): REAL;
  (* Returns the positive square root of x *)

PROCEDURE exp (x: REAL): REAL;
  (* Returns the exponential of x *)

PROCEDURE ln (x: REAL): REAL;
  (* Returns the natural logarithm of x *)

  (* The angle in all trigonometric functions is measured in radians *)

PROCEDURE sin (x: REAL): REAL;
  (* Returns the sine of x *)

PROCEDURE cos (x: REAL): REAL;
  (* Returns the cosine of x *)

PROCEDURE tan (x: REAL): REAL;
  (* Returns the tangent of x *)

PROCEDURE arcsin (x: REAL): REAL;
  (* Returns the arcsine of x, in the range [-pi/2, pi/2] *)

PROCEDURE arccos (x: REAL): REAL;
  (* Returns the arccosine of x, in the range [0, pi] *)

PROCEDURE arctan (x: REAL): REAL;
  (* Returns the arctangent of x, in the range [-pi/2, pi/2] *)

PROCEDURE power (base, exponent: REAL): REAL;
  (* Returns the value of the number base raised to the power exponent *)

PROCEDURE round (x: REAL): INTEGER;
  (* Returns the value of x rounded to the nearest integer *)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
PROCEDURE IsRMathException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional execution state
     because of the raising of the RealMath exception; otherwise returns FALSE.
  *)

END RealMath.
```

9.1.2 The Definition Module of LongMath

```
DEFINITION MODULE LongMath;

  (* Mathematical functions for the type LONGREAL *)

CONST
  pi   = 3.14159265358979323846264338327950288419 72;
  exp1 = 2.71828182845904523536028747135266249775 72;

PROCEDURE sqrt (x: LONGREAL): LONGREAL;
  (* Returns the positive square root of x *)

PROCEDURE exp (x: LONGREAL): LONGREAL;
  (* Returns the exponential of x *)

PROCEDURE ln (x: LONGREAL): LONGREAL;
  (* Returns the natural logarithm of x *)

  (* The angle in all trigonometric functions is measured in radians *)

PROCEDURE sin (x: LONGREAL): LONGREAL;
  (* Returns the sine of x *)

PROCEDURE cos (x: LONGREAL): LONGREAL;
  (* Returns the cosine of x *)

PROCEDURE tan (x: LONGREAL): LONGREAL;
  (* Returns the tangent of x *)

PROCEDURE arcsin (x: LONGREAL): LONGREAL;
  (* Returns the arcsine of x, in the range [-pi/2, pi/2] *)

PROCEDURE arccos (x: LONGREAL): LONGREAL;
  (* Returns the arccosine of x, in the range [0, pi] *)

PROCEDURE arctan (x: LONGREAL): LONGREAL;
  (* Returns the arctangent of x, in the range [-pi/2, pi/2] *)

PROCEDURE power (base, exponent: LONGREAL): LONGREAL;
  (* Returns the value of the number base raised to the power exponent *)

PROCEDURE round (x: LONGREAL): INTEGER;
  (* Returns the value of x rounded to the nearest integer *)

PROCEDURE IsRMathException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional execution state
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
    because of the raising of the LONGMATH exception; otherwise returns FALSE.
  *)

END LongMath.
```

9.1.3 The Constants of RealMath and LongMath

The constant `pi` shall provide an implementation-defined approximation to the mathematical constant π. The constant `exp1` shall provide an implementation-defined approximation to the mathematical constant e.

NOTE — Due to the approximations involved, `sin(pi)` might not equal zero exactly; similarly, `exp1` might not equal `exp(1)` exactly.

9.1.4 The Procedures of RealMath and LongMath

The semantics of the two modules shall be the same, except that where the module `RealMath` refers to the pervasive type `REAL`, the corresponding function in `LongMath` refers to the pervasive type `LONGREAL`.

NOTES

1 The above statement is merely to avoid needless repetition of the semantics for `LongMath`.

2 The functions sin, cos, tan and the absolute operator used in the definitions below are the mathematical functions that are of infinite precision. Hence the finite precision of actual implementations is modelled by defining a result of infinite precision, followed by an approximation operation (performed by *get-real-result* — see 6.8.2.2).

The units used for angular quantities shall be radians.

The procedure `sqrt`

The value of the call `sqrt(x)` shall be an implementation-defined approximation to the positive signed square root of `x`. An exception shall be raised if `x` is negative.

$sqrt : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} \mathbb{R}$

$sqrt\,(args)handler \triangleq$
  let $[x] = args$ in
  if $x \geq 0$
  then let $r : \mathbb{R}$ be st $r \geq 0 \wedge r^2 = x$ in
    def $ans = get\text{-}real\text{-}result(\text{REAL-TYPE}, r)\ handler$ in 197
    return ans
  else $realmath\text{-}exception(\text{NEGATIVE-SQRT-ARG})\ handler$ 420

The procedure `exp`

The value of the call `exp(x)` shall be an implementation-defined approximation to the mathematical constant e raised to the power of `x`.

$exp : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} \mathbb{R}$

$exp\,(args)handler \triangleq$
  let $[x] = args$ in
  let $r = e^x$ in
  def $ans = get\text{-}real\text{-}result(\text{REAL-TYPE}, r)\ handler$ in 197
  return ans

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The procedure `ln`

The value of the call `ln(x)` shall be an implementation-defined approximation to the natural logarithm of `x`. An exception shall be raised if `x` is zero or negative.

$ln : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} \mathbb{R}$

$ln\,(args)handler \triangleq$
  let $[x] = args$ in
  if $x > 0$
  then let $r : \mathbb{R}$ be st $e^r = x$ in
    def $ans = get\text{-}real\text{-}result(\text{REAL-TYPE}, r)\ handler$ in 197
    return ans
  else $realmath\text{-}exception(\text{NONPOSITIVE-LN-ARG})\ handler$ 420

The procedure `sin`

The value of the call `sin(x)` shall be an implementation-defined approximation to the sine of `x`.

$sin : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} \mathbb{R}$

$sin\,(args)handler \triangleq$
  let $[x] = args$ in
  let $r = \sin(x)$ in
  def $ans = get\text{-}real\text{-}result(\text{REAL-TYPE}, r)\ handler$ in 197
  return ans

The procedure `cos`

The value of the call `cos(x)` shall be an implementation-defined approximation to the cosine of `x`.

$cos : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} \mathbb{R}$

$cos\,(args)handler \triangleq$
  let $[x] = args$ in
  let $r = \cos(x)$ in
  def $ans = get\text{-}real\text{-}result(\text{REAL-TYPE}, r)\ handler$ in 197
  return ans

The procedure `tan`

The value of the call `tan(x)` shall be an implementation-defined approximation to the tangent of `x`. An exception shall be raised if `x` is an odd multiple of $\pi/2$.

$tan : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} \mathbb{R}$

$tan\,(args)handler \triangleq$
  let $[x] = args$ in
  if $\neg\, \exists\, n : \mathbb{Z} \cdot x = (2n + 1)\pi/2$
  then let $r = \tan(x)$ in
    def $ans = get\text{-}real\text{-}result(\text{REAL-TYPE}, r)\ handler$ in 197
    return ans
  else $realmath\text{-}exception(\text{TAN-OVERFLOW})\ handler$ 420

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The procedure arcsin

The value of the call `arcsin(x)` shall be an implementation-defined approximation to the arcsine of `x`. An exception shall be raised if the absolute value of x is greater than one.

$arcsin : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} \mathbb{R}$

$arcsin\,(args)handler \triangleq$
  let $[x] = args$ in
  if $1 \geq x \geq -1$
  then let $r : \mathbb{R}$ be st $\sin(r) = x \wedge$
    $-\pi/2 \leq r \leq \pi/2$ in
    def $ans = get\text{-}real\text{-}result(\text{REAL-TYPE}, r)\ handler$ in 197
    return ans
  else $realmath\text{-}exception(\text{ARCSIN-ARG-MAGNITUDE})\ handler$ 420

The procedure arccos

The value of the call `arccos(x)` shall be an implementation-defined approximation to the arccosine of `x`. An exception shall be raised if the absolute value of x is greater than one.

$arccos : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} \mathbb{R}$

$arccos\,(args)handler \triangleq$
  let $[x] = args$ in
  if $1 \geq x \geq -1$
  then let $r : \mathbb{R}$ be st $\cos(r) = x \wedge$
    $0 \leq r \leq \pi$ in
    def $ans = get\text{-}real\text{-}result(\text{REAL-TYPE}, r)\ handler$ in 197
    return ans
  else $realmath\text{-}exception(\text{ARCCOS-ARG-MAGNITUDE})\ handler$ 420

The procedure arctan

The value of the call `arctan(x)` shall be an implementation-defined approximation to the arctangent of `x`.

$arctan : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} \mathbb{R}$

$arctan\,(args)handler \triangleq$
  let $[x] = args$ in
  let $r : \mathbb{R}$ be st $\tan(r) = x \wedge$
    $-\pi/2 < r < \pi/2$ in
  def $ans = get\text{-}real\text{-}result(\text{REAL-TYPE}, r)\ handler$ in 197
  return ans

The procedure power

The value of the call `power(base,exponent)` shall be an implementation-defined approximation to the result obtained by raising `base` to the power of `exponent`. An exception shall be raised if the value of `base` is zero or negative.

$power : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} \mathbb{R}$

$power\,(args)handler \triangleq$
  let $[x, y] = args$ in
  if $x > 0$
  then let $r = x^y$ in
    def $ans = get\text{-}real\text{-}result(\text{REAL-TYPE}, r)\ handler$ in 197

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
    return ans
  else realmath-exception(NONPOSITIVE-POWER-ARG) handler                    420
```

annotations This function is mathematically equivalent to $\exp(y * \ln(x))$ but may be computed differently.

The procedure round

The value of the call `round(x)` shall be the nearest integer to the value of `x`. The result shall be an implementation-defined selection of the two possible values if the value of `x` is midway between two integer values. An exception shall occur and may be raised if the mathematical result is not within the range of the type `INTEGER`.

```
round : Arguments → Exception-handler ⟶° ℤ
round (args)handler ≙
  let [x] = args in
  let n : ℤ be st abs (n − x) ≤ 0.5 in
  get-whole-result(SIGNED-TYPE, n) handler                                  199
```

The procedure `IsRMathException`

If the calling coroutine is in the state of exceptional execution because of the raising of the `RealMath` exception, the value of the call `RealMath.IsRMathException()` shall be *true*; otherwise the value shall be *false*.

If the calling coroutine is in the state of exceptional execution because of the raising of the `LongMath` exception, the value of the call `LongMath.IsRMathException()` shall be *true*; otherwise the value shall be *false*.

```
IsRMathException : Arguments → Exception-handler ⟶° 𝔹
IsRMathException (args)handler ≙
  def source = realmath-source in
  def res = m-IsCurrentSource([source]) in                                  386
  return res
```

9.1.5 Exceptions Raised by RealMath and LongMath

Following the detection of one of the exceptions defined for the module `RealMath`, the `RealMath` exception shall be raised.

Following the detection of one of the exceptions defined for the module `LongMath`, the `LongMath` exception shall be raised.

The character string associated with each of the exceptions raised by `RealMath` and `LongMath` shall be implementation-defined.

```
types
  Realmathexception = NEGATIVE-SQRT-ARG
                    | NONPOSITIVE-LN-ARG
                    | NONPOSITIVE-POWER-ARG
                    | TAN-OVERFLOW
                    | ARCSIN-ARG-MAGNITUDE
                    | ARCCOS-ARG-MAGNITUDE

realmath-exception : Realmathexception → Exception-handler ⟶° ()
realmath-exception (rme)handler ≙
  def source = realmath-source in
  def message = realmath-exception-string(rme) in                           421
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

let *number* ∈ N in
m-RAISE([*source*, *number*, *message*]) *handler* 383

realmath-exception-string : *Realmathexception* $\xrightarrow{o}$ *Message*
realmath-exception-string (*rme*) $\triangleq$
implementation-defined

annotations The result is an implementation-defined string value.

9.1.6 The State for RealMath and LongMath

The state for each of the modules `RealMath` and `LongMath` shall include a value of the exception source type identifying exceptions raised by the corresponding module.

state *RealMath* of
realmath-source : *Exception-source*
end

9.1.7 The Initialization of RealMath and LongMath

The initialization of the modules `RealMath` and `LongMath` shall allocate a unique value of the exception source type to identify exceptions raised by the module.

realmath-initialization : () → *Exception-handler* $\xrightarrow{o}$ ()
realmath-initialization ()*handler* $\triangleq$
def *source* = *allocate-source*() in 342
if *source* ≠ nil
then *realmath-source* := *source*
else *mandatory-detection*(NO-EXCEPTION-SOURCE) *handler* 396

9.1.8 The Definition Module of ComplexMath

```
DEFINITION MODULE ComplexMath;

  (* Mathematical functions for the type COMPLEX *)

CONST
  i =    CMPLX (0.0, 1.0);
  one =  CMPLX (1.0, 0.0);
  zero = CMPLX (0.0, 0.0);

PROCEDURE abs (z: COMPLEX): REAL;
  (* Returns the length of z *)

PROCEDURE arg (z: COMPLEX): REAL;
  (* Returns the angle that z subtends to the positive real axis, in the range [-pi, pi] *)

PROCEDURE conj (z: COMPLEX): COMPLEX;
  (* Returns the complex conjugate of z *)

PROCEDURE power (base: COMPLEX; exponent: REAL): COMPLEX;
  (* Returns the value of the number base raised to the power exponent *)

PROCEDURE sqrt (z: COMPLEX): COMPLEX;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  (* Returns the principal square root of z, with arg in the range [-pi/2, pi/2] *)

PROCEDURE exp (z: COMPLEX): COMPLEX;
  (* Returns the complex exponential of z *)

PROCEDURE ln (z: COMPLEX): COMPLEX;
  (* Returns the principal value of the natural logarithm of z *)

PROCEDURE sin (z: COMPLEX): COMPLEX;
  (* Returns the sine of z *)

PROCEDURE cos (z: COMPLEX): COMPLEX;
  (* Returns the cosine of z *)

PROCEDURE tan (z: COMPLEX): COMPLEX;
  (* Returns the tangent of z *)

PROCEDURE arcsin (z: COMPLEX): COMPLEX;
  (* Returns the arcsine of z *)

PROCEDURE arccos (z: COMPLEX): COMPLEX;
  (* Returns the arccosine of z *)

PROCEDURE arctan (z: COMPLEX): COMPLEX;
  (* Returns the arctangent of z *)

PROCEDURE polarToComplex (abs, arg: REAL): COMPLEX;
  (* Returns the complex number with the specified polar coordinates *)

PROCEDURE scalarMult (scalar: REAL; z: COMPLEX): COMPLEX;
  (* Returns the scalar product of scalar with z *)

PROCEDURE IsCMathException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional execution state
     because of the ComplexMath exception; otherwise returns FALSE.
  *)

END ComplexMath.
```

9.1.9 The Definition Module of LongComplexMath

```
DEFINITION MODULE LongComplexMath;

  (* Mathematical functions for the type LONGCOMPLEX *)

CONST
  i =    CMPLX (0.0, 1.0);
  one =  CMPLX (1.0, 0.0);
  zero = CMPLX (0.0, 0.0);

PROCEDURE abs (z: LONGCOMPLEX): LONGREAL;
  (* Returns the length of z *)

PROCEDURE arg (z: LONGCOMPLEX): LONGREAL;
  (* Returns the angle that z subtends to the positive real axis, in the range [-pi, pi] *)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
PROCEDURE conj (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the complex conjugate of z *)

PROCEDURE power (base: LONGCOMPLEX; exponent: LONGREAL): LONGCOMPLEX;
  (* Returns the value of the number base raised to the power exponent *)

PROCEDURE sqrt (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the principal square root of z with arg in the range [-pi/2, pi/2] *)

PROCEDURE exp (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the complex exponential of z *)

PROCEDURE ln (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the principal value of the natural logarithm of z *)

PROCEDURE sin (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the sine of z *)

PROCEDURE cos (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the cosine of z *)

PROCEDURE tan (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the tangent of z *)

PROCEDURE arcsin (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the arcsine of z *)

PROCEDURE arccos (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the arccosine of z *)

PROCEDURE arctan (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the arctangent of z *)

PROCEDURE polarToComplex (abs, arg: LONGREAL): LONGCOMPLEX;
  (* Returns the complex number with the specified polar coordinates *)

PROCEDURE scalarMult (scalar: LONGREAL; z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the scalar product of scalar with z *)

PROCEDURE IsCMathException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional execution state
     because of the LongComplexMath exception; otherwise returns FALSE.
  *)

END LongComplexMath.
```

9.1.10 The Constants of ComplexMath and LongComplexMath

The constants `i`, `one`, and `zero` shall be the implementation-defined approximations to the specified values.

NOTE — These constants are provided for convenience.

9.1.11 The Procedures of ComplexMath and LongComplexMath

The semantics of the two modules shall be the same, except that where the module `ComplexMath` refers to the pervasive type `COMPLEX`, the corresponding function in `LongComplexMath` refers to the pervasive type `LONGCOMPLEX`.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NOTES

1 The above statement is merely to avoid needless repetition of the semantics for `LongComplexMath`.

2 The functions sin, cos, arctan and the absolute operator used in the definitions below are the mathematical functions that are of infinite precision. Hence the finite precision of actual implementations is modelled by defining a result of infinite precision, followed by an approximation operation (performed by *get-real-result* or *get-complex-result* — see 6.8.2.2 and 6.8.2.1).

The procedure `abs`

The value of the call `abs(z)` shall be an implementation-defined approximation to the modulus (otherwise known as the length, or absolute value) of `z`.

NOTE 3 — An overflow exception may be raised in this computation, even when the complex number is itself well defined.

$abs : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} \mathbb{R}$

$abs\,(args)handler \triangleq$
  let $[z] = args$ in
  let $r = modulus\,(z)$ in
  def $ans = get\text{-}real\text{-}result(\text{REAL-TYPE}, r)\ handler$ in 197
  return ans

The procedure `arg`

The value of the call `arg(z)` shall be an implementation-defined approximation to the angle that `z` subtends to the positive real axis in the complex plane. An exception shall be raised if the modulus of `z` is zero.

$arg : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} \mathbb{R}$

$arg\,(args)handler \triangleq$
  let $[z] = args$ in
  let $re = \mathrm{re}z$,
    $im = \mathrm{im}z$ in
  ($re > 0 \rightarrow$ def $ans = get\text{-}real\text{-}result(\text{REAL-TYPE}, \arctan(im/re))\ handler$ in 197
      return ans,
   $re < 0 \rightarrow$ let $res =$ if $im \geq 0$ then $\pi + \arctan(im/re)$ else $-\pi + \arctan(im/re)$ in
      def $ans = get\text{-}real\text{-}result(\text{REAL-TYPE}, res)\ handler$ in 197
      return ans,
   $re = 0 \rightarrow$ if $im = 0$
      then $complexmath\text{-}exception(\text{ZERO-ARG-PARAMETER})\ handler$ 428
      else def $ans = get\text{-}real\text{-}result(\text{REAL-TYPE}, \text{if } im > 0 \text{ then } \pi/2 \text{ else } -\pi/2)\ handler$ in 197
        return ans)

The procedure `conj`

The value of the call `conj(z)` shall be an implementation-defined approximation to the complex conjugate of `z`.

$conj : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} \mathbb{C}$

$conj\,(args)handler \triangleq$
  let $[z] = args$ in
  let $re = \mathrm{re}z$,
    $im = \mathrm{im}z$ in
  let $c : \mathbb{C}$ be st $\mathrm{re}c = re \wedge$
      $\mathrm{im}c = -im$ in
  def $ans = get\text{-}complex\text{-}result(\text{COMPLEX-TYPE}, c)\ handler$ in 194
  return ans

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The procedure power

The value of the call power(base,exponent) shall be an implementation-defined approximation to the result obtained by raising base to the power of exponent.

complex-power : *Arguments* → *Exception-handler* $\xrightarrow{o}$ $\mathbb{C}$

complex-power (*args*)*handler* $\triangleq$
  let [*base*, *exponent*] = *args* in
  let *abs* = *modulus* (*base*), 429
    arg = *angle* (*base*) in 429
  let $c : \mathbb{C}$ be st re$c$ = $abs^{exponent} \times \cos(exponent \times arg) \wedge$
    im$c$ = $abs^{exponent} \times \sin(exponent \times arg)$ in
  def *ans* = *get-complex-result*(COMPLEX-TYPE, *c*) *handler* in 194
  return *ans*

The procedure sqrt

The value of the call sqrt(z) shall be an implementation-defined approximation to the principal square root of z.

NOTE 4 — That is, the result is the complex number with an argument of half the value of the argument of z, and whose modulus has the value of the positive square root of the modulus of z.

complex-sqrt : *Arguments* → *Exception-handler* $\xrightarrow{o}$ $\mathbb{C}$

complex-sqrt (*args*)*handler* $\triangleq$
  let [*z*] = *args* in
  let $c : \mathbb{C}$ be st *modulus* $(c)^2$ = *modulus* $(z) \wedge$ 429 429
    angle (c) = *angle* $(z)/2$ in 429 429
  def *ans* = *get-complex-result*(COMPLEX-TYPE, *c*) *handler* in 194
  return *ans*

The procedure exp

The value of the call exp(z) shall be an implementation-defined approximation to the mathematical constant *e* raised to the power of z.

complex-exp : *Arguments* → *Exception-handler* $\xrightarrow{o}$ $\mathbb{C}$

complex-exp (*args*)*handler* $\triangleq$
  let [*z*] = *args* in
  let *re* = re$z$,
    im = im$z$ in
  let $c : \mathbb{C}$ be st re$c$ = $e^{re} \times \cos(im) \wedge$
    im$c$ = $e^{re} \times \sin(im)$ in
  def *ans* = *get-complex-result*(COMPLEX-TYPE, *c*) *handler* in 194
  return *ans*

The procedure ln

The value of the call ln(z) shall be an implementation-defined approximation to the principal value of the natural logarithm of z.

complex-ln : *Arguments* → *Exception-handler* $\xrightarrow{o}$ $\mathbb{C}$

complex-ln (*args*)*handler* $\triangleq$
  let [*z*] = *args* in
  let *abs* = *modulus* (*z*), 429

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
    arg = angle(z) in                                                    429
  let c : ℂ be st re c = ln(abs) ∧
                  im c = arg in
  def ans = get-complex-result(COMPLEX-TYPE, c) handler in               194
  return ans
```

The procedure `sin`

The value of the call `sin(z)` shall be an implementation-defined approximation to the complex sine of `z`.

```
complex-sin : Arguments → Exception-handler -o→ ℂ
complex-sin (args)handler ≙
  let [z] = args in
  let c = sin(z) in
  def ans = get-complex-result(COMPLEX-TYPE, c) handler in               194
  return ans
```

The procedure `cos`

The value of the call `cos(z)` shall be an implementation-defined approximation to the complex cosine of `z`.

```
complex-cos : Arguments → Exception-handler -o→ ℂ
complex-cos (args)handler ≙
  let [z] = args in
  let c = cos(z) in
  def ans = get-complex-result(COMPLEX-TYPE, c) handler in               194
  return ans
```

The procedure `tan`

The value of the call `tan(z)` shall be an implementation-defined approximation to the complex tangent of `z`. An exception shall be raised if `z` is an odd multiple of $\pi/2$.

```
complex-tan : Arguments → Exception-handler -o→ ℂ
complex-tan (args)handler ≙
  let [z] = args in
  let re = re z,
      im = im z in
  if z = 0
  then return z
  elseif ¬ (im = 0 ∧ (∃ n : ℤ · re = (2n + 1)π/2))
  then let c = tan(z) in
       def ans = get-complex-result(COMPLEX-TYPE, c) handler in          194
       return ans
  else complexmath-exception(COMPLEX-TAN-OVERFLOW) handler               428
```

The procedure `arcsin`

The value of the call `arcsin(z)` shall be an implementation-defined approximation to the principal value of the complex arcsine of `z`.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

$complex\text{-}arcsin : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} \mathbb{C}$

$complex\text{-}arcsin\,(args)handler \triangleq$
  let $[z] = args$ in
  let $cs = \{c : \mathbb{C} \mid \sin(c) = z \wedge angle(c) > 0\}$ in 429
  let $c1 : \mathbb{C}$ be st $c1 \in cs \wedge \neg \exists\, c2 \in cs \cdot angle(c2) < angle(c1)$ in 429 429
  def $ans = get\text{-}complex\text{-}result(\text{COMPLEX-TYPE}, c1)\ handler$ in 194
  return ans

The procedure `arccos`

The value of the call `arccos(z)` shall be an implementation-defined approximation to the complex arccosine of `z`.

$complex\text{-}arccos : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} \mathbb{C}$

$complex\text{-}arccos\,(args)handler \triangleq$
  let $[z] = args$ in
  let $cs = \{c : \mathbb{C} \mid \cos(c) = z \wedge angle(c) > 0\}$ in 429
  let $c1 : \mathbb{C}$ be st $c1 \in cs \wedge \neg \exists\, c2 \in cs \cdot angle(c2) < angle(c1)$ in 429 429
  def $ans = get\text{-}complex\text{-}result(\text{COMPLEX-TYPE}, c1)\ handler$ in 194
  return ans

The procedure `arctan`

The value of the call `arctan(z)` shall be an implementation-defined approximation to the complex arctangent of `z`.

$complex\text{-}arctan : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} \mathbb{C}$

$complex\text{-}arctan\,(args)handler \triangleq$
  let $[z] = args$ in
  let $cs = \{c : \mathbb{C} \mid \tan(c) = z \wedge angle(c) > 0\}$ in 429
  let $c1 : \mathbb{C}$ be st $c1 \in cs \wedge \neg \exists\, c2 \in cs \cdot angle(c2) < angle(c1)$ in 429 429
  def $ans = get\text{-}complex\text{-}result(\text{COMPLEX-TYPE}, c1)\ handler$ in 194
  return ans

The procedure `polarToComplex`

The value of the call `polarToComplex(abs,arg)` shall be an implementation-defined approximation to the complex number that has a modulus given by `abs` and an argument given by `arg`.

$polarToComplex : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} \mathbb{C}$

$polarToComplex\,(args)handler \triangleq$
  let $[abs, arg] = args$ in
  let $c : \mathbb{C}$ be st $rec = abs \times \cos(arg) \wedge$
    $imc = abs \times \sin(arg)$ in
  def $ans = get\text{-}complex\text{-}result(\text{COMPLEX-TYPE}, c)\ handler$ in 194
  return ans

The procedure `scalarMult`

The value of the call `scalarMult(scalar,z)` shall be an implementation-defined approximation to the scalar product of the real value `scalar` with the complex value `z`.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

scalarMult : *Arguments* → *Exception-handler* $\xrightarrow{o}$ $\mathbb{C}$

scalarMult (*args*)*handler* $\triangleq$
  let [*scalar*, *z*] = *args* in
  let *re* = re*z*,
    im = im*z* in
  let *c* : $\mathbb{C}$ be st re*c* = *scalar* × *re* ∧
      im*c* = *scalar* × *im* in
  def *ans* = *get-complex-result*(COMPLEX-TYPE, *c*) *handler* in 194
  return *ans*

The procedure `IsCMathException`

If the calling coroutine is in the state of exceptional execution because of the raising of a `ComplexMath` exception, the value of the call `ComplexMath.IsCMathException()` shall be *true*; otherwise the value shall be *false*.

If the calling coroutine is in the state of exceptional execution because of the raising of a `LongComplexMath` exception, the value of the call `LongComplexMath.IsCMathException()` shall be *true*; otherwise the value shall be *false*.

IsCMathException : *Arguments* → *Exception-handler* $\xrightarrow{o}$ $\mathbb{B}$

IsCMathException (*args*)*handler* $\triangleq$
  def *source* = *complexmath-source* in
  def *res* = *m-IsCurrentSource*([*source*]) in 386
  return *res*

9.1.12 Exceptions Raised by ComplexMath and LongComplexMath

Following the detection of one of the exceptions defined for the module `ComplexMath`, the `ComplexMath` exception shall be raised.

Following the detection of one of the exceptions defined for the module `LongComplexMath`, the `LongComplexMath` exception shall be raised.

The character string associated with each of the exceptions raised by `ComplexMath` and `LongComplexMath` shall be implementation-defined.

types

  Complexmathexception = ZERO-ARG-PARAMETER
    | COMPLEX-TAN-OVERFLOW

complexmath-exception : *Complexmathexception* → *Exception-handler* $\xrightarrow{o}$ ()

complexmath-exception (*cme*)*handler* $\triangleq$
  def *source* = *complexmath-source* in
  def *message* = *complexmath-exception-string*(*cme*) in 428
  let *number* ∈ $\mathbb{N}$ in
  m-RAISE([*source*, *number*, *message*]) *handler* 383

complexmath-exception-string : *Complexmathexception* $\xrightarrow{o}$ *Message*

complexmath-exception-string (*rme*) $\triangleq$
  implementation-defined

annotations The result is an implementation-defined string value.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9.1.13 The State for ComplexMath and LongComplexMath

The state for each of the modules `ComplexMath` and `LongComplexMath` shall include a value of the exception source type identifying exceptions raised by the corresponding module.

```
state ComplexMath of
  complexmath-source : Exception-source
end
```

9.1.14 The Initialization of ComplexMath and LongComplexMath

The initialization of the modules `ComplexMath` and `LongComplexMath` shall allocate a unique value of the exception source type to identify exceptions raised by the module.

```
complexmath-initialization : () → Exception-handler ^o→ ()
complexmath-initialization (handler) △
  def source = allocate-source() in                                    342
  if source ≠ nil
  then complexmath-source := source
  else mandatory-detection(NO-EXCEPTION-SOURCE) handler                396
```

9.1.15 Auxiliary Functions

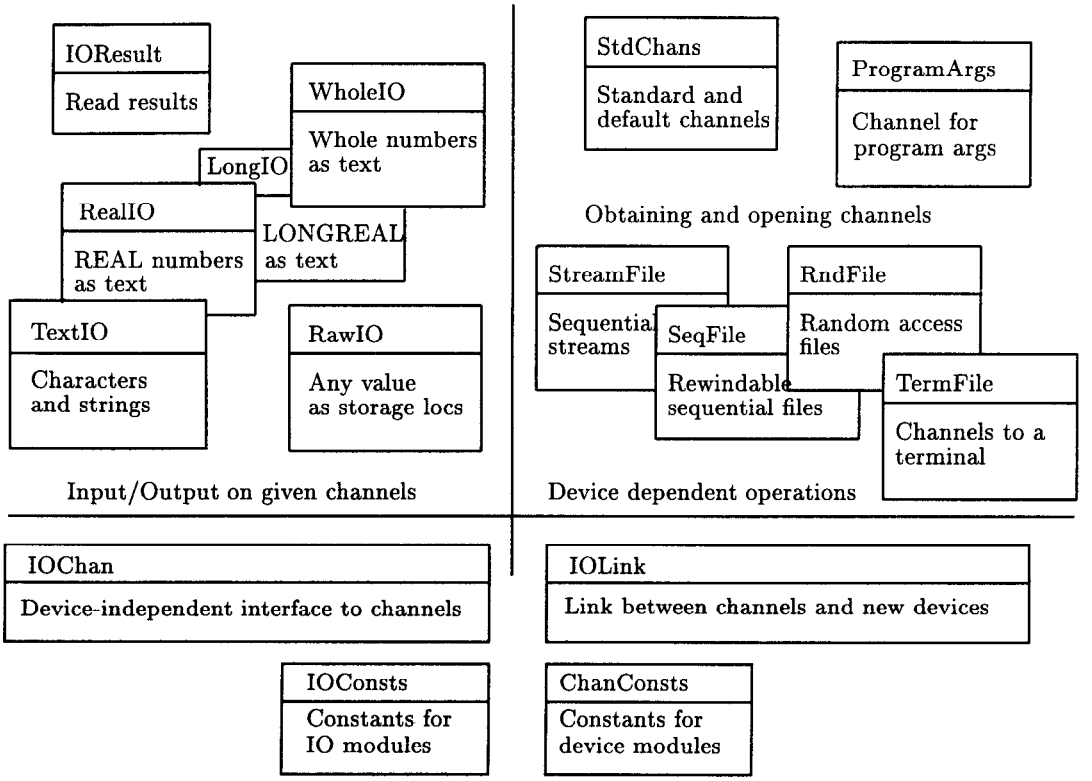
```
modulus : ℂ → ℝ
modulus (z) △
  let r : ℝ be st r ≥ 0 ∧ r² = (re z)² + (im z)² in
  r

angle : ℂ → ℝ
angle (z) △
  let P = {mk-(r, θ) | mk-(r, θ) : ℝ × ℝ · r ≥ 0 ∧ z = r × e^(iθ)} in
  (r = 0 → 0,
   r ≠ 0 → let mk-(-, ans) ∈ P be st −π < ans ≤ π in ans)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2 The Input/Output Library

The input/output library defined in this clause allows for the reading and writing of data streams over one or more *channels*. Channels are connected to sources of input data, or to destinations of output data, known as *devices* or *device instances*. There is a separation between modules that are concerned with device-independent operations, such as reading and writing, and modules concerned with device-dependent operations, such as making connections to named files. This separation allows the library to be extended to work with new devices. The module structure of the library is depicted in the following figure.



Channels already open to standard sources and destinations can be identified using procedures provided by the module `StdChans`. This module also provides for the identification and selection of channels used by default for input and output operations.

The modules `TextIO`, `WholeIO`, `RealIO`, and `LongIO`, provide facilities that allow the reading and writing of high-level units of data, using text operations on channels specified explicitly by a parameter. These high-level units include characters, strings, and whole numbers and real numbers in decimal notation. The module `RawIO` provides facilities that allow for the reading and writing of arbitrary data types, using raw (binary) operations on explicitly specified channels.

Text operations produce or consume data streams as sequences of characters and line marks. Raw operations produce or consume data streams as sequences of storage locations (i.e. as arrays whose component type is `SYSTEM.LOC`).

The library allows devices to support both text and raw operations on a single channel, although this behaviour is not required.

The module `IOResult` provides the facility for a program to determine whether the last operation to read data from a specified input channel found data in the required format.

Corresponding to the `TextIO` group of modules is a group of modules `STextIO`, `SWholeIO`, `SRealIO`, `SLongIO`, `SRawIO` and `SIOResult`. The prefix "S" serves as an abbreviation for "Simple". The procedures exported from this group do not take parameters identifying a channel. They operate on the default input and output channels, as identified by the module `StdChans`.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The module IOConsts defines types and constants used by IOResult and SIOResult.

The device modules StreamFile, SeqFile, RndFile, and TermFile provide facilities that allow a channel to be opened to a named stream, to a rewindable sequential file, to a random access file, or to a terminal device respectively. The device module ProgramArgs provides an open channel from which program arguments may be read. Device specific operations, such as positioning within a random access file, are also defined by the appropriate device module.

The module ChanConsts defines the constants and types used in those device module procedures that open channels.

The primitive device-independent operations on channels are provided by the module IOChan.

The module IOChan defines general input/output library exception values that may be raised when using any device through a channel. Device errors, such as a hardware read/write error, are reported by raising one of the general exception values, and providing an implementation-defined error number. Exception values associated with device-specific operations are defined by the appropriate device module.

The module IOLink provides facilities that allow a user to provide further specialized device modules for use with channels, following the pattern of the rest of the library.

NOTE — Partial implementations of the input/output library may provide modules selected exclusively from the group STextIO, SWholeIO, SRealIO, and SLongIO, normally with SIOResult and IOConsts. If any other module is provided, the module IOChan must also be provided, in accordance with the import dependencies between the definition modules of the library.

9.2.1 Standard and Default Channels

Standard channels do not have to be opened by a client program since they are already open and ready for use. Under some operating systems they may be connected to sources and destinations specified before the program is run, while on a stand-alone system they may be connected to a console terminal.

No method is provided for closing a standard channel, and the values used to identify standard channels are constant throughout the execution of the program.

Default channels are channels whose identities have been stored as those to be used by default for input and output operations. Initially these correspond to the standard channels, but their values may be varied to obtain the effect of redirection.

9.2.1.1 The Module StdChans

The module StdChans defines functions that identify channels already open to implementation-defined sources and destinations of standard input, standard output, and standard error output. Access to a 'null device' is provided to allow unwanted output to be suppressed. The null device throws away all data written to it, and gives an immediate end of input indication on reading.

The module StdChans further allows for identification and selection of the channels used by default for input and output operations.

9.2.1.1.1 The Definition Module of StdChans

```
DEFINITION MODULE StdChans;

  (* Access to standard and default channels *)

IMPORT IOChan;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
TYPE
  ChanId = IOChan.ChanId;
    (* Values of this type are used to identify channels *)

  (* The following functions return the standard channel values.
     These channels cannot be closed.
  *)

PROCEDURE StdInChan (): ChanId;
  (* Returns the identity of the implementation-defined standard source for program
     input.
  *)

PROCEDURE StdOutChan (): ChanId;
  (* Returns the identity of the implementation-defined standard source for program
     output.
  *)

PROCEDURE StdErrChan (): ChanId;
  (* Returns the identity of the implementation-defined standard destination for program
     error messages.
  *)

PROCEDURE NullChan (): ChanId;
  (* Returns the identity of a channel open to the null device. *)

  (* The following functions return the default channel values *)

PROCEDURE InChan (): ChanId;
  (* Returns the identity of the current default input channel. *)

PROCEDURE OutChan (): ChanId;
  (* Returns the identity of the current default output channel. *)

PROCEDURE ErrChan (): ChanId;
  (* Returns the identity of the current default error message channel. *)

  (* The following procedures allow for redirection of the default channels *)

PROCEDURE SetInChan (cid: ChanId);
  (* Sets the current default input channel to that identified by cid. *)

PROCEDURE SetOutChan (cid: ChanId);
  (* Sets the current default output channel to that identified by cid. *)

PROCEDURE SetErrChan (cid: ChanId);
  (* Sets the current default error channel to that identified by cid. *)

END StdChans.
```

9.2.1.1.2 The Procedures of StdChans

The procedure `StdInChan`

The value of the call `StdInChan()` shall be a value identifying a channel open to the implementation-defined standard source for program input.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

StdInChan : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Chan-id*

StdInChan (*args*)*handler* $\triangleq$
  return *stdinchan* ()

537

The procedure StdOutChan

The value of the call StdOutChan() shall be a value identifying a channel open to the implementation-defined standard destination for program output.

StdOutChan : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Chan-id*

StdOutChan (*args*)*handler* $\triangleq$
  return *stdoutchan* ()

538

The procedure StdErrChan

The value of the call StdErrChan() shall be a value identifying a channel open to the implementation-defined standard destination for program error messages.

StdErrChan : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Chan-id*

StdErrChan (*args*)*handler* $\triangleq$
  return *stderrchan* ()

538

The procedure NullChan

The value of the call NullChan() shall be a value identifying a channel open to the null device.

NOTE — The null device supports all operations by discarding all data written to it, or by giving an immediate end of input indication on reading.

NullChan : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Chan-id*

NullChan (*args*)*handler* $\triangleq$
  return *nullchan* ()

538

The procedure InChan

The value of the call InChan() shall be the identity of the current default input channel. This shall be the channel used by input procedures that do not take a channel parameter. Initially this shall be the value of the call StdInChan().

InChan : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Chan-id*

InChan (*args*)*handler* $\triangleq$
  inchan()

538

The procedure OutChan

The value of the call OutChan() shall be the identity of the current default output channel. This shall be the channel used by output procedures that do not take a channel parameter. Initially this shall be the value of the call StdOutChan().

OutChan : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Chan-id*

OutChan (*args*)*handler* $\triangleq$
  outchan()

538

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The procedure `ErrChan`

The value of the call `ErrChan()` shall be the identity of the current default output channel for program error messages. Initially this shall be the value of the call `StdErrChan()`.

ErrChan : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Chan-id*

ErrChan (*args*)*handler* ≙
  errchan() 538

The procedure `SetInChan`

The call `SetInChan(cid)` shall set the current default input channel to that identified by `cid`.

SetInChan : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

SetInChan (*args*)*handler* ≙
  let [*cid*] = *args* in *setinchan*(*cid*) 538

The procedure `SetOutChan`

The call `SetOutChan(cid)` shall set the current default output channel to that identified by `cid`.

SetOutChan : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

SetOutChan (*args*)*handler* ≙
  let [*cid*] = *args* in *setoutchan*(*cid*) 538

The procedure `SetErrChan`

The call `SetErrChan(cid)` shall set the current default output channel for error messages to that identified by `cid`.

SetErrChan : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

SetErrChan (*args*)*handler* ≙
  let [*cid*] = *args* in *seterrchan*(*cid*) 538

9.2.2 Reading and Writing of Data using Specified Channels

The module `TextIO` provides facilities that allow for the input and output of characters, character strings, and line marks, using text operations.

The module `WholeIO` provides facilities that allow for the input and output of whole numbers in decimal text form.

The modules `RealIO` and `LongIO` provide facilities that allow for the input and output of real numbers in decimal text form.

The module `RawIO` provides facilities that allow for for the direct input and output of data, using raw operations (i.e. without any interpretation).

The input procedures of the modules `TextIO`, `WholeIO`, `RealIO`, `LongIO`, and `RawIO` are sufficient for use where the format of the input data is known. Since, in practice, their use may be inconsistent with the format of the input data, they have the effect of setting a 'read result' for the used channel. The module `IOResult` provides the facility for obtaining the read result applicable to the most recent input operation on a given channel.

In all cases, channels are selected explicitly by passing an actual parameter of the type `ChanId` to the procedures of these modules.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9.2.2.1 The Module TextIO

The module TextIO provides facilities that allow for the input and output of characters, character strings, and line marks, using text operations.

9.2.2.1.1 The Definition Module of TextIO

```
DEFINITION MODULE TextIO;

  (* Input and output of character and string types over specified channels.
     The read result is of the type IOConsts.ReadResults.
  *)

IMPORT IOChan;

  (* The following procedures do not read past line marks *)

PROCEDURE ReadChar (cid: IOChan.ChanId; VAR ch: CHAR);
  (* If possible, removes a character from the input stream cid and assigns the
     corresponding value to ch.  The read result is set to the value allRight, endOfLine, or
     endOfInput.
  *)

PROCEDURE ReadRestLine (cid: IOChan.ChanId; VAR s: ARRAY OF CHAR);
  (* Removes any remaining characters from the input stream cid before the next line mark,
     copying to s as many as can be accommodated as a string value. The read result is set
     to the value allRight, outOfRange, endOfLine, or endOfInput.
  *)

PROCEDURE ReadString (cid: IOChan.ChanId; VAR s: ARRAY OF CHAR);
  (* Removes only those characters from the input stream cid before the next line mark that
     can be accommodated in s as a string value, and copies them to s.  The read result is
     set to the value allRight, endOfLine, or endOfInput.
  *)

PROCEDURE ReadToken (cid: IOChan.ChanId; VAR s: ARRAY OF CHAR);
  (* Skips leading spaces, and then removes characters from the input stream cid before the
     next space or line mark, copying to s as many as can be accommodated as a string value.
     The read result is set to the value allRight, outOfRange, endOfLine, or endOfInput.
  *)

  (* The following procedure reads past the next line mark *)

PROCEDURE SkipLine (cid: IOChan.ChanId);
  (* Removes successive items from the input stream cid up to and including the next line
     mark, or until the end of input is reached.  The read result is set to the value
     allRight, or endOfInput.
  *)

  (* Output procedures *)

PROCEDURE WriteChar (cid: IOChan.ChanId; ch: CHAR);
  (* Writes the value of ch to the output stream cid. *)

PROCEDURE WriteLn (cid: IOChan.ChanId);
  (* Writes a line mark to the output stream cid. *)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
PROCEDURE WriteString (cid: IOChan.ChanId; s: ARRAY OF CHAR);
  (* Writes the string value in s to the output stream cid. *)

END TextIO.
```

9.2.2.1.2 The Procedures of TextIO

The procedure `ReadChar`

If there is a character next in the input stream identified by `cid`, the call `ReadChar(cid,ch)` shall remove it from the stream and assign its value to `ch`; otherwise the value of `ch` is not defined. The read result for the channel shall be set to the value

`allRight`	if a character is read;
`endOfLine`	if no character is read, the next item being a line mark;
`endOfInput`	if no character is read, the input stream having ended.

ReadChar : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

ReadChar (*args*)*handler* $\triangleq$
  let [*cid*, *ch*] = *args* in
  def *tid* = *table-id*(*cid*) *handler* in 539
  def *res* = *do-look*(*tid*, *ch*) *handler* in 451
  if *res* = *allRight*
  then *do-skip*(*tid*) *handler* 452
  else skip

annotations Reading a character is modelled as *Look*, followed by *Skip* if a character was seen. Other implementation strategies are possible, for example a *TextRead* into a component of an array of the character-type.

The procedure `ReadRestLine`

If there is a character next in the input stream identified by `cid`, the call `ReadRestLine(cid,s)` shall read a string of characters; reading shall continue as long as there are still characters before the next line mark or the end of the stream. As much of the string as can be accommodated shall be copied to `s` as a string value. The read result for the channel shall be set to the value

`allRight`	if `s` is not empty and accomodates all of the string that has been read;
`outOfRange`	if `s` is not empty but does not accommodate all of the string;
`endOfLine`	if `s` is empty, the next item being a line mark;
`endOfInput`	if `s` is empty, the input stream having ended.

ReadRestLine : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

ReadRestLine (*args*)*handler* $\triangleq$
  let [*cid*, *s*] = *args* in
  def *tid* = *table-id*(*cid*) *handler* in 539
  def *line* = *read-while*(*tid*, $\lambda\, x$: *Char** · true) *handler* in 452
  (if *line* $\neq$ [ ]
   then if len *line* $\leq$ *capacity* (*s*) 579
        then *set-rdresult*(*tid*, *allRight*) 539
        else *set-rdresult*(*tid*, *outOfRange*) 539
   else skip;
   assign(*s*, *line*) *handler*) 150

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The procedure ReadString

If there is a character next in the input stream identified by cid, the call ReadString(cid,s) shall read a string of characters; reading shall continue as long as there are still characters before the next line mark or the end of the stream and the capacity of s has not been exhausted. The string shall be copied to s as a string value. The read result for the channel shall be set to the value

allRight	if s is not empty;
endOfLine	if s is empty, the next item being a line mark;
endOfInput	if s is empty, the input stream having ended.

$ReadString : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} ()$

$ReadString\,(args)handler \triangleq$
  let $[cid, s] = args$ in
  def $tid = table\text{-}id(cid)\ handler$ in 539
  def $nRead = read\text{-}chars(tid, s)\ handler$ in 453
  if $nRead < capacity\,(s)$ 579
  then $assign(s\,(nRead), \text{END-OF-STRING-CHAR})\ handler$ 150
  else skip

The procedure ReadToken

The call ReadToken(cid,s) shall first skip any leading spaces in the input stream identified by cid. If the next item is a character, a string of characters shall be read; reading shall continue as long as there are still non-space characters before the next line mark or the end of the stream. As much of the string as can be accommodated shall be copied to s as a string value. The read result for the channel shall be set to the value

allRight	if s is not empty and accomodates all of the string that has been read;
outOfRange	if s is not empty but does not accommodate all of the string;
endOfLine	if s is empty, the next item being a line mark;
endOfInput	if s is empty, the input stream having ended.

$ReadToken : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} ()$

$ReadToken\,(args)handler \triangleq$
  let $[cid, s] = args$ in
  def $tid = table\text{-}id(cid)\ handler$ in 539
  def $sps\text{-}tok = read\text{-}while(tid, \lambda\, x : Char^* \cdot isa\text{-}token\,(x))\ handler$ in 452 437
  let $sps, tok : Char^*$ be st $sps \frown tok = sps\text{-}tok \wedge tok = no\text{-}leading\text{-}spaces\,(sps\text{-}tok)$ in 451
  (if $tok \neq []$
   then if len $tok \leq capacity\,(s)$ 579
        then $set\text{-}rdresult(tid, allRight)$ 539
        else $set\text{-}rdresult(tid, outOfRange)$ 539
   else skip;
   $assign(s, tok)\ handler$) 150

$isa\text{-}token : Char^* \rightarrow \mathbb{B}$

$isa\text{-}token\,(s) \triangleq$
  let $sps, tok : Char^*$ be st $sps \frown tok = s \wedge tok = no\text{-}leading\text{-}spaces\,(s)$ in 451
  $\forall\, t \in \text{elems } tok \cdot \neg\, isa\text{-}space\,(t)$ 450

The procedure SkipLine

The call SkipLine(cid) shall read successive items from the input stream identified by cid up to and including the next line mark, or until the end of the stream is reached.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The read result for the channel shall be set to the value

`allRight`	if a line mark is read;
`endOfInput`	if no line mark is read, the input stream having ended.

SkipLine : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

SkipLine (*args*)*handler* $\triangleq$
  let [*cid*] = *args* in
  def *tid* = *table-id*(*cid*) *handler* in 539
  def *loc* = *allocate-abstract-storage*() in 337
  let *ch* = *mk-Elementary-variable* (*loc*, nil) in
  def res_1 = *do-look*(*tid*, *ch*) *handler* in 451
  def *result* = if res_1 = *allRight*
    then def res_2 = *do-skip-rest*(*tid*, *ch*) *handler* in res_2 438
    else res_1 in
  (if *result* = *endOfLine*
   then *do-skip*(*tid*) *handler* 452
   else skip;
   deallocate-abstract-storage(*loc*)) 337

do-skip-rest : *Table-id* × *Variable* → *Exception-handler* $\xrightarrow{o}$ *Enumeration-value*

do-skip-rest (*tid*, *ch*)*handler* $\triangleq$
  def *res* = *do-skiplook*(*tid*, *ch*) *handler* in 452
  if *res* = *allRight*
  then *do-skip-rest*(*tid*, *ch*) *handler* 438
  else return *res*

annotations Skipping a line is modelled as *Look*, followed by repeated calls of *SkipLook* if a character was seen, and a final *Skip* if a line mark is seen. Other implementation strategies are possible, for example repeated calls of *TextRead*, into an array of the character-type, until no characters are read, followed by a final *Skip* if no characters were read because the end of line has been reached.

The procedure `WriteChar`

The call `WriteChar(cid,ch)` shall write the character `ch` to the output stream identified by `cid`.

WriteChar : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

WriteChar (*args*)*handler* $\triangleq$
  let [*cid*, *ch*] = *args* in
  def *tid* = *table-id*(*cid*) *handler* in 539
  write-chars(*tid*, [*ch*]) *handler* 453

The procedure `WriteLn`

The call `WriteLn(cid)` shall write a line mark to the output stream identified by `cid`.

WriteLn : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

WriteLn (*args*)*handler* $\triangleq$
  let [*cid*] = *args* in
  def *tid* = *table-id*(*cid*) *handler* in 539
  write-eol(*tid*) *handler* 452

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The procedure `WriteString`

The call `WriteString(cid,s)` shall write the string value in `s` to the output stream identified by `cid`.

WriteString : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()
WriteString (*args*)*handler* $\triangleq$
  let [*cid*, *s*] = *args* in
  def *tid* = *table-id*(*cid*) *handler* in 539
  write-chars(*tid*, *pure-string* (*s*)) *handler* 453 579

9.2.2.2 The Module WholeIO

The module `WholeIO` provides facilities that allow for the input and output of whole numbers in decimal text form.

9.2.2.2.1 The Definition Module of WholeIO

```
DEFINITION MODULE WholeIO;

  (* Input and output of whole numbers in decimal text form over specified channels.
     The read result is of the type IOConsts.ReadResults.
  *)

IMPORT IOChan;

  (* The text form of a signed whole number is
       ["+" | "-"], decimal digit, {decimal digit}

     The text form of an unsigned whole number is
       decimal digit, {decimal digit}
  *)

PROCEDURE ReadInt (cid: IOChan.ChanId; VAR int: INTEGER);
  (* Skips leading spaces, and removes any remaining characters from cid that form part of a
     signed whole number.  The value of this number is assigned to int.  The read result is
     set to the value allRight, outOfRange, wrongFormat, endOfLine, or endOfInput.
  *)

PROCEDURE WriteInt (cid: IOChan.ChanId; int: INTEGER; width: CARDINAL);
  (* Writes the value of int to cid in text form, in a field of the given minimum width. *)

PROCEDURE ReadCard (cid: IOChan.ChanId; VAR card: CARDINAL);
  (* Skips leading spaces, and removes any remaining characters from cid that form part of
     an unsigned whole number.  The value of this number is assigned to card.  The read
     result is set to the value allRight, outOfRange, wrongFormat, endOfLine, or endOfInput.
  *)

PROCEDURE WriteCard (cid: IOChan.ChanId; card: CARDINAL; width: CARDINAL);
  (* Writes the value of card to cid in text form, in a field of the given minimum width. *)

END WholeIO.
```

9.2.2.2.2 The Procedures of WholeIO

The text form of a signed whole number shall be

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
["+" | "-"], decimal digit, {decimal digit}
```

isa-signed-number : $Char^* \rightarrow \mathbb{B}$

isa-signed-number $(s) \triangleq$
  let *sign-digits* = *no-leading-spaces* (s) in
  let *digits* = *no-leading-sign* (*sign-digits*) in
  $\forall\, d \in$ elems *digits* · *isa-numeral* (d)

annotations Return *true* if the sequence is a signed number *or could form part of a signed number*. Return *false* iff the sequence could not be a signed number.

The text form of an unsigned whole number shall be

```
decimal digit, {decimal digit}
```

isa-number : $Char^* \rightarrow \mathbb{B}$

isa-number $(s) \triangleq$
  let *digits* = *no-leading-spaces* (s) in
  $\forall\, d \in$ elems *digits* · *isa-numeral* (d)

annotations Return *true* if the sequence is an unsigned number *or could form part of an unsigned number*. Return *false* iff the sequence could not be an unsigned number.

The procedure `ReadInt`

The call `ReadInt(cid,int)` shall skip any leading spaces from the input stream identified by `cid`, and then read such characters as form part of a signed whole number. The read result for the channel shall be set to the value

`allRight`	if a signed whole number is read, and its value is in the range of the type `INTEGER` —the value of this number shall be assigned to `int`;
`outOfRange`	if a signed whole number is read, but its value is out of range of the type `INTEGER` —the value `MAX(INTEGER)` or `MIN(INTEGER)` shall be assigned to `int` according to the sign of the number;
`wrongFormat`	if there are characters read or to be read, but these are not in the format of a signed whole number —the value of `int` is not defined;
`endOfLine`	if no characters are read, the next item being a line mark —the value of `int` is not defined;
`endOfInput`	if no characters are read, the input having ended —the value of `int` is not defined.

ReadInt : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

ReadInt (*args*)*handler* $\triangleq$
  let $[cid, int] = args$ in
  def *tid* = *table-id*(*cid*) *handler* in
  def *sps-number* = *read-while*(*tid*, $\lambda\, x : Char^* \cdot$ *isa-signed-number*(x)) in
  let *number* = *no-leading-spaces* (*sps-number*) in
  let *mk-* (*value*, *result*) = if *number* = [ ]
      then def *oldres* = *rdresult*(*tid*) in let *newres* = if *oldres* = *allRight*
          then WRONG-FORMAT
          else nil in
        mk- (nil , *newres*)
      else *strint* (*number*) in
  (if *value* = nil
   then let *impdep* : ($\mathbb{Z}$-TYPE | UNDEFINED) in
        assign(*int*, *impdep*) *handler*
   else *assign*(*int*, *value*) *handler* ;
   if *result* ≠ nil
   then *set-rdresult*(*tid*, *rdenum* (*result*))

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  else skip)
```

The procedure `WriteInt`

The call `WriteInt(cid,int,width)` shall write the value of `int` to the output stream identified by `cid` in text form, with leading spaces as required to make the number of characters written at least that given by `width`. A sign shall be written only for negative values. In the special case of a value of zero for `width`, exactly one leading space shall be written.

```
WriteInt : Arguments → Exception-handler →o ()
WriteInt (args)handler ≙
  let [cid, int, width] = args in
  let number = intstr (int) in                                          585
  let s = if width = 0
          then spaces (1) ↷ number                                      451
          elseif len number ≥ width
          then number
          else spaces (width − len number) ↷ number in                  451
  def tid = table-id(cid) handler in                                    539
  write-chars(tid, s)                                                   453
```

The procedure `ReadCard`

The call `ReadCard(cid,card)` shall skip any leading spaces from the input stream identified by `cid`, and then read such characters as form part of an unsigned whole number. The read result for the channel shall be set to the value

`allRight`	if an unsigned whole number is read, and its value is in the range of the type `CARDINAL` —the value of the number shall be assigned to `card`;
`outOfRange`	if an unsigned whole number is read, but its value is out of range of the values of the type `CARDINAL` —the value `MAX(CARDINAL)` shall be assigned to `card`;
`wrongFormat`	if there are characters read or to be read, but these are not in the format of an unsigned whole number —the value of `card` is not defined;
`endOfLine`	if no characters are read, the next item being a line mark —the value of `card` is not defined;
`endOfInput`	if no characters are read, the input having ended —the value of `card` is not defined.

```
ReadCard : Arguments → Exception-handler →o ()
ReadCard (args)handler ≙
  let [cid, card] = args in
  def tid = table-id(cid) handler in                                    539
  def sps-number = read-while(tid, λ x : Char* · isa-number(x)) in      452
  let number = no-leading-spaces (sps-number) in                        451
  let mk-(value, result) = if digits = []
                           then def oldres = rdresult(tid) in let newres = if oldres = allRight   539
                                                                           then WRONG-FORMAT
                                                                           else nil in
                                mk-(nil , newres)
                           else strcard (number) in                     585
  (if value = nil
   then let impdep : (ℤ-TYPE | UNDEFINED) in
        assign(card, impdep) handler                                    150
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

else *assign*(*card*, *value*) *handler* ; 150
if *result* ≠ nil
then *set-rdresult*(*tid*, *rdenum* (*result*)) 539 449
else skip)

The procedure `WriteCard`

The call `WriteCard(cid,card,width)` shall write the value of `card` to the output stream identified by `cid` in text form, with leading spaces as required to make the number of characters written at least that given by `width`. In the special case of a value of zero for `width`, exactly one leading space shall be written.

WriteCard : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()
WriteCard (*args*)*handler* ≙
let [*cid*, *card*, *width*] = *args* in
let *number* = *cardstr* (*card*) in 586
let *s* = if *width* = 0
then *spaces* (1) ↷ *number* 451
elseif len *number* ≥ *width*
then *number*
else *spaces* (*width* − len *number*) ↷ *number* in 451
def *tid* = *table-id*(*cid*) *handler* in 539
write-chars(*tid*, *s*) 453

9.2.2.3 The Modules RealIO and LongIO

The modules `RealIO` and `LongIO` provide facilities that allow for the input and output of real numbers in decimal text form.

In the case of `RealIO`, real number parameters are of the type `REAL`. In the case of `LongIO`, real number parameters are of the type `LONGREAL`.

9.2.2.3.1 The Definition Module of RealIO

```
DEFINITION MODULE RealIO;

  (* Input and output of real numbers in decimal text form over specified channels.
     The read result is of the type IOConsts.ReadResults.
  *)

IMPORT IOChan;

  (* The text form of a signed fixed-point real number is
       ["+" | "-"], decimal digit, {decimal digit},
       [".", {decimal digit}]

     The text form of a signed floating-point real number is
       signed fixed-point real number,
       "E"|"e", ["+" | "-"], decimal digit, {decimal digit}
  *)

PROCEDURE ReadReal (cid: IOChan.ChanId; VAR real: REAL);
  (* Skips leading spaces, and removes any remaining characters from cid that form part of a
     signed fixed or floating point number.  The value of this number is assigned to real.
     The read result is set to the value allRight, outOfRange, wrongFormat, endOfLine, or
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
    endOfInput.
  *)

PROCEDURE WriteFloat (cid: IOChan.ChanId; real: REAL; sigFigs: CARDINAL; width: CARDINAL);
  (* Writes the value of real to cid in floating-point text form, with sigFigs significant
     figures, in a field of the given minimum width.
  *)

PROCEDURE WriteEng (cid: IOChan.ChanId; real: REAL; sigFigs: CARDINAL; width: CARDINAL);
  (* As for WriteFloat, except that the number is scaled with one to three digits in the
     whole number part, and with an exponent that is a multiple of three.
  *)

PROCEDURE WriteFixed (cid: IOChan.ChanId; real: REAL; place: INTEGER; width: CARDINAL);
  (* Writes the value of real to cid in fixed-point text form, rounded to the given place
     relative to the decimal point, in a field of the given minimum width.
  *)

PROCEDURE WriteReal (cid: IOChan.ChanId; real: REAL; width: CARDINAL);
  (* Writes the value of real to cid, as WriteFixed if the sign and magnitude can be shown
     in the given width, or otherwise as WriteFloat.  The number of places or significant
     digits depends on the given width.
  *)

END RealIO.
```

9.2.2.3.2 The Definition Module of LongIO

```
DEFINITION MODULE LongIO;

  (* Input and output of long real numbers in decimal text form over specified channels.
     The read result is of the type IOConsts.ReadResults.
  *)

IMPORT IOChan;

  (* The text form of a signed fixed-point real number is
       ["+" | "-"], decimal digit, {decimal digit}, [".", {decimal digit}]

     The text form of a signed floating-point real number is
       signed fixed-point real number,
       "E"|"e", ["+" | "-"], decimal digit, {decimal digit}
  *)

PROCEDURE ReadReal (cid: IOChan.ChanId; VAR real: LONGREAL);
  (* Skips leading spaces, and removes any remaining characters from cid that form part of a
     signed fixed or floating point number.  The value of this number is assigned to real.
     The read result is set to the value allRight, outOfRange, wrongFormat, endOfLine, or
     endOfInput.
  *)

PROCEDURE WriteFloat (cid: IOChan.ChanId; real: LONGREAL; sigFigs: CARDINAL; width: CARDINAL);
  (* Writes the value of real to cid in floating-point text form, with sigFigs significant
     figures, in a field of the given minimum width.
  *)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
PROCEDURE WriteEng (cid: IOChan.ChanId; real: LONGREAL; sigFigs: CARDINAL; width: CARDINAL);
  (* As for WriteFloat, except that the number is scaled with one to three digits in the
     whole number part, and with an exponent that is a multiple of three.
  *)

PROCEDURE WriteFixed (cid: IOChan.ChanId; real: LONGREAL; place: INTEGER; width: CARDINAL);
  (* Writes the value of real to cid in fixed-point text form, rounded to the given place
     relative to the decimal point, in a field of the given minimum width.
  *)

PROCEDURE WriteReal (cid: IOChan.ChanId; real: LONGREAL; width: CARDINAL);
  (* Writes the value of real to cid, as WriteFixed if the sign and magnitude can be shown
     in the given width, or otherwise as WriteFloat.  The number of places or significant
     digits depends on the given width.
  *)

END LongIO.
```

9.2.2.3.3 The Procedures of RealIO and LongIO

The semantics of the two modules shall be the same, except that when module `RealIO` refers to real number values, these values shall be of the type `REAL`, and when module `LongIO` refers to real number values, these values shall be of the type `LONGREAL`.

NOTE 1 — The above statement is merely to avoid needless repetition of the semantics for the two modules.

The text form of a signed fixed-point real number shall be

```
["+" | "-"], decimal digit, {decimal digit},
[".", {decimal digit}]
```

The text form of a signed floating-point real number shall be

```
signed fixed-point real number,
"E"|"e", ["+" | "-"], decimal digit, {decimal digit}
```

$isa\text{-}real\text{-}number : Char^* \rightarrow \mathbb{B}$

$isa\text{-}real\text{-}number\,(s) \triangleq$
  let $number = no\text{-}leading\text{-}spaces\,(s)$ in 451
  let $rest1 = no\text{-}leading\text{-}sign\,(number)$ in 451
  let $whole, rest2 : Char^*$ be st $whole \frown rest2 = rest1 \wedge rest2 = no\text{-}leading\text{-}digits\,(rest1)$ in 451
  let $rest3 = no\text{-}leading\text{-}char\,(rest2, decimal\text{-}point\text{-}character)$ in 451
  let $rest4 = no\text{-}leading\text{-}digits\,(rest3)$ in 451
  let $rest5 = no\text{-}leading\text{-}exponent\,(rest4)$ in 451
  let $exp = no\text{-}leading\text{-}sign\,(rest5)$ in 451
  $rest1 = [] \vee (whole \neq [] \wedge \forall\, d \in \text{elems } exp \cdot isa\text{-}numeral\,(d))$ 451

annotations Return *true* if the sequence is a real number *or could form part of a real number*. Return *false* iff the sequence could not be a real number.

The procedure `ReadReal`

The call `ReadReal(cid,real)` shall skip any leading spaces from the input stream identified by `cid`, and then read such characters as form part of a signed fixed or floating point number. The read result for the channel shall be set to the value

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

`allRight`	if a signed real number is read, and its value is in the range of the type of `real` —the value of this number shall be assigned to `real`;
`outOfRange`	if a signed real number is read, but its value is out of range of the type of `real` —the maximum or minimum value of the type of `real` shall be assigned to `real` according to the sign of the number;
`wrongFormat`	if there are characters read or to be read, but these are not in the format of a signed real number —the value of `real` is not defined;
`endOfLine`	if no characters are read, the next item being a line mark —the value of `real` is not defined;
`endOfInput`	if no characters are read, the input having ended —the value of `real` is not defined.

$ReadReal : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} ()$

$ReadReal\,(args)handler \triangleq$
  let $[cid, real] = args$ in
  def $tid = table\text{-}id(cid)\ handler$ in 539
  def $sps\text{-}number = read\text{-}while(tid, \lambda\, x : Char^* \cdot isa\text{-}real\text{-}number(x))$ in 452
  let $number = no\text{-}leading\text{-}spaces\ (sps\text{-}number)$ in 451
  let $mk\text{-}(value, result) =$ if $number = []$
    then def $oldres = rdresult(tid)$ in let $newres =$ if $oldres = allRight$ 539
      then WRONG-FORMAT
      else nil in
      $mk\text{-}(\text{nil}, newres)$
    else $strreal\,(number)$ in 589
  (if $value = \text{nil}$
   then let $impdep$: (ℝ-TYPE | UNDEFINED) in
     $assign(real, impdep)\ handler$ 150
   else $assign(real, value)\ handler$; 150
   if $result \neq \text{nil}$
   then $set\text{-}rdresult(tid, rdenum\,(result))$ 539 449
   else skip)

The procedure `WriteFloat`

The call `WriteFloat(cid,real,sigFigs,width)` shall write the value of `real` to the output stream identified by `cid` in floating-point text form, with leading spaces as required to make the number of characters written at least that given by `width`. A sign shall be written only for negative values. In the special case of a value of zero for `width`, exactly one leading space shall be written.

One significant digit shall be included in the whole number part. The signed exponent part shall be included only if the exponent value is not zero. If the value of `sigFigs` is greater than zero, that number of significant digits shall be included, otherwise an implementation-defined number of significant digits shall be included. The decimal point shall not be included if there are no significant digits in the fractional part.

NOTE 2 — The following table gives examples of output by `WriteFloat`.

`sigFigs`	3923009	39.23009	0.0003923009
1	4E+6	4E+1	4E-4
2	3.9E+6	3.9E+1	3.9E-4
5	3.9230E+6	3.9230E+1	3.9230E-4

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

WriteFloat : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

WriteFloat (*args*)*handler* $\triangleq$
  let [*cid*, *real*, *sigFigs*, *width*] = *args* in
  let *number* = *floatstr* (*real*, *sigFigs*) in 590
  let *s* = if *width* = 0
        then *spaces* (1) $\curvearrowright$ *number* 451
        elseif len *number* ≥ *width*
        then *number*
        else *spaces* (*width* − len *number*) $\curvearrowright$ *number* in 451
  def *tid* = *table-id*(*cid*) *handler* in 539
  write-chars(*tid*, *s*) 453

The procedure `WriteEng`

The call `WriteEng(cid,real,sigFigs,width)` shall behave as the call `WriteFloat(cid,real,sigFigs,width)` except that the number shall be scaled with one to three digits in the whole number part, and with an exponent that is a multiple of three.

NOTE 3 — The following table gives examples of output by `WriteEng`.

sigFigs	3923009	39.23009	0.0003923009
1	4E+6	40	400E-6
2	3.9E+6	39	390E-6
5	3.9230E+6	39.230	392.30E-6

WriteEng : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

WriteEng (*args*)*handler* $\triangleq$
  let [*cid*, *real*, *sigFigs*, *width*] = *args* in
  let *number* = *engstr* (*real*, *sigFigs*) in 590
  let *s* = if *width* = 0
        then *spaces* (1) $\curvearrowright$ *number* 451
        elseif len *number* ≥ *width*
        then *number*
        else *spaces* (*width* − len *number*) $\curvearrowright$ *number* in 451
  def *tid* = *table-id*(*cid*) *handler* in 539
  write-chars(*tid*, *s*) 453

The procedure `WriteFixed`

The call `WriteFixed(cid,real,place,width)` shall write the value of `real` to the output stream identified by `cid` in fixed-point text form with leading spaces as required to make the number of characters written at least that given by `width`. A sign shall be written only for negative values. In the special case of a value of zero for `width`, exactly one leading space shall be written.

At least one digit shall be included in the whole number part. The value shall be rounded to the given value of `place` relative to the decimal point. The decimal point shall be suppressed if `place` is less than zero.

NOTE 4 — The following table gives examples of output by `WriteFixed`.

places	3923009	39.23009	0.0003923009
-5	3920000	0	0
-2	3923010	40	0
-1	3923009	39	0
0	3923009.	39.	0.
1	3923009.0	39.2	0.0
4	3923009.0000	39.2301	0.0004

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

WriteFixed : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()
WriteFixed (*args*)*handler* $\triangleq$
  let [*cid*, *real*, *place*, *width*] = *args* in
  let *number* = *fixedstr* (*real*, *place*) in 591
  let *s* = if *width* = 0
        then *spaces* (1) $\curvearrowright$ *number* 451
        elseif len *number* ≥ *width*
        then *number*
        else *spaces* (*width* − len *number*) $\curvearrowright$ *number* in 451
  def *tid* = *table-id*(*cid*) *handler* in 539
  write-chars(*tid*, *s*) 453

The procedure `WriteReal`

If the sign and magnitude of `real` can be expressed in a field given by `width`, the call `WriteReal(cid,real,width)` shall behave as the call `WriteFixed(cid,real,place,width)`, with a value of `place` chosen to fill exactly the remaining field. Otherwise the call shall behave as the call `WriteFloat(cid,real,sigFigs,width)`, with a value of `sigFigs` of at least one, limited to those that can be included together with the sign and exponent part in the given width.

In the special case of a width of zero, the effect shall be as for a call of `WriteFloat(cid,real,0,0)`.

WriteReal : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()
WriteReal (*args*)*handler* $\triangleq$
  let [*cid*, *real*, *width*] = *args* in
  let *number* = *realstr* (*real*, *width*) in 591
  let *s* = if *width* = 0
        then *spaces* (1) $\curvearrowright$ *number* 451
        elseif len *number* ≥ *width*
        then *number*
        else *spaces* (*width* − len *number*) $\curvearrowright$ *number* in 451
  def *tid* = *table-id*(*cid*) *handler* in 539
  write-chars(*tid*, *s*) 453

9.2.2.4 The Module RawIO

The module `RawIO` provides facilities that allow for for the direct input and output of data using raw operations (i.e. without any interpretation).

9.2.2.4.1 The Definition Module of RawIO

```
DEFINITION MODULE RawIO;

  (* Reading and writing data over specified channels using raw operations, that is, with no
     conversion or interpretation. The read result is of the type IOConsts.ReadResults.
  *)

IMPORT IOChan, SYSTEM;

PROCEDURE Read (cid: IOChan.ChanId; VAR to: ARRAY OF SYSTEM.LOC);
  (* Reads storage units from cid, and assigns them to successive components of to. The read
     result is set to the value allRight, wrongFormat, or endOfInput.
  *)

PROCEDURE Write (cid: IOChan.ChanId; from: ARRAY OF SYSTEM.LOC);
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  (* Writes storage units to cid from successive components of from. *)

END RawIO.
```

9.2.2.4.2 The Procedures of RawIO

The procedure `Read`

While the stream identified by `cid` is not exhausted, the call `Read(cid,to)` shall read successive storage units from that channel, and assign them without interpretation to successive components of `to`. The read result for the channel shall be set to the value

`allRight`	if items are read for all components;
`wrongFormat`	if some items are read, but not for all components;
`endOfInput`	if no items are read, the input having ended.

Read : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

Read (*args*)*handler* $\triangleq$
  let [*cid*, *to*] = *args* in
  def *tid* = *table-id*(*cid*) *handler* in 539
  def *nlocs* = *read-locs*(*tid*, *to*) *handler* in 453
  if *nlocs* < *capacity* (*to*) ∧ *nlocs* > 0 579
  then *set-rdresult*(*tid*, *wrongFormat*) 539
  else skip

The procedure `Write`

The call `Write(cid,from)` shall write successive components of `from` to the channel identified by `cid`, as storage units without interpretation.

Write : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

Write (*args*)*handler* $\triangleq$
  let [*cid*, *from*] = *args* in
  def *tid* = *table-id*(*cid*) *handler* in 539
  write-locs(*tid*, *map-to-seq* (*from*)) *handler* 454 615

9.2.2.5 The Module IOConsts

The module `IOConsts` defines the enumeration type used to express read results. Programs do not normally need to import from `IOConsts` directly, since client modules define identifiers that correspond to those defined by this module.

9.2.2.5.1 The Definition Module of IOConsts

```
DEFINITION MODULE IOConsts;

  (* Types and constants for input/output modules *)

TYPE
  ReadResults =  (* This type is used to classify the result of an input operation *)
  (
    notKnown,    (* no read result is set *)
    allRight,    (* data is as expected or as required *)
    outOfRange,  (* data cannot be represented *)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
    wrongFormat, (* data not in expected format *)
    endOfLine,   (* end of line seen before expected data *)
    endOfInput   (* end of input seen before expected data *)
  );

END IOConsts.
```

9.2.2.5.2 Auxiliary Formal Model Definitions

The following definitions give the enumeration values from `IOConsts` used in the formal model.

values

readresults = [`notKnown`, `allRight`, `outOfRange`, `wrongFormat`, `endOfLine`, `endOfInput`];

notKnown = *mk-Enumeration-value* (`notKnown`, *readresults*);

allRight = *mk-Enumeration-value* (`allRight`, *readresults*);

outOfRange = *mk-Enumeration-value* (`outOfRange`, *readresults*);

wrongFormat = *mk-Enumeration-value* (`wrongFormat`, *readresults*);

endOfLine = *mk-Enumeration-value* (`endOfLine`, *readresults*);

endOfInput = *mk-Enumeration-value* (`endOfInput`, *readresults*)

rdenum : *Conv-result* → *Enumeration-value*

rdenum (*cr*) ≜

cases *cr* :
  ALL-RIGHT → *allRight*,
  OUT-OF-RANGE → *outOfRange*,
  WRONG-FORMAT→ *wrongFormat*
end

annotations Interpret a numeric conversion result as the corresponding read result.

9.2.2.6 The Module IOResult

The module `IOResult` provides the facility for a program to determine whether the last operation to read data from a specified input channel found data in the required format.

9.2.2.6.1 The Definition Module of IOResult

```
DEFINITION MODULE IOResult;

  (* Read results for specified channels *)

IMPORT IOConsts, IOChan;

TYPE
  ReadResults = IOConsts.ReadResults;

  (*
    ReadResults =  (* This type is used to classify the result of an input operation *)
    (
      notKnown,    (* no read result is set *)
      allRight,    (* data is as expected or as required *)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
      outOfRange,  (* data cannot be represented *)
      wrongFormat, (* data not in expected format *)
      endOfLine,   (* end of line seen before expected data *)
      endOfInput   (* end of input seen before expected data *)
    );
  *)

PROCEDURE ReadResult (cid: IOChan.ChanId): ReadResults;
  (* Returns the result for the last read operation on the channel cid. *)

END IOResult.
```

9.2.2.6.2 The Procedures of IOResult

The procedure `ReadResult`

The value of the call `ReadResult(cid)` shall be the stored read result for the channel identified by `cid`.

IOResult-ReadResult : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Enumeration-value*
IOResult-ReadResult (*args*)*handler* $\triangleq$
  let [*cid*] = *args* in
  def *tid* = *table-id*(*cid*) *handler* in 539
  rdresult(*tid*) 539

9.2.2.7 Auxiliary Formal Model Definitions

The following values and functions are used in the conversion operations

values

decimal-point-character = '.';

exponent-character = '*E*';

alternative-exponent-character = '*e*'

isa-space : *Char* → $\mathbb{B}$
isa-space (*c*) $\triangleq$
  c = *space-character*

annotations Return whether *c* is a space character

split : *Char** × (*Char** → $\mathbb{B}$) → *Char**
split (*s*, *f*) $\triangleq$
  let *first*, *rest* : *Char** be st *first* $\frown$ *rest* = *s* ∧
    f(*first*) ∧ (*rest* = [] ∨ ¬ *f*(*first* $\frown$ [hd *rest*]) in
  rest

annotations Return the shortest trailing substring of *s* for which the leading substring of *s* satisfies *f*

is-optspaces : *Char** → $\mathbb{B}$
is-optspaces (*s*) $\triangleq$
  elems *s* = {*space-character*} ∨ *s* = []

annotations Return whether *s* is empty or consists entirely of space characters.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

is-optdigits : $Char^* \to \mathbb{B}$

is-optdigits $(s) \triangleq$
  $\forall c \in \mathsf{elems}\ s \cdot c \in digits$

annotations Return whether s is empty or consists entirely of digits

is-optsign : $Char^* \to \mathbb{B}$

is-optsign $(s) \triangleq$
  $s = [] \vee (\mathsf{len}\ s = 1 \wedge isa\text{-}sign\ (\mathsf{hd}\ s))$ 605

annotations Return whether s is empty or consists of a single sign character.

no-leading-spaces : $Char^* \to Char^*$

no-leading-spaces $(s) \triangleq$
  split $(s, is\text{-}optspaces)$ 450

annotations Return s stripped of any leading space characters.

no-leading-sign : $Char^* \to Char^*$

no-leading-sign $(s) \triangleq$
  split $(s, is\text{-}optsign)$ 450

annotations Return s stripped of any leading sign character.

no-leading-char : $Char^* \times Char \to Char^*$

no-leading-char $(s, c) \triangleq$
  split $(s, \lambda\, x : Char^* \cdot x = [] \vee x = [c])$ 450

annotations Return s stripped of any leading character c.

no-leading-exponent : $Char^* \to Char^*$

no-leading-exponent $(s) \triangleq$
  split $(s, \lambda\, x : Char^* \cdot x = [] \vee x = [exponent\text{-}character] \vee x = [alternative\text{-}exponent\text{-}character])$ 450

annotations Return s stripped of any leading exponent character.

no-leading-digits : $Char^* \to Char^*$

no-leading-digits $(s) \triangleq$
  split $(s, is\text{-}optdigits)$ 450

annotations Return s stripped of any leading digits.

isa-numeral : $Char \to \mathbb{B}$

isa-numeral $(c) \triangleq$
  $c \in digits$

annotations Is the character a numeral.

spaces $(n : \mathbb{N})ans : Char^*$
post let $s : Char^*$ be st $\mathsf{len}\ s = n \wedge \mathsf{elems}\ s = \{space\text{-}character\}$ in
    $ans = s$

annotations Return a sequence of n space characters.

The following operations model the use of the operations provided by a device associated with a channel

do-look : $Table\text{-}id \times Variable \to Exception\text{-}handler \xrightarrow{o} Enumeration\text{-}value$

do-look $(tid, ch)handler \triangleq$
  def $pid = device\text{-}op(tid, \mathrm{LOOK})$ in 539
  def $loc = allocate\text{-}abstract\text{-}storage()$ in 337

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  let res = mk-Elementary-variable (loc, nil ) in
  (call-proper-procedure-value(pid) ([tid, ch, res]) handler;                    339
   def result = contents(loc) in                                                 335
   (deallocate-abstract-storage(loc) ;                                           337
    return result ))
```

annotations do-look models the call of the *Look* operation for the device table *tid*, using a supplied variable of the character type, a local elementary variable, and returning the read result as a function result.

```
do-skip : Table-id → Exception-handler →o ()
do-skip (tid)handler ≙
  def pid = device-op(tid, SKIP) in                                              539
  call-proper-procedure-value(pid) ([tid]) handler;                              339
```

annotations do-skip models the call of the *Skip* operation for the device table *tid*.

```
do-skiplook : Table-id × Variable → Exception-handler →o Enumeration-value
do-skiplook (tid, ch)handler ≙
  def pid = device-op(tid, SKIPLOOK) in                                          539
  def loc = allocate-abstract-storage() in                                       337
  let res = mk-Elementary-variable (loc, nil ) in
  (call-proper-procedure-value(pid) ([tid, ch, res]) handler;                    339
   def result = contents(loc) in                                                 335
   (deallocate-abstract-storage(loc) ;                                           337
    return result ))
```

annotations do-skiplook models the call of the *SkipLook* operation for the device table *tid*, using a supplied variable of the character type, a local elementary variable, and returning the read result as a function result.

```
read-while : Table-id × (Char* → 𝔹) → Exception-handler →o Char*
read-while (tid, f)handler ≙
  read-until-false([], tid, f) handler                                           452
```

annotations Return the longest sequence of characters removed from the current line for which f is true.

```
read-until-false : Char* × Table-id × (Char* → 𝔹) → Exception-handler →o Char*
read-until-false (ins, tid, continue-reading)handler ≙
  def loc = allocate-abstract-storage() in                                       337
  let chv = mk-Elementary-variable (loc, nil ) in
  def res = do-look(tid, chv) handler in                                         451
  def ch = contents(loc) in                                                      335
  (deallocate-abstract-storage(loc) ;                                            337
   if res = allRight
   then let ans = ins ↷ [ch] in
        if continue-reading (ans)                                                452
        then (do-skip(tid) handler;                                              452
              read-until-false(ans, tid, continue-reading) handler )             452
        else return ins
   else return ins )
```

```
write-eol : Table-id → Exception-handler →o ()
write-eol (tid)handler ≙
  def pid = device-op(tid, WRITELN) in                                           539
  call-proper-procedure-value(pid) ([tid]) handler                               339
```

annotations Write an end of line marker using the *WriteLn* device operation.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

read-chars : *Table-id* × *Array-variable* → *Exception-handler* $\xrightarrow{o}$ N

read-chars (*tid*, *to*)*handler* $\triangleq$
  def *pid* = *device-op*(*tid*, TEXTREAD) in 539
  let *max* = *capacity* (*to*) in 579
  def *loc* = *allocate-abstract-storage*() in 337
  let *charsRead* = *mk-Elementary-variable* (*loc*, nil) in
  (*assign*(*charsRead*, 0) *handler*; 150
   getchars(*tid*, *pid*, *to*, 0, *max*, *charsRead*) *handler*; 453
   def *n* = *contents*(*loc*) in 335
   (*deallocate-abstract-storage*(*loc*) ; 337
    return *n*))

getchars : *Table-id* × *Procedure-id* × *Array-variable* × N × N × *Variable* → *Exception-handler* $\xrightarrow{o}$ ()

getchars (*tid*, *pid*, *to*, *offset*, *limit*, *nRead*)*handler* $\triangleq$
  def *sofar* = *value-of-a-variable*(*nRead*) in 224
  (def *ato* = *address-of* (*to* (*offset*), CHARACTER-TYPE) ρ_{PERV} in 360
   call-proper-procedure-value(*pid*) ([*tid*, *ato*, *limit*, *nRead*]) *handler*; 339
   def *thisN* = *value-of-a-variable*(*nRead*) in 224
   (*assign*(*nRead*, *thisN* + *sofar*) *handler*; 150
    if (*thisN* = 0) ∨ ((*thisN* + *sofar*) = *limit*)
    then skip
    else *getchars*(*tid*, *pid*, *to*, *offset* + *thisN*, *max* − *thisN*, *nRead*) *handler*)) 453

annotations Call a device *TextRead* procedure, repeatedly if some, but not enough, data returned.

write-chars : *Table-id* × *Char** → *Exception-handler* $\xrightarrow{o}$ ()

write-chars (*tid*, *chars*)*handler* $\triangleq$
  def *pid* = *device-op*(*tid*, TEXTWRITE) in 539
  let *n* = len *chars* in
  let *sftyped*, *ctyped* : *Type-name* in
  let *structures* = {*ctyped* ↦ *mk-Subrange-structure* (UNSIGNED-TYPE, {0, . . . , *n* − 1},
                                sftyped ↦ *mk-Array-structure* (*ctyped*, CHARACTER-TYPE))} in
  let ρ = *overwrite-struc-environment* (*structures*)ρ_{PERV} in 315
  def *from* = *allocate-program-storage*(*sftyped*) ρ in 337
  (for all *i* ∈ inds *chars*
   do *assign*(*from* (*i* − 1), *chars* (*i*)) *handler* ; 150
   def *afrom* = *address-of* (*from* (0), CHARACTER-TYPE) ρ_{PERV} in 360
   call-proper-procedure-value(*pid*) ([*tid*, *afrom*, *n*]) *handler*; 339
   deallocate-program-storage(*from*)) 337

annotations Writing a character sequence is modelled as using the device *TextWrite* operation to write out the components from an array variable.

read-locs : *Table-id* × *Array-variable* → *Exception-handler* $\xrightarrow{o}$ N

read-locs (*tid*, *to*)*handler* $\triangleq$
  def *pid* = *device-op*(*tid*, BINREAD) in 539
  let *max* = *capacity* (*to*) in 579
  def *loc* = *allocate-abstract-storage*() in 337
  let *locsRead* = *mk-Elementary-variable* (*loc*, nil) in
  (*assign*(*locsRead*, 0) *handler*; 150
   getlocs(*tid*, *pid*, *to*, 0, *max*, *locsRead*) *handler*; 453
   def *n* = *contents*(*loc*) in 335
   (*deallocate-abstract-storage*(*loc*) ; 337
    return *n*))

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

getlocs : *Table-id* × *Procedure-id* × *Array-variable* × ℕ × ℕ × *Variable* → *Exception-handler* $\xrightarrow{o}$ ()

getlocs (*tid*, *pid*, *to*, *offset*, *limit*, *nRead*)*handler* $\triangleq$
  def *sofar* = *value-of-a-variable*(*nRead*) in 224
  (def *ato* = *address-of*(*to* (*offset*), LOC-TYPE) ρ_{EMPTY} in 360
   call-proper-procedure-value(*pid*) ([*tid*, *ato*, *limit*, *nRead*]) *handler*; 339
   def *thisN* = *value-of-a-variable*(*nRead*) in 224
   (*assign*(*nRead*, *thisN* + *sofar*) *handler*; ■ 150
    if (*thisN* = 0) ∨ ((*thisN* + *sofar*) = *limit*)
    then skip
    else *getlocs*(*tid*, *pid*, *to*, *offset* + *thisN*, *max* − *thisN*, *nRead*) *handler*)) 453

annotations Call a device *RawRead* procedure, repeatedly if some, but not enough, data returned.

write-locs : *Table-id* × *Elementary-LOC-value** → *Exception-handler* $\xrightarrow{o}$ ()

write-locs (*tid*, *locs*)*handler* $\triangleq$
  def *pid* = *device-op*(*tid*, BINWRITE) in 539
  let *n* = len *locs* in
  let *sftyped*, *ctyped* : *Type-name* in
  let *structures* = {*ctyped* ↦ *mk-Subrange-structure* (UNSIGNED-TYPE, {0, . . . , *n* − 1},
                                sftyped ↦ *mk-Array-structure* (*ctyped*, LOC-TYPE))} in
  let ρ_{ps} = *merge-environment* (ρ_{PERV}, ρ_{SYSTEM}) in 319
  let ρ = *overwrite-struc-environment* (*structures*)ρ_{ps} in 315
  def *from* = *allocate-program-storage*(*sftyped*) ρ in 337
  (for all *i* ∈ inds *locs*
   do *assign*(*var* (*i* − 1), *locs* (*i*)) *handler* ; 150
   def *afrom* = *address-of*(*from* (0), LOC-TYPE) ρ_{EMPTY} in 360
   call-proper-procedure-value(*pid*) ([*tid*, *afrom*, *n*]) *handler*; 339
   deallocate-program-storage(*from*)) 337

annotations Writing a location sequence is modelled as using the device *RawWrite* operation to write out the components from an array variable.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9.2.3 Reading and Writing of Data using Default Channels

The modules STextIO, SWholeIO, SRealIO, SLongIO, SRawIO, and SIOResult provide procedures that operate over default input and output channels, and so do not take a parameter identifying a channel.

9.2.3.1 The Module STextIO

The module STextIO corresponds to TextIO.

9.2.3.1.1 The Definition Module of STextIO

```
DEFINITION MODULE STextIO;

  (* Input and output of character and string types over default channels. The read result
     is of the type IOConsts.ReadResults.
  *)

  (* The following procedures do not read past line marks *)

PROCEDURE ReadChar (VAR ch: CHAR);
  (* If possible, removes a character from the default input stream, and assigns the
     corresponding value to ch.  The read result is set to allRight, endOfLine or
     endOfInput.
  *)

PROCEDURE ReadRestLine (VAR s: ARRAY OF CHAR);
  (* Removes any remaining characters from the default input stream before the next line
     mark, copying to s as many as can be accommodated as a string value.  The read result
     is set to the value allRight, outOfRange, endOfLine, or endOfInput.
  *)

PROCEDURE ReadString (VAR s: ARRAY OF CHAR);
  (* Removes only those characters from the default input stream before the next line mark
     that can be accommodated in s as a string value, and copies them to s.  The read result
     is set to the value allRight, endOfLine, or endOfInput.
  *)

PROCEDURE ReadToken (VAR s: ARRAY OF CHAR);
  (* Skips leading spaces, and then removes characters from the default input stream before
     the next space or line mark, copying to s as many as can be accommodated as a string
     value.  The read result is set to the value allRight, outOfRange, endOfLine, or
     endOfInput.
  *)

  (* The following procedure reads past the next line mark *)

PROCEDURE SkipLine;
  (* Removes successive items from the default input stream up to and including the next
     line mark or until the end of input is reached. The read result is set to the value
     allRight, or endOfInput.
  *)


  (* Output procedures *)
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
PROCEDURE WriteChar (ch: CHAR);
  (* Writes the value of ch to the default output stream. *)

PROCEDURE WriteLn;
  (* Writes a line mark to the default output stream. *)

PROCEDURE WriteString (s: ARRAY OF CHAR);
  (* Writes the string value of s to the default output stream. *)

END STextIO.
```

9.2.3.1.2 The Procedures of STextIO

The procedures of the module `STextIO` shall behave as the corresponding procedures of the module `TextIO`, except that input shall be taken from the default input channel, and output shall be sent to the default output channel.

9.2.3.2 The Module SWholeIO

The module `SWholeIO` corresponds to `WholeIO`.

9.2.3.2.1 The Definition Module of SWholeIO

```
DEFINITION MODULE SWholeIO;

  (* Input and output of whole numbers in decimal text form over default
     channels.  The read result is of the type IOConsts.ReadResults.
  *)

  (* The text form of a signed whole number is
       ["+" | "-"], decimal digit, {decimal digit}

     The text form of an unsigned whole number is
       decimal digit, {decimal digit}
  *)

PROCEDURE ReadInt (VAR int: INTEGER);
  (* Skips leading spaces, and removes any remaining characters from the default input
     channel that form part of a signed whole number.  The value of this number is assigned
     to int.  The read result is set to the value allRight, outOfRange, wrongFormat,
     endOfLine, or endOfInput.
  *)

PROCEDURE WriteInt (int: INTEGER; width: CARDINAL);
  (* Writes the value of int to the default output channel in text form, in a field of the
     given minimum width.
  *)

PROCEDURE ReadCard (VAR card: CARDINAL);
  (* Skips leading spaces, and removes any remaining characters from the default input
     channel that form part of an unsigned whole number.  The value of this number is
     assigned to card.  The read result is set to the value allRight, outOfRange,
     wrongFormat, endOfLine, or endOfInput.
  *)

PROCEDURE WriteCard (card: CARDINAL; width: CARDINAL);
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  (* Writes the value of card to the default output channel in text form, in a field of the
     given minimum width.
  *)

END SWholeIO.
```

9.2.3.2.2 The Procedures of SWholeIO

The procedures of the module SWholeIO shall behave as the corresponding procedures of the module WholeIO, except that input shall be taken from the default input channel, and output shall be sent to the default output channel.

9.2.3.3 The Modules SRealIO and SLongIO

The module SRealIO corresponds to RealIO and the module SLongIO corresponds to LongIO.

9.2.3.3.1 The Definition Module of SRealIO

```
DEFINITION MODULE SRealIO;

  (* Input and output of real numbers in decimal text form over default
     channels.  The read result is of the type IOConsts.ReadResults.
  *)

  (* The text form of a signed fixed-point real number is
       ["+" | "-"], decimal digit, {decimal digit},
       [".", {decimal digit}]

     The text form of a signed floating-point real number is
       signed fixed-point real number,
       "E"|"e", ["+" | "-"], decimal digit, {decimal digit}
  *)

PROCEDURE ReadReal (VAR real: REAL);
  (* Skips leading spaces, and removes any remaining characters from the default input
     channel that form part of a signed fixed or floating point number. The value of this
     number is assigned to real.  The read result is set to the value allRight, outOfRange,
     wrongFormat, endOfLine, or endOfInput.
  *)

PROCEDURE WriteFloat (real: REAL; sigFigs: CARDINAL; width: CARDINAL);
  (* Writes the value of real to the default output channel in floating-point text form,
     with sigFigs significant figures, in a field of the given minimum width.
  *)

PROCEDURE WriteEng (real: REAL; sigFigs: CARDINAL; width: CARDINAL);
  (* As for WriteFloat, except that the number is scaled with one to three digits in the
     whole number part, and with an exponent that is a multiple of three.
  *)

PROCEDURE WriteFixed (real: REAL; place: INTEGER; width: CARDINAL);
  (* Writes the value of real to the default output channel in fixed-point text form,
     rounded to the given place relative to the decimal point, in a field of the given
     minimum width.
  *)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
PROCEDURE WriteReal (real: REAL; width: CARDINAL);
  (* Writes the value of real to the default output channel, as WriteFixed if the sign and
     magnitude can be shown in the given width, or otherwise as WriteFloat.  The number of
     places or significant digits depends on the given width.
  *)

END SRealIO.
```

9.2.3.3.2 The Definition Module of SLongIO

```
DEFINITION MODULE SLongIO;

  (* Input and output of long real numbers in decimal text form using default channels.  The
     read result is of the type IOConsts.ReadResults.
  *)

  (* The text form of a signed fixed-point real number is
       ["+" | "-"], decimal digit, {decimal digit},
       [".", {decimal digit}]

     The text form of a signed floating-point real number is
       signed fixed-point real number,
       "E"|"e", ["+" | "-"], decimal digit, {decimal digit}
  *)

PROCEDURE ReadReal (VAR real: LONGREAL);
  (* Skips leading spaces, and removes any remaining characters from the default input
     channel that form part of a signed fixed or floating point number. The value of this
     number is assigned to real.  The read result is set to the value allRight, outOfRange,
     wrongFormat, endOfLine, or endOfInput.
  *)

PROCEDURE WriteFloat (real: LONGREAL; sigFigs: CARDINAL; width: CARDINAL);
  (* Writes the value of real to the default output channel in floating-point text form,
     with sigFigs significant figures, in a field of the given minimum width.
  *)

PROCEDURE WriteEng (real: LONGREAL; sigFigs: CARDINAL; width: CARDINAL);
  (* As for WriteFloat, except that the number is scaled with one to three digits in the
     whole number part, and with an exponent that is a multiple of three.
  *)

PROCEDURE WriteFixed (real: LONGREAL; place: INTEGER; width: CARDINAL);
  (* Writes the value of real to the default output channel in fixed-point text form,
     rounded to the given place relative to the decimal point, in a field of the given
     minimum width.
  *)

PROCEDURE WriteReal (real: LONGREAL; width: CARDINAL);
  (* Writes the value of real to the default output channel, as WriteFixed if the sign and
     magnitude can be shown in the given width, or otherwise as WriteFloat.  The number of
     places or significant digits depends on the given width.
  *)

END SLongIO.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9.2.3.3.3 The Procedures of SRealIO and SLongIO

The procedures of the module SRealIO shall behave as the corresponding procedures of the module RealIO, except that input shall be taken from the default input channel, and output shall be sent to the default output channel.

The procedures of the module SLongIO shall behave as the corresponding procedures of the module LongIO, except that input shall be taken from the default input channel, and output shall be sent to the default output channel.

9.2.3.4 The Module SRawIO

The module SRawIO corresponds to RawIO.

9.2.3.4.1 The Definition Module of SRawIO

```
DEFINITION MODULE SRawIO;

  (* Reading and writing data over default channels using raw operations, that is, with no
     conversion or interpretation. The read result is of the type IOConsts.ReadResults.
  *)

IMPORT SYSTEM;

PROCEDURE Read (VAR to: ARRAY OF SYSTEM.LOC);
  (* Reads storage units from the default input channel, and assigns them to successive
     components of to.  The read result is set to the value allRight, wrongFormat, or
     endOfInput.
  *)

PROCEDURE Write (from: ARRAY OF SYSTEM.LOC);
  (* Writes storage units to the default output channel from successive components of from.
  *)

END SRawIO.
```

9.2.3.4.2 The Procedures of SRawIO

The procedures of the module SRawIO shall behave as the corresponding procedures of the module RawIO, except that input shall be taken from the default input channel, and output shall be sent to the default output channel.

9.2.3.5 The Module SIOResult

The module SIOResult corresponds to IOResult.

9.2.3.5.1 The Definition Module of SIOResult

```
DEFINITION MODULE SIOResult;

  (* Read results for the default input channel *)

IMPORT IOConsts;

TYPE
  ReadResults = IOConsts.ReadResults;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  (*
    ReadResults =   (* This type is used to classify the result of an input operation *)
    (
      notKnown,     (* no read result is set *)
      allRight,     (* data is as expected or as required *)
      outOfRange,   (* data cannot be represented *)
      wrongFormat,  (* data not in expected format *)
      endOfLine,    (* end of line seen before expected data *)
      endOfInput    (* end of input seen before expected data *)
    );
  *)

PROCEDURE ReadResult (): ReadResults;
  (* Returns the result for the last read operation on the default input channel. *)

END SIOResult.
```

9.2.3.5.2 The Procedures of SIOResult

The procedure `ReadResult`

The value of the call `ReadResult()` shall be the stored read result for the default input channel.

SIOResult-ReadResult : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Enumeration-value*

SIOResult-ReadResult (*args*)*handler* $\triangleq$
  def *cid* = *iolib-defin* in
  def *tid* = *table-id*(*cid*) *handler* in 539
  rdresult(*tid*) 539

NOTE — The existence of the module `IOConsts` allows the definition module `SIOResult` to be independent of the modules `IOResult` and `IOChan`.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9.2.4 The Interface for Device-Independent Channel Operations

The module IOChan provides access to channel operations that are provided in a device-independent manner for all channels.

Device-dependent operations (which include operations for opening new channels and subsequently closing them) are defined in the definition module for each device.

9.2.4.1 The Module IOChan

The module IOChan defines the type ChanId that is used to identify channels, and provides facilities for device-independent access to operations supported by the device to which a channel is connected.

9.2.4.1.1 The Definition Module of IOChan

```
DEFINITION MODULE IOChan;

  (* Types and procedures forming the interface to channels for device-independent data
     transfer modules
  *)

IMPORT IOConsts, ChanConsts, SYSTEM;

TYPE
  ChanId; (* Values of this type are used to identify channels *)

  (* There is one distinguished value identifying an invalid channel on which no data transfer
     operations are available.  It may be used to initialize variables of type ChanId.
  *)

PROCEDURE InvalidChan (): ChanId;
  (* Returns the value identifying the invalid channel. *)

  (* For each of the following operations, if the device supports the operation on the
     channel, the behaviour of the procedure conforms with the description below.  The full
     behaviour is defined for each device module.  If the device does not support the
     operation on the channel, the behaviour of the procedure is to raise the exception
     notAvailable.
  *)

  (* Text operations - these perform any required translation between the internal and
     external representation of text.
  *)

PROCEDURE Look (cid: ChanId; VAR ch: CHAR; VAR res: IOConsts.ReadResults);
  (* If there is a character as the next item in the input stream cid, assigns its value to
     ch without removing it from the stream; otherwise the value of ch is not defined.  res
     (and the stored read result) are set to the value allRight, endOfLine, or endOfInput.
  *)

PROCEDURE Skip (cid: ChanId);
  (* If the input stream cid has ended, the exception skipAtEnd is raised; otherwise the
     next character or line mark in cid is removed, and the stored read result is set to the
     value allRight.
  *)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
PROCEDURE SkipLook (cid: ChanId; VAR ch: CHAR; VAR res: IOConsts.ReadResults);
  (* If the input stream cid has ended, the exception skipAtEnd is raised; otherwise the
     next character or line mark in cid is removed.  If there is a character as the next
     item in cid stream, assigns its value to ch without removing it from the stream.
     Otherwise, the value of ch is not defined.  res (and the stored read result) are set to
     the value allRight, endOfLine, or endOfInput.
  *)

PROCEDURE WriteLn (cid: ChanId);
  (* Writes a line mark over the channel cid. *)

PROCEDURE TextRead (cid: ChanId; to: SYSTEM.ADDRESS; maxChars: CARDINAL;
                    VAR charsRead: CARDINAL);
  (* Reads at most maxChars characters from the current line in cid, and assigns
     corresponding values to successive components of an ARRAY OF CHAR variable for which
     the address of the first component is to. The number of characters read is assigned
     to charsRead. The stored read result is set to allRight, endOfLine, or endOfInput.
  *)

PROCEDURE TextWrite (cid: ChanId; from: SYSTEM.ADDRESS; charsToWrite: CARDINAL);
  (* Writes a number of characters given by the value of charsToWrite, from successive
     components of an ARRAY OF CHAR variable for which the address of the first component
     is from, to the channel cid.
  *)

  (* Direct raw operations  - these do not effect translation between the internal and
     external representation of data
  *)

PROCEDURE RawRead (cid: ChanId; to: SYSTEM.ADDRESS; maxLocs: CARDINAL;
                   VAR locsRead: CARDINAL);
  (* Reads at most maxLocs items from cid, and assigns corresponding values to successive
     components of an ARRAY OF LOC variable for which the address of the first component
     is to. The number of items read is assigned to locsRead. The stored read result
     is set to the value allRight, or endOfInput.
  *)

PROCEDURE RawWrite (cid: ChanId; from: SYSTEM.ADDRESS; locsToWrite: CARDINAL);
  (* Writes a number of items given by the value of locsToWrite, from successive
     components of an ARRAY OF LOC variable for which the address of the first component
     is from, to the channel cid.
  *)

  (* Common operations *)

PROCEDURE GetName (cid: ChanId; VAR s: ARRAY OF CHAR);
  (* Copies to s a name associated with the channel cid, possibly truncated
     (depending on the capacity of s).
  *)

PROCEDURE Reset (cid: ChanId);
  (* Resets the channel cid to a state defined by the device module. *)

PROCEDURE Flush (cid: ChanId);
  (* Flushes any data buffered by the device module out to the channel cid. *)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  (* Access to read results *)

PROCEDURE SetReadResult (cid: ChanId; res: IOConsts.ReadResults);
  (* Sets the read result value for the channel cid to the value res. *)

PROCEDURE ReadResult (cid: ChanId): IOConsts.ReadResults;
  (* Returns the stored read result value for the channel cid. (This is initially the value
     notKnown).
  *)

  (* Users can discover which flags actually apply to a channel *)

PROCEDURE CurrentFlags (cid: ChanId): ChanConsts.FlagSet;
  (* Returns the set of flags that currently apply to the channel cid. *)

  (* The following exceptions are defined for this module and its clients *)

TYPE
  ChanExceptions =
    (wrongDevice,      (* device specific operation on wrong device *)
     notAvailable,     (* operation attempted that is not available on that channel *)
     skipAtEnd,        (* attempt to skip data from a stream that has ended *)
     softDeviceError,  (* device specific recoverable error *)
     hardDeviceError,  (* device specific non-recoverable error *)
     textParseError,   (* input data does not correspond to a character or line mark -
                          optional detection *)
     notAChannel       (* given value does not identify a channel - optional detection *)
    );

PROCEDURE IsChanException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional execution state
     because of the raising of an exception from ChanExceptions;
     otherwise returns FALSE.
  *)

PROCEDURE ChanException (): ChanExceptions;
  (* If the current coroutine is in the exceptional execution state because of the
     raising of an exception from ChanExceptions, returns the corresponding
     enumeration value, and otherwise raises an exception.
  *)

  (* When a device procedure detects a device error, it raises the exception softDeviceError
     or hardDeviceError.  If these exceptions are handled, the following facilities may be
     used to discover an implementation-defined error number for the channel.
  *)

TYPE
  DeviceErrNum = INTEGER;

PROCEDURE DeviceError (cid: ChanId): DeviceErrNum;
  (* If a device error exception has been raised for the channel cid, returns the error
     number stored by the device module.
  *)

END IOChan.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9.2.4.1.2 The Procedures of IOChan

The procedure `InvalidChan`

The value of the call `InvalidChan()` shall be the identity of the invalid channel.

NOTE 1 — The invalid channel is a channel on which no data transfer operations are available; enquiries on the invalid channel indicate that this is the case. The identity of the invalid channel can be used to initialize variables of the type `ChanId`.

InvalidChan : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Chan-id*

InvalidChan (*args*)*handler* ≙
  return *badchan* ()

538

Text Operations

Each of the following procedures invokes a corresponding operation for the device associated with the given channel. If the associated device supports the operation on the channel, the behaviour of the procedure conforms with the given description. The full behaviour is defined separately for each device.

These device operations produce a text stream. A text stream is a sequence of items, each of which corresponds either to a character or a line mark. The sequence may be empty.

The text operations provided by a device module perform any necessary translation between the internal representation (as a sequence of characters and line marks) and the external representation used by the source or destination. This may involve, for example, translation to and from escape sequences used in a coded character set, mapping between the external and internal representation of lines, or the interpretation of format effectors.

The interpretation of control characters is implementation-defined. The exception `textParseError` occurs (but need not be raised) if input data does not correspond to a character or line mark.

If the device does not support the operation on the channel, it raises the exception `notAvailable`.

The procedure `Look`

The call `Look(cid,ch,res)` shall invoke the *Look* operation for the device that is associated with the channel identified by `cid`.

NOTE 2 — If supported on the channel, the device *Look* operation attempts to examine the next item in the input stream for the channel identified by `cid`, without removing it. If the next item is a character, its value is assigned to `ch`; otherwise, the value of `ch` is not defined. `res` is set to the same value as the stored read result for the channel `cid`, this being:

`allRight`	if a character is seen next;
`endOfLine`	if no character is seen, the next item being a line mark;
`endOfInput`	if no character is seen, the input having ended.

Look : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

Look (*args*)*handler* ≙
  let [*cid*, *ch*, *res*] = *args* in
  def *tid* = *table-id*(*cid*) *handler* in
  def *pid* = *device-op*(*tid*, LOOK) in
  call-proper-procedure-value(*pid*) ([*tid*, *ch*, *res*]) *handler*

539 539 339

The procedure `Skip`

The call `Skip(cid)` shall invoke the *Skip* operation for the device that is associated with the channel identified by `cid`.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NOTE 3 — If supported on the channel, the device *Skip* operation attempts to remove the next item in the input stream for the channel identified by `cid`. If there is no next item, the end of the input stream having been reached, the exception `skipAtEnd` is raised; otherwise the next character or line mark in the stream is removed, and the stored read result for the channel `cid` is set to the value `allRight`.

Skip : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

Skip (*args*)*handler* $\triangleq$
  let [*cid*] = *args* in
  def *tid* = *table-id*(*cid*) *handler* in 539
  def *pid* = *device-op*(*tid*, SKIP) in 539
  call-proper-procedure-value(*pid*) ([*tid*]) *handler* 339

The procedure `SkipLook`

The call `SkipLook(cid,ch,res)` shall invoke the *SkipLook* operation for the device that is associated with the channel identified by `cid`.

NOTE 4 — If supported on the channel, the device *SkipLook* operation attempts to remove the next item in the input stream for the channel identified by `cid` and then to examine the following item without removing it. If there is no next item, the end of the input stream having been reached, the exception `skipAtEnd` is raised; otherwise the next character or line mark in the stream is removed. If this is followed by a character as the next item in the stream, its value is assigned to `ch`, without removing the character from the stream; otherwise, the value of `ch` is not defined. `res` is set to the same value as the stored read result for the channel `cid`, this being:

`allRight`	if a character is seen next;
`endOfLine`	if no character is seen, the next item being a line mark;
`endOfInput`	if no character is seen, the input having ended.

SkipLook : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

SkipLook (*args*)*handler* $\triangleq$
  let [*cid*, *ch*, *res*] = *args* in
  def *tid* = *table-id*(*cid*) *handler* in 539
  def *pid* = *device-op*(*tid*, SKIPLOOK) in 539
  call-proper-procedure-value(*pid*) ([*tid*, *ch*, *res*]) *handler* 339

The procedure `WriteLn`

The call `WriteLn(cid)` shall invoke the *WriteLn* operation for the device that is associated with the channel identified by `cid`.

NOTE 5 — If supported on the channel, the device *WriteLn* operation writes a line mark to the output stream identified by `cid`.

IOChan-WriteLn : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

IOChan-WriteLn (*args*)*handler* $\triangleq$
  let [*cid*] = *args* in
  def *tid* = *table-id*(*cid*) *handler* in 539
  def *pid* = *device-op*(*tid*, WRITELN) in 539
  call-proper-procedure-value(*pid*) ([*tid*]) *handler* 339

The procedure `TextRead`

The call `TextRead(cid,to,maxChars,charsRead)` shall invoke the *TextRead* operation for the device that is associated with the channel identified by `cid`.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NOTES

6 If supported on the channel, the device *TextRead* operation reads at most `maxChars` characters from the current line on the input stream for the channel identified by `cid`, and assigns their values to successive components of an array variable of the character type for which the address of the first component is `to`. The number of characters read is assigned to `charsRead`. The read result for the channel `cid` is set to the value

`allRight`	if '`maxChars = charsRead = 0`' or ('`maxChars > 0`' and '`charsRead > 0`');
`endOfLine`	if '`maxChars > 0`' and '`charsRead = 0`', the next item being a line mark;
`endOfInput`	if '`maxChars > 0`' and '`charsRead = 0`', the input having ended.

7 The intention is to allow 'sub-arrays' to be selected, by passing the address of a starting component within a larger array. An exception occurs, but need not be raised, if the call leads to an attempt to access a non-existent component of the larger array.

TextRead : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

TextRead (*args*)*handler* $\triangleq$
  let [*cid*, *to*, *maxChars*, *charsRead*] = *args* in
  def *tid* = *table-id*(*cid*) *handler* in 539
  def *pid* = *device-op*(*tid*, TEXTREAD) in 539
  call-proper-procedure-value(*pid*) ([*tid*, *to*, *maxChars*, *charsRead*]) *handler* 339

The procedure `TextWrite`

The call `TextWrite(cid,from,charsToWrite)` shall invoke the *TextWrite* operation for the device that is associated with the channel identified by `cid`.

NOTES

8 If supported on the channel, the device *TextWrite* operation copies `charsToWrite` characters, from successive components of an array variable of the character type, for which the address of the first component is `from`, to the output stream for the channel identified by `cid`. Copying starts from the index given by `offset`.

9 The intention is to allow 'sub-arrays' to be selected, by passing the address of a starting component within a larger array. An exception occurs, but need not be raised, if the call leads to an attempt to access a non-existent component of the larger array.

TextWrite : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

TextWrite (*args*)*handler* $\triangleq$
  let [*cid*, *from*, *charsToWrite*] = *args* in
  def *tid* = *table-id*(*cid*) *handler* in 539
  def *pid* = *device-op*(*tid*, TEXTWRITE) in 539
  call-proper-procedure-value(*pid*) ([*tid*, *from*, *charsToWrite*]) *handler* 339

Raw Operations

Each of the following procedures invokes a corresponding operation for the device associated with the given channel. If the associated device supports the operation on the channel, the behaviour of the procedure conforms with the given description. The full behaviour is defined for each device module.

The raw operations provided by a device module transfer data location by location with no translation or interpretation.

If the device does not support the operation on the channel, it raises the exception `notAvailable`.

The procedure `RawRead`

The call `RawRead(cid,to,maxLocs,locsRead)` shall invoke the *RawRead* operation for the device that is associated with the channel identified by `cid`.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NOTES

10 If supported on the channel, the device *RawRead* operation reads at most `maxLocs` items from the input stream for the channel identified by `cid`, and assigns their values to successive components of an array variable of the location type for which the address of the first component is `to`. The number of items read is assigned to `locsRead`. The read result for the channel `cid` is set to the value

`allRight` if ('`maxLocs = locsRead = 0`') or ('`maxLocs > 0` and '`locsRead > 0`')
`endOfInput` if '`maxLocs > 0`' and '`locsRead = 0`'

11 The intention is to allow 'sub-arrays' to be selected, by passing the address of a starting component within a larger array. An exception occurs, but need not be raised, if the call leads to an attempt to access a non-existent component of the larger array.

RawRead : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

RawRead (*args*)*handler* $\triangleq$
  let [*cid*, *to*, *maxLocs*, *locsRead*] = *args* in
  def *tid* = *table-id*(*cid*) *handler* in 539
  def *pid* = *device-op*(*tid*, RAWREAD) in 539
  call-proper-procedure-value(*pid*) ([*tid*, *to*, *maxLocs*, *locsRead*]) *handler* 339

The procedure `RawWrite`

The call `RawWrite(cid,from,locsToWrite)` shall invoke the *RawWrite* operation for the device that is associated with the channel identified by `cid`.

NOTES

12 If supported on the channel, the device *RawWrite* operation copies `locsToWrite` items, from successive components of an array variable of the character type, for which the address of the first component is `from`, to the output stream for the channel identified by `cid`. Copying starts from the index given by `offset`.

13 The intention is to allow 'sub-arrays' to be selected, by passing the address of a starting component within a larger array. An exception occurs, but need not be raised, if the call leads to an attempt to access a non-existent component of the larger array.

RawWrite : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

RawWrite (*args*)*handler* $\triangleq$
  let [*cid*, *from*, *locsToWrite*] = *args* in
  def *tid* = *table-id*(*cid*) *handler* in 539
  def *pid* = *device-op*(*tid*, RAWWRITE) in 539
  call-proper-procedure-value(*pid*) ([*tid*, *from*, *locsToWrite*]) *handler* 339

Common Operations

Each of the following procedures invokes a corresponding operation for the device associated with the given channel. The behaviour of the procedure conforms with the given description. The full behaviour is defined for each device module.

The procedure `GetName`

The call `GetName(cid,s)` shall invoke the *GetName* operation for the device that is associated with the channel identified by `cid`.

NOTES

14 The device *GetName* operation copies to `s` (as a string value) a name associated with the channel identified by `cid`.

15 The name will be truncated if the capacity of `s` is inadequate .

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Getname : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

Getname (*args*)*handler* $\triangleq$
  let [*cid*, *s*] = *args* in
  def *tid* = *table-id*(*cid*) *handler* in 539
  def *pid* = *device-op*(*tid*, GETNAME) in 539
  call-proper-procedure-value(*pid*) ([*tid*, *s*]) *handler* 339

The procedure `Reset`

The call `Reset(cid)` shall invoke the *Reset* operation for the device that is associated with the channel identified by `cid`.

NOTE 16 — The device *Reset* operation resets the device associated with the channel identified by `cid` to a state defined by the device module.

Reset : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

Reset (*args*)*handler* $\triangleq$
  let [*cid*] = *args* in
  def *tid* = *table-id*(*cid*) *handler* in 539
  def *pid* = *device-op*(*tid*, RESET) in 539
  call-proper-procedure-value(*pid*) ([*tid*]) *handler* 339

The procedure `Flush`

The call `Flush(cid)` shall invoke the *Flush* operation for the device that is associated with the channel identified by `cid`.

NOTE 17 — The device *Flush* operation flushes any data buffered by the device module out to the destination associated with `cid`.

Flush : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

Flush (*args*)*handler* $\triangleq$
  let [*cid*] = *args* in
  def *tid* = *table-id*(*cid*) *handler* in 539
  def *pid* = *device-op*(*tid*, FLUSH) in 539
  call-proper-procedure-value(*pid*) ([*tid*]) *handler* 339

Access to Read Results

Higher-level data input procedures, for units such as strings and numerals, may alter the read result for a channel to indicate success or a particular kind of failure of interpretation. The result can be recovered, if necessary, by the caller of the data input procedure.

The procedure `SetReadResult`

The call `SetReadResult(cid,res)` shall set the read result for the channel identified by `cid` to the value given by `res`.

SetReadResult : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

SetReadResult (*args*)*handler* $\triangleq$
  let [*cid*, *res*] = *args* in
  def *tid* = *table-id*(*cid*) *handler* in 539
  set-rdresult(*tid*, *res*) 539

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The procedure `ReadResult`

The value of the call `ReadResult(cid)` shall be the stored read result for the channel identified by `cid`.

IOChannel-ReadResult : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Enumeration-value*

IOChannel-ReadResult (*args*)*handler* $\triangleq$
  let [*cid*] = *args* in
  def *tid* = *table-id*(*cid*) *handler* in 539
  rdresult(*tid*) 539

Channel Enquiries

The procedure `CurrentFlags`

The value of the call `CurrentFlags(cid)` shall be the set of flags that currently apply to the channel identified by `cid`, as defined for the associated device.

CurrentFlags : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Set-value*

CurrentFlags (*args*)*handler* $\triangleq$
  let [*cid*] = *args* in
  def *tid* = *table-id*(*cid*) *handler* in 539
  def *props* = *get-props*(*tid*) in 539
  return *props*

Exceptions and Device Errors

The procedure `IsChanException`

If the calling coroutine is in the state of exceptional execution because of the raising of an exception from `ChanExceptions`, the value of the call `IsChanException()` shall be *true*; otherwise the value shall be *false*.

IsChanException : *Arguments* → *Exception-handler* $\xrightarrow{o}$ $\mathbb{B}$

IsChanException (*args*)*handler* $\triangleq$
  def *source* = *iolib-source* in
  def *res* = *m-IsCurrentSource*([*source*]) in 386
  return *res*

The procedure `ChanException`

If the calling coroutine is in the state of exceptional execution because of the raising of an exception from `ChanExceptions`, the value of the call `ChanException()` shall be the value that identifies the raised exception; otherwise the language exception `exException` shall be raised.

ChanException : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Value*

ChanException (*args*)*handler* $\triangleq$
  def *source* = *iolib-source* in
  def *n* = *m-CurrentNumber*([*source*]) *handler* in 386
  let *res* = *val-enumeration* (*n*, *chanexceptions*) in 330
  return *res*

The procedure `DeviceError`

The value of the call `DeviceError(cid)` shall be the error number stored by the device module for the channel identified

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

by `cid`, provided that a device error exception has been raised during an operation on that channel; otherwise the value of the call is not defined.

NOTE 18 — When a device procedure detects a device error, it raises the exception `softDeviceError` or `hardDeviceError`. If these exceptions are handled, the procedure `DeviceError` may be used to discover the implementation-defined error number stored by the device module for the channel that was in use when the device error occurred.

DeviceError : *Arguments* → *Exception-handler* $\xrightarrow{o}$ $\mathbb{Z}$

DeviceError (*args*)*handler*) $\triangleq$
  let [*cid*] = *args* in
  def *tid* = *table-id*(*cid*) *handler* in 539
  def *error* = *get-error*(*tid*) in 539
  return *error*

9.2.4.1.3 Device-independent Exceptions Raised by the Input/Output Library

The initialization of the module `IOChan`, or of a module imported by `IOChan`, shall allocate a unique value of the exception source type to identify device-independent exceptions raised by the input/output library.

iochan-initialization : () → *Exception-handler* $\xrightarrow{o}$ ()

iochan-initialization ()*handler* $\triangleq$
  def *source* = *allocate-source*() in 342
  if *source* ≠ nil
  then *iolib-source* := *source*
  else *mandatory-detection*(NO-EXCEPTION-SOURCE) *handler* 396

The device-independent exceptions raised by the input/output library shall be identified by the values of the enumeration type `ChanExceptions`.

The detection of the exceptions `textParseError` and `notAChannel` shall be implementation-defined.

The character string associated with each of the exceptions raised by the input-output library shall be implementation-defined.

values

chanexceptions = [`wrongDevice`, `notAvailable`, `skipAtEnd`, `softDeviceError`, `hardDeviceError`, `textParseError`, `notAChannel`];

wrongDevice = *mk-Enumeration-value* (`wrongDevice`, *chanexceptions*);

notAvailable = *mk-Enumeration-value* (`notAvailable`, *chanexceptions*);

skipAtEnd = *mk-Enumeration-value* (`skipAtEnd`, *chanexceptions*);

softDeviceError = *mk-Enumeration-value* (`softDeviceError`, *chanexceptions*);

hardDeviceError = *mk-Enumeration-value* (`hardDeviceError`, *chanexceptions*);

textParseError = *mk-Enumeration-value* (`textParseError`, *chanexceptions*);

notAChannel = *mk-Enumeration-value* (`notAChannel`, *chanexceptions*)

chan-exception : *Enumeration-value* → *Exception-handler* $\xrightarrow{o}$ ()

chan-exception (*ce*)*handler* $\triangleq$
  if *chan-exception-detected* (*ce*) 471
  then def *message* = *chan-exception-string*(*ce*) in 471
    raise-chan-exception(*ce*, *message*) *handler* 471
  else implementation-dependent

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

chan-exception-using : *Enumeration-value* × *Message* → *Exception-handler* $\xrightarrow{o}$ ()

chan-exception-using (*ce*, *message*)*handler* $\triangleq$
  if *chan-exception-detected* (*ce*)
  then *raise-chan-exception*(*ce*, *message*) *handler* 471
  else implementation-dependent 471

chan-exception-detected : *Enumeration-value* → $\mathbb{B}$

chan-exception-detected (*ce*) $\triangleq$
  cases *ce* :
    wrongDevice → true,
    notAvailable → true,
    skipAtEnd → true,
    softDeviceError → true,
    hardDeviceError → true,
    textParseError → implementation-defined,
    notAChannel → implementation-defined
  end

raise-chan-exception : *Enumeration-value* × *Message* → *Exception-handler* $\xrightarrow{o}$ ()

raise-chan-exception (*ce*, *message*)*handler* $\triangleq$
  def *source* = *iolib-source* in
  let *number* ∈ inds *chanexceptions* be st
      ce = *val-enumeration* (*number*, *chanexceptions*) in 330
  m-RAISE([*source*, *number*, *message*]) *handler* 383

chan-exception-string : *Enumeration-value* $\xrightarrow{o}$ *Message*

chan-exception-string (*ce*) $\triangleq$
  return implementation-defined

annotations The result is an implementation-defined string value.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9.2.5 Obtaining Channels from Device Modules

Separate device modules are defined that provide a program with the facility to obtain a new channel, connected either to a sequential stream, a rewindable sequential file, a random access file, or a terminal device.

A request to obtain a channel is made by calling an appropriate 'open procedure', in general supplying a name that identifies the source or destination to which the connection is to be made.

The required input/output operations are specified using combinations of flags that are defined in terms of constants imported from the module `ChanConsts`.

An open procedure returns a parameter of an enumeration type (exported from the module `ChanConsts`) that indicates the success, or otherwise, of the request.

Each of these device modules defines a predicate allowing a check to be made that a given channel was opened by that module, as well as a 'close procedure' that allows a program to break the connection and release the channel.

Procedures are also provided for device-dependent operations, such as setting the read/write position on a random access file.

A further device module is defined to allow access to the program arguments over a pre-opened channel.

9.2.5.1 The Module ChanConsts

The module `ChanConsts` defines common types and values for use with open procedures. Programs do not normally need to import from `ChanConsts` directly, since device modules define identifiers that correspond to those defined by this module.

9.2.5.1.1 The Definition Module of ChanConsts

```
DEFINITION MODULE ChanConsts;

  (* Common types and values for channel open requests and results *)

TYPE
  ChanFlags =         (* Request flags possibly given when a channel is opened *)
  ( readFlag,         (* input operations are requested/available *)
    writeFlag,        (* output operations are requested/available *)
    oldFlag,          (* a file may/must/did exist before the channel is opened *)
    textFlag,         (* text operations are requested/available *)
    rawFlag,          (* raw operations are requested/available *)
    interactiveFlag, (* interactive use is requested/applies *)
    echoFlag          (* echoing by interactive device on removal of characters from input
                         stream requested/applies *)
  );

  FlagSet = SET OF ChanFlags;

  (* Singleton values of FlagSet, to allow for example, read + write *)

CONST
  read = FlagSet{readFlag};    (* input operations are requested/available *)
  write = FlagSet{writeFlag}; (* output operations are requested/available *)
  old = FlagSet{oldFlag};      (* a file may/must/did exist before the channel is opened *)
  text = FlagSet{textFlag};    (* text operations are requested/available *)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  raw = FlagSet{rawFlag};      (* raw operations are requested/available *)
  interactive = FlagSet{interactiveFlag}; (* interactive use is requested/applies *)
  echo = FlagSet{echoFlag};    (* echoing by interactive device on removal of characters from
                                  input stream requested/applies *)

TYPE
  OpenResults =          (* Possible results of open requests *)
    (opened,             (* the open succeeded as requested *)
     wrongNameFormat,    (* given name is in the wrong format for the implementation *)
     wrongFlags,         (* given flags include a value that does not apply to the device *)
     tooManyOpen,        (* this device cannot support any more open channels *)
     outOfChans,         (* no more channels can be allocated *)
     wrongPermissions,   (* file or directory permissions do not allow request *)
     noRoomOnDevice,     (* storage limits on the device prevent the open *)
     noSuchFile,         (* a needed file does not exist *)
     fileExists,         (* a file of the given name already exists when a new one is required *)
     wrongFileType,      (* the file is of the wrong type to support the required operations *)
     noTextOperations,   (* text operations have been requested, but are not supported *)
     noRawOperations,    (* raw operations have been requested, but are not supported *)
     noMixedOperations,  (* text and raw operations have been requested, but they
                            are not supported in combination *)
     alreadyOpen,        (* the source/destination is already open for operations not supported
                            in combination with the requested operations *)
     otherProblem        (* open failed for some other reason *)
    );

END ChanConsts.
```

9.2.5.1.2 The Use of ChanConsts

To save repetition in the natural language definition of the device modules, the meaning given to some values of `FlagSet` and `OpenResults` is defined here. The meaning of the other flags is given for the open operations to which they apply.

In a call of a device module open procedure that has a request parameter of the type `FlagSet` and a result parameter of the type `OpenResults`:

If the result is `opened`, the following operations shall be provided for the opened channel for the combinations of request flags shown:

	read	write	
text	text input	text output	—as defined for the device
raw	raw input	raw output	—as defined for the device

NOTE — The supplied flags specify the *minimal* functionality that must be available for the open operation to succeed. Implementations are free to allow operations in addition to those specified in the request flags provided that these are reflected in the enquiry flags returned for the channel.

If the result is other than `opened`, the channel parameter shall be assigned the value identifying the *invalid* channel, on which no input/output operations are provided. The result shall be chosen according to the following table:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

`wrongNameFormat`	if the given name is not in the format defined for the implementation
`wrongFlags`	if the given flags include a value that does not apply to the device
`tooManyOpen`	if the device cannot support any more open channels
`outOfChans`	if no more channels can be allocated
`wrongPermissions`	if file or directory permissions do not allow the request to be met
`noRoomOnDevice`	if storage limits on the device do not allow the request to be met
`noSuchFile`	if a needed file does not exist
`fileExists`	if a file of the given name already exists when a new one is required
`wrongFileType`	if the named file is of the wrong type to support the required operations
`noTextOperations`	if text operations have been requested, but are not supported by the device
`noRawOperations`	if raw operations have been requested, but are not supported by the device
`noMixedOperations`	if text and raw operations have been requested, but they are not supported in combination by the device
`alreadyOpen`	if the source/destination is already open for operations that are not supported in combination with the operations now requested
`otherProblem`	if the open failed for a reason other than the above

9.2.5.1.3 Auxiliary Formal Model Definitions

The following definitions give the enumeration values from `ChanConsts` used in the formal model.

values

chanflags = [`readFlag`, `writeFlag`, `oldFlag`, `textFlag`, `rawFlag`, `interactiveFlag`, `echoFlag`];

readFlag = *mk-Enumeration-value* (`readFlag`, *chanflags*);

writeFlag = *mk-Enumeration-value* (`writeFlag`, *chanflags*);

oldFlag = *mk-Enumeration-value* (`oldFlag`, *chanflags*);

textFlag = *mk-Enumeration-value* (`textFlag`, *chanflags*);

rawFlag = *mk-Enumeration-value* (`rawFlag`, *chanflags*);

interactiveFlag = *mk-Enumeration-value* (`interactiveFlag`, *chanflags*);

echoFlag = *mk-Enumeration-value* (`echoFlag`, *chanflags*);

openresults = [`opened`, `wrongNameFormat`, `wrongFlags`, `tooManyOpen`, `outOfChans`, `wrongPermissions`, `noRoomOnDevice`, `noSuchFile`, `fileExists`, `wrongFileType`, `noTextOperations`, `noRawOperations`, `noMixedOperations`, `alreadOpen`, `otherProblem`];

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

opened = *mk-Enumeration-value* (opened, *openresults*);

wrongNameFormat = *mk-Enumeration-value* (wrongNameFormat, *openresults*);

wrongFlags = *mk-Enumeration-value* (wrongFlags, *openresults*);

tooManyOpen = *mk-Enumeration-value* (tooManyOpen, *openresults*);

outOfChans = *mk-Enumeration-value* (outOfChans, *openresults*);

wrongPermissions = *mk-Enumeration-value* (wrongPermissions, *openresults*);

noRoomOnDevice = *mk-Enumeration-value* (noRoomOnDevice, *openresults*);

noSuchFile = *mk-Enumeration-value* (noSuchFile, *openresults*);

fileExists = *mk-Enumeration-value* (fileExists, *openresults*);

wrongFileType = *mk-Enumeration-value* (wrongFileType, *openresults*);

noTextOperations = *mk-Enumeration-value* (noTextOperations, *openresults*);

noRawOperations = *mk-Enumeration-value* (noRawOperations, *openresults*);

noMixedOperations = *mk-Enumeration-value* (noMixedOperations, *openresults*);

alreadyOpen = *mk-Enumeration-value* (alreadyOpen, *openresults*);

otherProblem = *mk-Enumeration-value* (otherProblem, *openresults*)

9.2.5.2 The Module StreamFile

The module StreamFile provides facilities that allow for obtaining and releasing channels that are connected to named sources and/or destinations for independent sequential data streams.

9.2.5.2.1 The Definition Module of StreamFile

```
DEFINITION MODULE StreamFile;

  (* Independent sequential data streams *)

IMPORT IOChan, ChanConsts;

TYPE
  ChanId = IOChan.ChanId;
  FlagSet = ChanConsts.FlagSet;
  OpenResults = ChanConsts.OpenResults;

  (* Accepted singleton values of FlagSet *)

CONST
  read = FlagSet{ChanConsts.readFlag};   (* input operations are requested/available *)
  write = FlagSet{ChanConsts.writeFlag}; (* output operations are requested/available *)
  old = FlagSet{ChanConsts.oldFlag};     (* a file may/must/did exist before the channel is
                                            opened *)
  text = FlagSet{ChanConsts.textFlag};   (* text operations are requested/available *)
  raw = FlagSet{ChanConsts.rawFlag};     (* raw operations are requested/available *)


PROCEDURE Open (VAR cid: ChanId; name: ARRAY OF CHAR; flags: FlagSet; VAR res: OpenResults);
  (* Attempts to obtain and open a channel connected to a sequential stream of the given name.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
    Without the write flag, read is implied; the read flag implies old;
    without the raw flag, text is implied.
    If successful, assigns to cid the identity of the opened channel, and assigns the value
    opened to res.
    If a channel cannot be opened as required, the value of res indicates the reason, and
    cid identifies the invalid channel.
  *)

PROCEDURE IsStreamFile (cid: ChanId): BOOLEAN;
  (* Tests if the channel identified by cid is open to a sequential stream. *)

PROCEDURE Close (VAR cid: ChanId);
  (* If the channel identified by cid is not open to a sequential stream, the exception
     wrongDevice is raised; otherwise closes the channel, and assigns the value identifying
     the invalid channel to cid.
  *)

END StreamFile.
```

9.2.5.2.2 The Procedures of StreamFile

In a request to open a sequential stream, the flags `read`, `write`, `old`, `text`, and `raw` shall apply. If `raw` is not included in the request parameter `flags`, inclusion of `text` shall be implied.

values

$streamfile\text{-}flags = \{readFlag, writeFlag, oldFlag, textFlag, rawFlag\}$

$streamfile\text{-}merge\text{-}flags : Set\text{-}value \times Set\text{-}value \rightarrow [Set\text{-}value]$

$streamfile\text{-}merge\text{-}flags\ (given, required) \triangleq$
  let $default =$ if $rawFlag \in given$ then $\{\ \}$ else $\{textFlag\}$ in
  if $given \subseteq streamfile\text{-}flags$
  then $given \cup required \cup default$
  else nil

$streamfile\text{-}check\text{-}file : Char^* \times Set\text{-}value \rightarrow Exception\text{-}handler \xrightarrow{o} Enumeration\text{-}value$

$streamfile\text{-}check\text{-}file\ (sname, flags) handler \triangleq$
  implementation-defined

annotations Returns *opened* if the requested read/write/text/raw operations are available on a stream of the given name; otherwise returns *tooManyOpen*, *wrongPermissions*, *noRoomOnDevice*, *wrongFileType*, *noTextOperations*, *noRawOperations*, *noMixedOperations*, *alreadyOpen*, or *otherProblem*.

The procedure `Open`

If successful, the call `Open(cid,name,flags,res)` shall assign to `cid` the identity of a channel that is connected to a sequential stream specified by `name`, and the value `opened` shall be assigned to `res`.

If `write` is not included in `flags`, inclusion of `read` shall be implied; if `read` is given or implied, inclusion of `old` shall be implied; a source of the given name shall already exist if the call is to succeed.

If `write` is included, a destination of the given name shall not already exist, unless the flag `old` is given or implied.

If a channel cannot be opened as required, the value of `res` shall indicate the reason, and `cid` shall identify the invalid channel.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NOTE — Distinct modes in combination with `text` and/or `raw` are given by the following equivalent sets of flags:

read from an existing source:	`read`	`old`	`read+old`
write to a new destination:	`write`		
write to a new or old destination:	`write+old`		
read/write an existing source/destination:	`read+write`	`read+write+old`	

streamfile-Open : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

```
streamfile-Open (args)handler ≙
  let [vcid, name, flags, vres] = args in
  let sname = pure-string (name) in                                          579
  let required1 = if writeFlag ∉ flags then {readFlag, oldFlag} else { } in
  let required2 = if readFlag ∈ flags then {oldFlag} else { } in
  let xflags = streamfile-merge-flags (flags, required1 ∪ required2) in      476
  if xflags = nil
  then (assign(vcid, badchan ()) handler;                                    150 538
        assign(vres, wrongFlags) handler                                     150
        )
  else def res = streamfile-check(sname, xflags) handler in                  477
       if res ≠ opened
       then (assign(vcid, badchan ()) handler;                               150 538
             assign(vres, res) handler                                       150
             )
       else def mk-(cid, tid) = make-a-channel(STREAMFILE, xflags, streamfile-ops) in   540
            if cid = badchan ()                                              538
            then (assign(vcid, badchan ()) handler;                          150 538
                  assign(vres, outOfChannels) handler                        150
                  )
            else (def ifile = if readFlag ∈ xflags then read-file(name) handler else nil in   537
                  let ofile = if writeFlag ∈ xflags then [] else nil in
                  streamfile-make-a-buffer(tid, ifile, ofile, sname) ;       483
                  assign(vcid, cid) handler;                                 150
                  assign(vres, opened) handler                               150
                  )
```

streamfile-check : *Char** × *Set-value* → *Exception-handler* $\xrightarrow{o}$ *Enumeration-value*

```
streamfile-check (sname, flags)handler ≙
  def res = if is-legal-filename (sname)                                     529
            then if readFlag ∈ flags ∧ ¬ is-existing-file (sname)            529
                 then noSuchFile
                 else if writeFlag ∈ flags ∧ is-existing-file (sname) ∧ oldFlag ∉ flags   529
                      then fileExists
                      else streamfile-check-file(sname, flags) handler       476
            else wrongNameFormat in
  return res
```

The procedure `IsStreamFile`

The value of the call `IsStreamFile(cid)` shall be *true* if the channel identified by `cid` is open to a sequential stream, and *false* otherwise.

IsStreamFile : *Arguments* → *Exception-handler* $\xrightarrow{o}$ $\mathbb{B}$

```
IsStreamFile (args)handler ≙
  let [cid] = args in
  def tid = table-id(cid) in                                                 539
  def did = get-device-id(tid) in                                            540
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  return did = STREAMFILE
```

The procedure `Close`

If the channel identified by `cid` is open to a sequential stream, the call `Close(cid)` shall close the channel and assign the value identifying the invalid channel to `cid`; otherwise, the exception `wrongDevice` shall be raised.

```
streamfile-Close : Arguments → Exception-handler →° ()
streamfile-Close (args)handler △
  let [cid] = args in
  def tid = table-id(cid) in                                          539
  (vet(tid, STREAMFILE) handler;                                      529
   streamfile-free([tid]) handler;                                    482
   release-a-channel(cid, tid) ;                                      540
   assign(cid, badchan ()) handler                                    150 538
  )
```

9.2.5.2.3 The Device Procedures of StreamFile

If supported on a channel open to a `StreamFile` device, the `StreamFile` device procedures shall have the semantics defined below; otherwise, they shall have the semantics of the corresponding device procedure for the invalid channel.

NOTE — Device procedures, other than *Free*, are invoked through corresponding procedures from `IOChan`.

The `StreamFile` device procedure *Look*

```
streamfile-look : Arguments → Exception-handler →° ()
streamfile-look (args)handler △
  let [tid, vch, vres] = args in
  (check-flags(tid, {readFlag}) handler;                              529
   def file = streamfile-get-ifile(tid) in                            483
   def mk-(ch, res) = if file = []
                      then mk-(nil , endOfInput)
                      else def choice = a-text-item(tid, file) handler in   529
                           if choice = END-LINE
                           then mk-(nil , endOfLine)
                           else mk-(choice, allRight) in
   (set-rdresult(tid, res) ;                                          539
    assign(vres, res) handler;                                        150
    if ch = nil
    then let impdep : (Char | UNDEFINED) in
         assign(vch, impdep) handler                                  150
    else assign(vch, ch) handler                                      150
    )
  )
```

The `StreamFile` device procedure *Skip*

```
streamfile-skip : Arguments → Exception-handler →° ()
streamfile-skip (args)handler △
  let [tid] = args in
  (check-flags(tid, {readFlag}) handler;                              529
   def file = streamfile-get-ifile(tid) in                            483
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
    if file = []
    then chan-exception(skipAtEnd) handler                                    470
    else def choice = a-text-item(tid, file) handler in                       529
         (let skipped = len emit (tid, choice) in
          let nfile = file (skipped + 1, ..., len file) in
          streamfile-set-ifile(tid, nfile) ;                                  483
          set-rdresult(tid, allRight)                                         539
          )
    )
```

The StreamFile device procedure *SkipLook*

```
streamfile-skiplook : Arguments → Exception-handler -o-> ()
streamfile-skiplook (args)handler △
  let [tid, vch, vres] = args in
  (streamfile-skip([tid]) handler;                                            478
   streamfile-look(args) handler                                              478
   )
```

The StreamFile device procedure *WriteLn*

```
streamfile-writeln : Arguments → Exception-handler -o-> ()
streamfile-writeln (args)handler △
  let [tid] = args in
  (check-flags(tid, {writeFlag}) handler;                                     529
   def file = streamfile-get-ofile(tid) in                                    483
   let bos = emit (tid, END-LINE) in                                          529
   let newf = file ↷ bos in
   streamfile-set-ofile(tid, newf)                                            483
   )
```

The StreamFile device procedure *TextRead*

```
streamfile-textread : Arguments → Exception-handler -o-> ()
streamfile-textread (args)handler △
  let [tid, ato, maxChars, vcharsRead] = args in
  (check-flags(tid, {readFlag}) handler;                                      529
   if maxChars = 0
   then (set-rdresult(tid, allRight) ;                                        539
         assign(vcharsRead, 0) handler                                        150
         )
   else def file = streamfile-get-ifile(tid) in                               483
        if file = []
        then (set-rdresult(tid, endOfInput) ;                                 539
              assign(vcharsRead, 0) handler                                   150
              )
        else def choice = a-text-item(tid, file) handler in                   529
             if choice = END-LINE
             then (set-rdresult(tid, endOfLine) ;                             539
                   assign(vcharsRead, 0) handler                              150
                   )
             else def mk-(ans, nchars) = chars-up-to(tid, file, maxChars) handler in   530
                  def mk-(vto, cap) = make-subarray(ato, maxChars) in         529
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
                let safechars = if nchars < cap
                                then nchars else cap in
                (let chars = ans(1, ..., safechars) in
                 (for all i ∈ inds chars
                  do assign(vto(i − 1), chars(i)) handler ;                    150
                  let amount = len emit(tid, chars) in
                  let nfile = file(amount + 1, ..., len file) in
                  streamfile-set-ifile(tid, nfile) ;                           483
                  set-rdresult(tid, allRight) ;                                539
                  assign(vcharsRead, safechars) handler                        150
                  );
                 if safechars < nchars
                 then non-mandatory-detection(NONEXISTENT) handler             397
                 else skip
                 )
    )
```

The StreamFile device procedure *TextWrite*

```
streamfile-textwrite : Arguments → Exception-handler -o→ ()
streamfile-textwrite(args)handler ≙
  let [tid, ato, maxChars] = args in
  def mk-(vfrom, cap) = make-subarray(ato, maxChars) in                        529
  (check-flags(tid, {writeFlag}) handler;                                      529
   def file = streamfile-get-ofile(tid) in                                     483
   let safechars = if maxChars ≤ cap then maxChars else cap in
   (if safechars > 0
    then let frmseq = map-to-seq(vfrom) in                                     615
         let outs = frmseq(1, ..., safechars) in
         let bos = emit(tid, outs) in                                          529
         let newf = file ↷ bos in
         streamfile-set-ofile(tid, newf)                                       483
    else skip;
    if safechars < maxChars
    then non-mandatory-detection(NONEXISTENT) handler                          397
    else skip
    )
   )
```

The StreamFile device procedure *RawRead*

```
streamfile-rawread : Arguments → Exception-handler -o→ ()
streamfile-rawread(args)handler ≙
  let [tid, ato, maxLocs, vlocsRead] = args in
  (check-flags(tid, {readFlag}) handler;                                       529
   if maxLocs = 0
   then (set-rdresult(tid, allRight) ;                                         539
         assign(vlocsRead, 0) handler                                          150
         )
   else def file = streamfile-get-ifile(tid) in                                483
        if file = []
        then (set-rdresult(tid, endOfInput) ;                                  539
              assign(vlocsRead, 0) handler                                     150
              )
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
    else let ans = up-to (file, maxLocs) in                                    530
         def mk-(vto, cap) = make-subarray(ato, maxLocs) in                    529
         let safelocs = if len ans ≤ cap then len ans else cap in
         (let locs = ans (1, ..., safelocs) in
          (for all i ∈ inds locs
           do assign(vto (i − 1), locs (i)) handler ;                          150
           let amount = len locs in
           let nfile = file (amount + 1, ..., len file) in
           streamfile-set-ifile(tid, nfile) ;                                  483
           set-rdresult(tid, allRight) ;                                       539
           assign(vlocsRead, safelocs) handler                                 150
           );
          if safelocs < len ans
          then non-mandatory-detection(NONEXISTENT) handler                    397
          else skip
          )
  )
```

The StreamFile device procedure *RawWrite*

```
streamfile-rawwrite : Arguments → Exception-handler →o ()
streamfile-rawwrite (args)handler ≙
  let [tid, ato, maxLocs] = args in
  def mk-(vfrom, cap) = make-subarray(ato, maxLocs) in                         529
  (check-flags(tid, {writeFlag}) handler;                                      529
   def file = streamfile-get-ofile(tid) in                                     483
   let safelocs = if maxLocs ≤ cap then maxLocs else cap in
   (if safelocs > 0
    then let frmseq = map-to-seq (vfrom) in                                    615
         let outs = frmseq (1, ..., safelocs) in
         let newf = file ↷ outs in
         streamfile-set-ofile(tid, newf)                                       483
    else skip;
    if safelocs < maxLocs
    then non-mandatory-detection(NONEXISTENT) handler                          397
    else skip
    )
  )
```

The StreamFile device procedure *GetName*

```
streamfile-getname : Arguments → Exception-handler →o ()
streamfile-getname (args)handler ≙
  let [tid, vs] = args in
  def filename = streamfile-get-name(tid) in                                   483
  assign(vs, filename) handler                                                 150
```

The StreamFile device procedure *Reset*

```
streamfile-reset : Arguments → Exception-handler →o ()
streamfile-reset (args)handler ≙
  let [tid] = args in
  streamfile-flush([tid]) handler                                              482
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The `StreamFile` device procedure *Flush*

streamfile-flush : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

streamfile-flush (*args*)*handler* $\triangleq$
  let [*tid*] = *args* in
  def *props* = *get-props*(*tid*) in 539
  if *writeFlag* ∈ *props*
  then def *file* = *streamfile-get-ofile*(*tid*) in 483
       def *filename* = *streamfile-get-name*(*tid*) in 483
       write-file(*filename*, *file*) *handler* 537

The `StreamFile` device procedure *Free*

streamfile-free : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

streamfile-free (*args*)*handler* $\triangleq$
  let [*tid*] = *args* in
  (*streamfile-flush*([*tid*]) *handler*; 482
   streamfile-release-a-buffer(*tid*) 483
)

values

streamfile-ops =
{LOOK ↦ *streamfile-look-pid*,
SKIP ↦ *streamfile-skip-pid*,
SKIPLOOK ↦ *streamfile-skiplook-pid*,
WRITELN ↦ *streamfile-writeln-pid*,
READTEXT ↦ *streamfile-readtext-pid*,
WRITETEXT ↦ *streamfile-writetext-pid*,
READBIN ↦ *streamfile-readraw-pid*,
WRITEBIN ↦ *streamfile-writeraw-pid*,
FLUSH ↦ *streamfile-flush-pid*,
RESET ↦ *streamfile-reset-pid*,
FREE ↦ *streamfile-free-pid*}

;

streamfile-dens =
{*streamfile-look-pid* ↦ *make-proper-procden* (*streamfile-look*), 339
streamfile-skip-pid ↦ *make-proper-procden* (*streamfile-skip*), 339
streamfile-skiplook-pid ↦ *make-proper-procden* (*streamfile-skiplook*), 339
streamfile-writeln-pid ↦ *make-proper-procden* (*streamfile-writeln*), 339
streamfile-readtext-pid ↦ *make-proper-procden* (*streamfile-readtext*), 339
streamfile-writetext-pid ↦ *make-proper-procden* (*streamfile-writetext*), 339
streamfile-readraw-pid ↦ *make-proper-procden* (*streamfile-readraw*), 339
streamfile-writeraw-pid ↦ *make-proper-procden* (*streamfile-writeraw*), 339
streamfile-flush-pid ↦ *make-proper-procden* (*streamfile-flush*), 339
streamfile-reset-pid ↦ *make-proper-procden* (*streamfile-reset*), 339
streamfile-free-pid ↦ *make-proper-procden* (*streamfile-free*)} 339

annotations The mapping *streamfile-dens*, from elements of type *Procedure-id* to the corresponding procedure denotations, is part of the mapping *dens* used in the construction of the abstract implementation module — see Annex B.

9.2.5.2.4 The State for StreamFile

types

FileStream = [*Elementary-loc-value**]

state *StreamFile* of
  streamfile-ifiles : *Table-id* $\xrightarrow{m}$ *FileStream*
  streamfile-ofiles : *Table-id* $\xrightarrow{m}$ *FileStream*
  streamfile-names : *Table-id* $\xrightarrow{m}$ *Char**

  init *mk-StreamFile* (*ifiles*, *ofiles*, *names*) $\triangleq$
    ifiles = {↦} ∧
    ofiles = {↦} ∧
    names = {↦}
end

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Auxiliaries

Access to the state for StreamFile

$streamfile\text{-}make\text{-}a\text{-}buffer : Table\text{-}id \times FileStream \times FileStream \times Char^* \xrightarrow{o} ()$

$streamfile\text{-}make\text{-}a\text{-}buffer\,(tid, ifile, ofile, name) \triangleq$
$\quad \| \; (streamfile\text{-}ifiles := streamfile\text{-}ifiles \cup \{tid \mapsto ifile\},$
$\quad\quad streamfile\text{-}ofiles := streamfile\text{-}ofiles \cup \{tid \mapsto ofile\},$
$\quad\quad streamfile\text{-}names := streamfile\text{-}names \cup \{tid \mapsto name\})$

$streamfile\text{-}release\text{-}a\text{-}buffer : Table\text{-}id \xrightarrow{o} ()$

$streamfile\text{-}release\text{-}a\text{-}buffer\,(tid) \triangleq$
$\quad \| \; (streamfile\text{-}ifiles := \{tid\} \lhd streamfile\text{-}ifiles,$
$\quad\quad streamfile\text{-}ofiles := \{tid\} \lhd streamfile\text{-}ofiles,$
$\quad\quad streamfile\text{-}names := \{tid\} \lhd streamfile\text{-}names)$

$streamfile\text{-}set\text{-}ifile : Table\text{-}id \times FileStream \xrightarrow{o} ()$

$streamfile\text{-}set\text{-}ifile\,(tid, file) \triangleq$
$\quad streamfile\text{-}ifiles\,(tid) := file$

$streamfile\text{-}get\text{-}ifile : Table\text{-}id \xrightarrow{o} FileStream$

$streamfile\text{-}get\text{-}ifile\,(tid) \triangleq$
$\quad \mathsf{def}\ file = streamfile\text{-}ifiles\,(tid)\ \mathsf{in}$
$\quad \mathsf{return}\ file$

$streamfile\text{-}set\text{-}ofile : Table\text{-}id \times FileStream \xrightarrow{o} ()$

$streamfile\text{-}set\text{-}ofile\,(tid, file) \triangleq$
$\quad streamfile\text{-}ofiles\,(tid) := file$

$streamfile\text{-}get\text{-}ofile : Table\text{-}id \xrightarrow{o} FileStream$

$streamfile\text{-}get\text{-}ofile\,(tid) \triangleq$
$\quad \mathsf{def}\ file = streamfile\text{-}ofiles\,(tid)\ \mathsf{in}$
$\quad \mathsf{return}\ file$

$streamfile\text{-}set\text{-}name : Table\text{-}id \times Char^* \xrightarrow{o} ()$

$streamfile\text{-}set\text{-}name\,(tid, name) \triangleq$
$\quad streamfile\text{-}names\,(tid) := name$

$streamfile\text{-}get\text{-}name : Table\text{-}id \xrightarrow{o} Char^*$

$streamfile\text{-}get\text{-}name\,(tid) \triangleq$
$\quad \mathsf{def}\ name = streamfile\text{-}names\,(tid)\ \mathsf{in}$
$\quad \mathsf{return}\ name$

9.2.5.3 The Module SeqFile

The module `SeqFile` provides facilities that allow for obtaining and releasing channels that are connected to named rewindable sequential stored files.

If opened for both writing and reading, data written to the file may be read back from the start of the file. Rewriting from the start of the file causes the previous contents to be lost.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9.2.5.3.1 The Definition Module of SeqFile

```
DEFINITION MODULE SeqFile;

  (* Rewindable sequential files *)

IMPORT IOChan, ChanConsts;

TYPE
  ChanId = IOChan.ChanId;
  FlagSet = ChanConsts.FlagSet;
  OpenResults = ChanConsts.OpenResults;

  (* Accepted singleton values of FlagSet *)

CONST
  read = FlagSet{ChanConsts.readFlag};   (* input operations are requested/available *)
  write = FlagSet{ChanConsts.writeFlag}; (* output operations are requested/available *)
  old = FlagSet{ChanConsts.oldFlag};     (* a file may/must/did exist before the channel is
                                            opened *)
  text = FlagSet{ChanConsts.textFlag};   (* text operations are requested/available *)
  raw = FlagSet{ChanConsts.rawFlag};     (* raw operations are requested/available *)

PROCEDURE OpenWrite (VAR cid: ChanId; name: ARRAY OF CHAR; flags: FlagSet;
                     VAR res: OpenResults);
  (* Attempts to obtain and open a channel connected to a stored rewindable file of the
     given name.
     The write flag is implied; without the raw flag, text is implied.
     If successful, assigns to cid the identity of the opened channel, assigns the value
     opened to res, and selects output mode, with the write position at the start of the
     file (i.e. the file is of zero length).
     If a channel cannot be opened as required, the value of res indicates the reason, and
     cid identifies the invalid channel.
  *)

PROCEDURE OpenAppend (VAR cid: ChanId; name: ARRAY OF CHAR; flags: FlagSet;
                      VAR res: OpenResults);
  (* Attempts to obtain and open a channel connected to a stored rewindable file of the
     given name.
     The write and old flags are implied; without the raw flag, text is implied.
     If successful, assigns to cid the identity of the opened channel, assigns the value
     opened to res, and selects output mode, with the write position corresponding to the
     length of the file.
     If a channel cannot be opened as required, the value of res indicates the reason, and
     cid identifies the invalid channel.
  *)

PROCEDURE OpenRead (VAR cid: ChanId; name: ARRAY OF CHAR; flags: FlagSet;
                    VAR res: OpenResults);
  (* Attempts to obtain and open a channel connected to a stored rewindable file of the
     given name.
     The read and old flags are implied; without the raw flag, text is implied.
     If successful, assigns to cid the identity of the opened channel, assigns the value
     opened to res, and selects input mode, with the read position corresponding to the
     start of the file.
     If a channel cannot be opened as required, the value of res indicates the reason, and
     cid identifies the invalid channel.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  *)

PROCEDURE IsSeqFile (cid: ChanId): BOOLEAN;
  (* Tests if the channel identified by cid is open to a rewindable sequential file. *)

PROCEDURE Reread (cid: ChanId);
  (* If the channel identified by cid is not open to a rewindable sequential file, the
     exception wrongDevice is raised; otherwise attempts to set the read position to the
     start of the file, and to select input mode.
     If the operation cannot be performed (perhaps because of insufficient permissions)
     neither input mode nor output mode is selected.
  *)

PROCEDURE Rewrite (cid: ChanId);
  (* If the channel identified by cid is not open to a rewindable sequential file, the
     exception wrongDevice is raised; otherwise, attempts to truncate the file to zero
     length, and to select output mode.
     If the operation cannot be performed (perhaps because of insufficient permissions)
     neither input mode nor output mode is selected.
  *)

PROCEDURE Close (VAR cid: ChanId);
  (* If the channel identified by cid is not open to a rewindable sequential file, the
     exception wrongDevice is raised; otherwise closes the channel, and assigns the value
     identifying the invalid channel to cid.
  *)

END SeqFile.
```

9.2.5.3.2 The Procedures of SeqFile

In a request to open a rewindable sequential file, the flags `read`, `write`, `old`, `text`, and `raw` shall apply. If `raw` is not included in the request parameter `flags`, inclusion of `text` shall be implied.

Channels open to rewindable sequential files may be in input mode or in output mode.
In input mode, only input operations shall be available — '`(IOChan.Flags()*(read+write) = read)`' shall be true, and an attempt to write over the channel shall raise the exception `notAvailable`.
In output mode, only output operations shall be available — '`(IOChan.Flags()*(read+write) = write)`' shall be true, and an attempt to read from the channel shall raise the exception `notAvailable`. All data written to a rewindable sequential file shall be appended to previous data written to that file.

values

$seqfile\text{-}flags = \{readFlag, writeFlag, oldFlag, textFlag, rawFlag\}$

$seqfile\text{-}merge\text{-}flags : Set\text{-}value \times Set\text{-}value \rightarrow [Set\text{-}value]$

$seqfile\text{-}merge\text{-}flags\,(given, required) \triangleq$
  let $default$ = if $rawFlag \in given$ then $\{\}$ else $\{textFlag\}$ in
  if $given \subseteq seqfile\text{-}flags$
  then $given \cup required \cup default$
  else nil

$seqfile\text{-}check\text{-}file : Char^* \times Set\text{-}value \rightarrow Exception\text{-}handler \xrightarrow{o} Enumeration\text{-}value$

$seqfile\text{-}check\text{-}file\,(sname, flags)handler \triangleq$
  implementation-defined

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations Returns *opened* if the requested read/write/text/raw operations are available on a sequential access file of the given name; otherwise returns *tooManyOpen*, *wrongPermissions*, *noRoomOnDevice*, *wrongFileType*, *noTextOperations*, *noRawOperations*, *noMixedOperations*, *alreadyOpen*, or *otherProblem*.

The procedure `OpenWrite`

If successful, the call `OpenWrite(cid,name,flags,res)` shall assign to `cid` the identity of a channel that is connected to a stored file specified by `name`; the value `opened` shall be assigned to `res`. Output mode shall be selected and the file shall be of zero length.

Inclusion of the `write` flag in the parameter `flags` shall be implied.

If the call is to succeed, a destination of the given name shall not already exist unless the flag `old` is given; if the `read` flag is included in the request, the `Reread` operation shall be available.

The effect of a reset operation on the channel shall be to truncate the file to zero length and to select output mode.

If a channel cannot be opened as required, the value of `res` shall indicate the reason, and `cid` shall identify the invalid channel.

NOTE 1 — Distinct modes in combination with `text` and/or `raw` are given by the following equivalent sets of flags:

write to a new file:	`write`	
write to a new file or a truncated existing file:	`old`	`write+old`
write to a new file, need read operations:	`write+read`	`read`
write to a new or existing file, need read operations:	`old+read`	`write+old+read`

$$OpenWrite : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} ()$$

$$OpenWrite\,(args)handler \triangleq$$

```
  let [vcid, name, flags, vres] = args in
  let sname = pure-string (name) in                                          579
  let wflags = seqfile-merge-flags (flags, {writeFlag}) in                   485
  if wflags = nil
  then (assign(vcid, badchan ()) handler;                                    150 538
        assign(vres, wrongFlags) handler                                     150
        )
  else def res = seqfile-check-write(sname, wflags) handler in               486
       if res ≠ opened
       then (assign(vcid, badchan ()) handler;                               150 538
             assign(vres, res) handler                                       150
             )
       else let tflags = wflags − {readFlag} in
            def mk-(cid, tid) = make-a-channel(SEQFILE, tflags, seqfile-ops) in   540
            if cid = badchan ()                                              538
            then (assign(vcid, badchan ()) handler;                          150 538
                  assign(vres, outOfChannels) handler                        150
                  )
            else (let tflags = wflags − {readFlag} in
                  seqfile-make-a-buffer(tid, [], sname, tflags) ;            496
                  assign(vcid, cid) handler;                                 150
                  assign(vres, opened) handler                               150
                  )
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
seqfile-check-write : Char* × Set-value → Exception-handler -o→ Enumeration-value
seqfile-check-write (sname, flags)handler △
  def res = if is-legal-filename (sname)                                    529
            then if is-existing-file (sname) ∧ oldFlag ∉ flags              529
                 then fileExists
                 else seqfile-check-file(sname, flags) handler              485
            else wrongNameFormat in
  return res
```

The procedure `OpenAppend`

If successful, the call `OpenAppend(cid,name,flags,res)` shall assign to `cid` the identity of a channel that is connected to a stored file specified by `name`; the value `opened` shall be assigned to `res`. Output mode shall be selected.

Inclusion of the `write` and `old` flags in the parameter `flags` shall be implied; a destination of the given name may already exist.

If the `read` flag is included in the request, the `Reread` operation shall be available if the call is to succeed.

The effect of a reset operation on the channel shall be to select output mode.

If a channel cannot be opened as required, the value of `res` shall indicate the reason, and `cid` shall identify the invalid channel.

NOTE 2 — Distinct modes in combination with `text` and/or `raw` are given by the following equivalent sets of flags:

write to a new or append to an existing file:	`write`	`old`	`write+old`	
write to a new or append to an existing file, need read operations:	`read`	`write+read`	`old+read`	`write+old+read`

```
OpenAppend : Arguments → Exception-handler -o→ ()
OpenAppend (args)handler △
  let [vcid, name, flags, vres] = args in
  let sname = pure-string (name) in                                         579
  let aflags = seqfile-merge-flags (flags, {writeFlag, oldFlag}) in         485
  if aflags = nil
  then (assign(vcid, badchan ()) handler;                                   150 538
        assign(vres, wrongFlags) handler                                    150
        )
  else def res = seqfile-check-append(sname, aflags) handler in             487
       if res ≠ opened
       then (assign(vcid, badchan ()) handler;                              150 538
             assign(vres, res) handler                                      150
             )
       else let tflags = aflags − {readFlag} in
            def mk-(cid, tid) = make-a-channel(SEQFILE, tflags, seqfile-ops) in   540
            if cid = badchan ()                                             538
            then (assign(vcid, badchan ()) handler;                         150 538
                  assign(vres, outOfChannels) handler                       150
                  )
            else (def file = if is-existing-file (sname)                    529
                             then read-file(name) handler else [] in        537
                  let tflags = aflags − {readFlag, writeFlag} in
                  seqfile-make-a-buffer(tid, file, sname, tflags) ;         496
                  assign(vcid, cid) handler;                                150
                  assign(vres, opened) handler                              150
                  )
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

seqfile-check-append : *Char** × *Set-value* → *Exception-handler* $\xrightarrow{o}$ *Enumeration-value*

seqfile-check-append (*sname*, *flags*)*handler* $\triangleq$
  def *res* = if *is-legal-filename* (*sname*) 529
    then *seqfile-check-file*(*sname*, *flags*) *handler* 485
    else *wrongNameFormat* in
  return *res*

The procedure `OpenRead`

If successful, the call `OpenRead(cid,name,flags,res)` shall assign to `cid` the identity of a channel that is connected to a stored file specified by `name`; the value `opened` shall be assigned to `res`. Input mode shall be selected and the read position shall correspond to the start of the file.

Inclusion of the `read` and `old` flags in the parameter `flags` shall be implied; a destination of the given name shall already exist if the call is to succeed.

If the `write` flag is included in the request, the `Rewrite` operation shall be available if the call is to succeed.

The effect of a reset operation on the channel shall be to select input mode and to set the read position to the start of the file.

If a channel cannot be opened as required, the value of `res` shall indicate the reason, and `cid` shall identify the invalid channel.

NOTE 3 — Distinct modes in combination with `text` and/or `raw` are given by the following equivalent sets of flags:

read from an existing file:	`read`	`old`	`read+old`	
read from an existing file, need write operations:	`write`	`read+write`	`old+write`	`read+old+write`

OpenRead : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

OpenRead (*args*)*handler* $\triangleq$
  let [*vcid*, *name*, *flags*, *vres*] = *args* in
  let *sname* = *pure-string* (*name*) in 579
  let *rflags* = *seqfile-merge-flags* (*flags*, {*readFlag*, *oldFlag*}) in 485
  if *rflags* = nil
  then (*assign*(*vcid*, *badchan* ()) *handler*; 150 538
    assign(*vres*, *wrongFlags*) *handler* 150
)
  else def *res* = *seqfile-check-read*(*sname*, *rflags*) *handler* in 488
    if *res* ≠ *opened*
    then (*assign*(*vcid*, *badchan* ()) *handler*; 150 538
      assign(*vres*, *res*) *handler* 150
)
    else let *tflags* = *rflags* − {*writeFlag*} in
      def *mk-* (*cid*, *tid*) = *make-a-channel*(SEQFILE, *tflags*, *seqfile-ops*) in 540
      if *cid* = *badchan* () 538
      then (*assign*(*vcid*, *badchan* ()) *handler*; 150 538
        assign(*vres*, *outOfChannels*) *handler* 150
)
      else (def *file* = *read-file*(*name*) *handler* in 537
        let *tflags* = *rflags* − {*writeFlag*} in
        seqfile-make-a-buffer(*tid*, *file*, *sname*, *tflags*) ; 496
        assign(*vcid*, *cid*) *handler*; 150
        assign(*vres*, *opened*) *handler* 150
)

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

seqfile-check-read : *Char** × *Set-value* → *Exception-handler* $\xrightarrow{o}$ *Enumeration-value*

seqfile-check-read (*sname*, *flags*)*handler* $\triangleq$
  def *res* = if *is-legal-filename* (*sname*) 529
        then if *is-existing-file* (*sname*) 529
          then *seqfile-check-file*(*sname*, *flags*) *handler* 485
          else *noSuchFile*
        else *wrongNameFormat* in
  return *res*

The procedure `IsSeqFile`

The value of the call `IsSeqFile(cid)` shall be *true* if the channel identified by `cid` is open to a rewindable sequential file, and *false* otherwise.

IsSeqFile : *Arguments* → *Exception-handler* $\xrightarrow{o}$ $\mathbb{B}$

IsSeqFile (*args*)*handler* $\triangleq$
  let [*cid*] = *args* in
  def *tid* = *table-id*(*cid*) in 539
  def *did* = *get-device-id*(*tid*) in 540
  return *did* = SEQFILE

The procedure `Reread`

If the channel identified by `cid` is open to a rewindable sequential file, the call `Reread(cid)` shall attempt to set the read position of the channel to the start of the file, and to select input mode; otherwise, the exception `wrongDevice` shall be raised.

If the operation cannot be performed, perhaps because of insufficient permissions, neither input mode nor output mode shall be selected.

Reread : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

Reread (*args*)*handler* $\triangleq$
  let [*cid*] = *args* in
  def *tid* = *table-id*(*cid*) in 539
  (*vet*(*tid*, SEQFILE) *handler*; 529
   def *rprops* = *seqfile-get-realprops*(*tid*) in 497
   def *canrr* = *can-reread*(*tid*) in 490
   if *canrr*
   then (let *nprops* = *rprops* − {*writeFlag*} + {*readFlag*} in
      set-props(*tid*, *nprops*) ; 539
      seqfile-set-position(*tid*, 1) 496
)
   else *set-props*(*tid*, *rprops*) 539
)

The procedure `Rewrite`

If the channel identified by `cid` is open to a rewindable sequential file, the call `Rewrite(cid)` shall attempt to set the write position of the channel to the start of the file, to truncate the file to zero length, and to select output mode; otherwise, the exception `wrongDevice` shall be raised.

If the operation cannot be performed, perhaps because of insufficient permissions, neither input mode nor output mode shall be selected.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Rewrite : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

Rewrite (*args*)*handler* $\triangleq$
  let [*cid*] = *args* in
  def *tid* = *table-id*(*cid*) in 539
  (*vet*(*tid*, SEQFILE) *handler*; 529
   def *rprops* = *seqfile-get-realprops*(*tid*) in 497
   def *canrw* = *can-rewrite*(*tid*) in 490
   if *canrw*
   then (let *nprops* = *rprops* − {*readFlag*} + {*writeFlag*} in
         set-props(*tid*, *nprops*) ; 539
         seqfile-set-file(*tid*, [ ]) 496
)
   else *set-props*(*tid*, *rprops*) 539
)

The procedure `Close`

If the channel identified by `cid` is open to a rewindable sequential file, the call `Close(cid)` shall close the channel and assign the value identifying the invalid channel to `cid`; otherwise, the exception `wrongDevice` shall be raised.

seqfile-Close : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

seqfile-Close (*args*)*handler* $\triangleq$
  let [*cid*] = *args* in
  def *tid* = *table-id*(*cid*) in 539
  (*vet*(*tid*, SEQFILE) *handler*; 529
   seqfile-free([*tid*]) *handler*; 495
   release-a-channel(*cid*, *tid*) ; 540
   assign(*cid*, *badchan* ()) *handler* 150 538
)

Auxiliaries

can-reread : *Table-id* $\xrightarrow{o}$ $\mathbb{B}$

can-reread (*tid*) $\triangleq$
  implementation-defined

can-rewrite : *Table-id* $\xrightarrow{o}$ $\mathbb{B}$

can-rewrite (*tid*) $\triangleq$
  implementation-defined

9.2.5.3.3 The Device Procedures of SeqFile

If supported on a channel open to a `SeqFile` device, the `SeqFile` device procedures shall have the semantics defined below; otherwise, they shall have the semantics of the corresponding device procedure for the invalid channel.

NOTE — Device procedures, other than *Free*, are invoked through corresponding procedures from `IOChan`.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The SeqFile device procedure *Look*

```
seqfile-look : Arguments → Exception-handler ⟶° ()
seqfile-look (args)handler ≙
  let [tid, vch, vres] = args in
  (check-flags(tid, {readFlag}) handler;                                  529
   def rpos = seqfile-get-position(tid) in                                496
   def file = seqfile-get-file(tid) in                                    496
   def mk-(ch, res) = if rpos > len file
                      then mk-(nil , endOfInput)
                      else let s = rest (file, rpos) in                   614
                           def choice = a-text-item(tid, s) handler in    529
                           if choice = END-LINE
                           then mk-(nil , endOfLine)
                           else mk-(choice, allRight) in
   (set-rdresult(tid, res) ;                                              539
    assign(vres, res) handler;                                            150
    if ch = nil
    then let impdep : (Char | UNDEFINED) in
         assign(vch, impdep) handler                                      150
    else assign(vch, ch) handler                                          150
    )
  )
```

The SeqFile device procedure *Skip*

```
seqfile-skip : Arguments → Exception-handler ⟶° ()
seqfile-skip (args)handler ≙
  let [tid] = args in
  (check-flags(tid, {readFlag}) handler;                                  529
   def rpos = seqfile-get-position(tid) in                                496
   def file = seqfile-get-file(tid) in                                    496
   if rpos > len file
   then chan-exception(skipAtEnd) handler                                 470
   else let s = rest (file, rpos) in                                      614
        def choice = a-text-item(tid, s) handler in                       529
        (seqfile-set-position(tid, rpos + len emit (tid, choice)) ;       496
         set-rdresult(tid, allRight)                                      539
         )
  )
```

The SeqFile device procedure *SkipLook*

```
seqfile-skiplook : Arguments → Exception-handler ⟶° ()
seqfile-skiplook (args)handler ≙
  let [tid, vch, vres] = args in
  (seqfile-skip([tid]) handler;                                           491
   seqfile-look(args) handler                                             491
   )
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The SeqFile device procedure *WriteLn*

```
seqfile-writeln : Arguments → Exception-handler →o ()
seqfile-writeln (args)handler ≜
  let [tid] = args in
  (check-flags(tid, {writeFlag}) handler;                                  529
   def file = seqfile-get-file(tid) in                                     496
   let bos = emit (tid, END-LINE) in                                       529
   let newf = file ↷ bos in
   seqfile-set-file(tid, newf)                                             496
   )
```

The SeqFile device procedure *TextRead*

```
seqfile-textread : Arguments → Exception-handler →o ()
seqfile-textread (args)handler ≜
  let [tid, ato, maxChars, vcharsRead] = args in
  (check-flags(tid, {readFlag}) handler;                                   529
   if maxChars = 0
   then (set-rdresult(tid, allRight) ;                                     539
         assign(vcharsRead, 0) handler                                     150
         )
   else def file = seqfile-get-file(tid) ,                                 496
            rpos = seqfile-get-position(tid) in                            496
        if rpos > len file
        then (set-rdresult(tid, endOfInput) ;                              539
              assign(vcharsRead, 0) handler                                150
              )
        else let s = rest (file, rpos) in                                  614
             def choice = a-text-item(tid, s) handler in                   529
             if choice = END-LINE
             then (set-rdresult(tid, endOfLine) ;                          539
                   assign(vcharsRead, 0) handler                           150
                   )
             else def mk-(ans, nchars) = chars-up-to(tid, s, maxChars) handler in   530
                  def mk-(vto, cap) = make-subarray(ato, maxChars) in      529
                  let safechars = if nchars ≤ cap
                                  then nchars else cap in
                  (let chars = ans (1, ..., safechars) in
                   (for all i ∈ inds chars
                    do assign(vto (i − 1), chars (i)) handler ;            150
                    let amount = len emit (tid, chars) in
                    seqfile-set-position(tid, rpos + amount) ;             496
                    set-rdresult(tid, allRight) ;                          539
                    assign(vcharsRead, safechars) handler                  150
                    );
                   if safechars < nchars
                   then non-mandatory-detection(NONEXISTENT) handler       397
                   else skip
                   )
  )
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The SeqFile device procedure *TextWrite*

```
seqfile-textwrite : Arguments → Exception-handler ^o→ ()
seqfile-textwrite (args)handler ≙
  let [tid, ato, maxChars] = args in
  def mk-(vfrom, cap) = make-subarray(ato, maxChars) in                    529
  (check-flags(tid, {writeFlag}) handler;                                  529
   def file = seqfile-get-file(tid) in                                     496
   let safechars = if maxChars ≤ cap then maxChars else cap in
   (if safechars > 0
    then let frmseq = map-to-seq (vfrom) in                                615
         let outs = frmseq (1, ..., safechars) in
         let bos = emit (tid, outs) in                                     529
         let newf = file ↷ bos in
         seqfile-set-file(tid, newf)                                       496
    else skip;
    if safechars < maxChars
    then non-mandatory-detection(NONEXISTENT) handler                      397
    else skip
    )
  )
```

The SeqFile device procedure *RawRead*

```
seqfile-rawread : Arguments → Exception-handler ^o→ ()
seqfile-rawread (args)handler ≙
  let [tid, ato, maxLocs, vlocsRead] = args in
  (check-flags(tid, {readFlag}) handler;                                   529
   if maxLocs = 0
   then (set-rdresult(tid, allRight) ;                                     539
         assign(vlocsRead, 0) handler                                      150
         )
   else def rpos = seqfile-get-position(tid) in                            496
        def file = seqfile-get-file(tid) in                                496
        if rpos > len file
        then (set-rdresult(tid, endOfInput) ;                              539
              assign(vlocsRead, 0) handler                                 150
              )
        else let s = rest (file, rpos) in                                  614
             let ans = up-to (s, maxLocs) in                               530
             def mk-(vto, cap) = make-subarray(ato, maxLocs) in            529
             let safelocs = if len ans ≤ cap then len ans else cap in
             (let locs = ans (1, ..., safelocs) in
              (for all i ∈ inds locs
               do assign(vto (i − 1), locs (i)) handler ;                  150
               let amount = len locs in
               seqfile-set-position(tid, rpos + amount) ;                  496
               set-rdresult(tid, allRight) ;                               539
               assign(vlocsRead, safechars) handler                        150
               );
              if safelocs < len ans
              then non-mandatory-detection(NONEXISTENT) handler            397
              else skip
              )
  )
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The SeqFile device procedure *RawWrite*

```
seqfile-rawwrite : Arguments → Exception-handler -o→ ()
seqfile-rawwrite (args)handler ≙
  let [tid, ato, maxLocs] = args in
  def mk-(vfrom, cap) = make-subarray(ato, maxLocs) in                    529
  (check-flags(tid, {writeFlag})) handler;                                529
   def file = seqfile-get-file(tid) in                                    496
   let safelocs = if maxLocs ≤ cap
                  then maxLocs
                  else cap in
   (if safelocs > 0
    then let frmseq = map-to-seq (vfrom) in                               615
         let outs = frmseq (1, ..., safelocs) in
         let newf = file ↷ outs in
         seqfile-set-file(tid, newf)                                      496
    else skip;
    if safelocs < maxLocs
    then non-mandatory-detection(NONEXISTENT) handler                     397
    else skip
    )
   )
```

The SeqFile device procedure *GetName*

```
seqfile-getname : Arguments → Exception-handler -o→ ()
seqfile-getname (args)handler ≙
  let [tid, vs] = args in
  def filename = seqfile-get-name(tid) in                                 496
  assign(vs, filename) handler                                            150
```

The SeqFile device procedure *Reset*

```
seqfile-reset : Arguments → Exception-handler -o→ ()
seqfile-reset (args)handler ≙
  let [tid] = args in
  def rprops = seqfile-get-realprops(tid) in                              497
  if writeFlag ∈ rprops
  then (let tprops = rprops − {readFlag} in
        set-props(tid, tprops) ;                                          539
        seqfile-set-file(tid, []) ;                                       496
        seqfile-flush([tid]) handler                                      495
        )
  else if readFlag ∈ rprops
       then (let tprops = rprops − {writeFlag} in
             set-props(tid, tprops) ;                                     539
             seqfile-set-position(tid, 1)                                 496
             )
       else (let tprops = rprops − {readFlag} + {writeFlag} in
             set-props(tid, tprops) ;                                     539
             seqfile-flush([tid]) handler                                 495
             )
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The SeqFile device procedure *Flush*

seqfile-flush : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

seqfile-flush (*args*)*handler* $\triangleq$
  let [*tid*] = *args* in
  def *props* = *get-props*(*tid*) in 539
  if *writeFlag* ∈ *props*
  then def *file* = *seqfile-get-file*(*tid*) in 496
       def *filename* = *seqfile-get-name*(*tid*) in 496
       write-file(*filename*, *file*) *handler* 537

The SeqFile device procedure *Free*

seqfile-free : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

seqfile-free (*args*)*handler* $\triangleq$
  let [*tid*] = *args* in
  (*seqfile-flush*([*tid*]) *handler*; 495
   seqfile-release-a-buffer(*tid*) 496
)

values

seqfile-ops =
{LOOK ↦ *seqfile-look-pid*,
SKIP ↦ *seqfile-skip-pid*,
SKIPLOOK ↦ *seqfile-skiplook-pid*,
WRITELN ↦ *seqfile-writeln-pid*,
READTEXT ↦ *seqfile-readtext-pid*,
WRITETEXT ↦ *seqfile-writetext-pid*,
READBIN ↦ *seqfile-readraw-pid*,
WRITEBIN ↦ *seqfile-writeraw-pid*,
FLUSH ↦ *seqfile-flush-pid*,
RESET ↦ *seqfile-reset-pid*,
FREE ↦ *seqfile-free-pid*}

;

seqfile-dens =
{*seqfile-look-pid* ↦ *make-proper-procden* (*seqfile-look*), 339
seqfile-skip-pid ↦ *make-proper-procden* (*seqfile-skip*), 339
seqfile-skiplook-pid ↦ *make-proper-procden* (*seqfile-skiplook*), 339
seqfile-writeln-pid ↦ *make-proper-procden* (*seqfile-writeln*), 339
seqfile-readtext-pid ↦ *make-proper-procden* (*seqfile-readtext*), 339
seqfile-writetext-pid ↦ *make-proper-procden* (*seqfile-writetext*), 339
seqfile-readraw-pid ↦ *make-proper-procden* (*seqfile-readraw*), 339
seqfile-writeraw-pid ↦ *make-proper-procden* (*seqfile-writeraw*), 339
seqfile-flush-pid ↦ *make-proper-procden* (*seqfile-flush*), 339
seqfile-reset-pid ↦ *make-proper-procden* (*seqfile-reset*), 339
seqfile-free-pid ↦ *make-proper-procden* (*seqfile-free*)} 339

annotations The mapping *seqfile-dens*, from elements of type *Procedure-id* to the corresponding procedure denotations, is part of the mapping *dens* used in the construction of the abstract implementation module — see Annex B.

9.2.5.3.4 The State for SeqFile

types

FileSeq = *Elementary-loc-value**

state *SeqFile* of
  seqfile-files : *Table-id* $\xrightarrow{m}$ *FileSeq*
  seqfile-positions : *Table-id* $\xrightarrow{m}$ N
  seqfile-names : *Table-id* $\xrightarrow{m}$ *Char**
  seqfile-realprops : *Table-id* $\xrightarrow{m}$ *Set-value*

  inv *mk-SeqFile* (*files*, *positions*, *names*) $\triangleq$
    dom *files* = dom *positions* = dom *names*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

init *mk-SeqFile* (*files*, *positions*, *names*, *realprops*) $\triangleq$
  files = $\{\mapsto\}$ $\wedge$
  positions = $\{\mapsto\}$ $\wedge$
  names = $\{\mapsto\}$ $\wedge$
  realprops = $\{\mapsto\}$
end

Auxiliaries

Access to the state for SeqFile

seqfile-make-a-buffer : *Table-id* $\times$ *FileSeq* $\times$ *Char** $\times$ *Set-value* $\xrightarrow{o}$ ()

seqfile-make-a-buffer (*tid*, *file*, *wpos*, *name*, *flags*) $\triangleq$
  || (*seqfile-positions* := *seqfile-positions* $\cup$ {*tid* $\mapsto$ 1},
    seqfile-files := *seqfile-files* $\cup$ {*tid* $\mapsto$ *file*},
    seqfile-names := *seqfile-names* $\cup$ {*tid* $\mapsto$ *name*},
    seqfile-realprops := *seqfile-realprops* $\cup$ {*tid* $\mapsto$ *flags*})

seqfile-release-a-buffer : *Table-id* $\xrightarrow{o}$ ()

seqfile-release-a-buffer (*tid*) $\triangleq$
  || (*seqfile-positions* := {*tid*} $\lhd$ *seqfile-positions*;
    seqfile-files := {*tid*} $\lhd$ *seqfile-files*,
    seqfile-names := {*tid*} $\lhd$ *seqfile-names*,
    seqfile-realprops := {*tid*} $\lhd$ *seqfile-realprops*)

seqfile-set-position : *Table-id* $\times$ **N** $\xrightarrow{o}$ ()

seqfile-set-position (*tid*, *pos*) $\triangleq$
  seqfile-positions (*tid*) := *pos*

seqfile-get-position : *Table-id* $\xrightarrow{o}$ **N**

seqfile-get-position (*tid*) $\triangleq$
  def *pos* = *seqfile-positions* (*tid*) in
  return *pos*

seqfile-set-file : *Table-id* $\times$ *FileSeq* $\xrightarrow{o}$ ()

seqfile-set-file (*tid*, *file*) $\triangleq$
  seqfile-files (*tid*) := *file*

seqfile-get-file : *Table-id* $\xrightarrow{o}$ *FileSeq*

seqfile-get-file (*tid*) $\triangleq$
  def *file* = *seqfile-files* (*tid*) in
  return *file*

seqfile-set-name : *Table-id* $\times$ *Char** $\xrightarrow{o}$ ()

seqfile-set-name (*tid*, *name*) $\triangleq$
  seqfile-names (*tid*) := *name*

seqfile-get-name : *Table-id* $\xrightarrow{o}$ *Char**

seqfile-get-name (*tid*) $\triangleq$
  def *name* = *seqfile-names* (*tid*) in

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

return *name*

seqfile-get-realprops : *Table-id* $\xrightarrow{o}$ *Set-value*

seqfile-get-realprops (*tid*) $\triangleq$
def *props* = *seqfile-realprops* (*tid*) in
return *props*

9.2.5.4 The Module RndFile

The module `RndFile` provides facilities that allow for obtaining and releasing channels that are connected to named random access files.

9.2.5.4.1 The Definition Module of RndFile

```
DEFINITION MODULE RndFile;

  (* Random access files *)

IMPORT IOChan, ChanConsts, SYSTEM;

TYPE
  ChanId = IOChan.ChanId;
  FlagSet = ChanConsts.FlagSet;
  OpenResults = ChanConsts.OpenResults;

  (* Accepted singleton values of FlagSet *)

CONST
  read = FlagSet{ChanConsts.readFlag};   (* input operations are requested/available *)
  write = FlagSet{ChanConsts.writeFlag}; (* output operations are requested/available *)
  old = FlagSet{ChanConsts.oldFlag};     (* a file may/must/did exist before the channel is
                                            opened *)
  text = FlagSet{ChanConsts.textFlag};   (* text operations are requested/available *)
  raw = FlagSet{ChanConsts.rawFlag};     (* raw operations are requested/available *)

PROCEDURE OpenOld (VAR cid: ChanId; name: ARRAY OF CHAR; flags: FlagSet;
                   VAR res: OpenResults);
  (* Attempts to obtain and open a channel connected to a stored random access file of the
     given name.
     The old flag is implied; without the write flag, read is implied; without the text
     flag, raw is implied.
     If successful, assigns to cid the identity of the opened channel, assigns the value
     opened to res, and sets the read/write position to the start of the file.
     If a channel cannot be opened as required, the value of res indicates the reason, and
     cid identifies the invalid channel.
  *)

PROCEDURE OpenClean (VAR cid: ChanId; name: ARRAY OF CHAR; flags: FlagSet;
                     VAR res: OpenResults);
  (* Attempts to obtain and open a channel connected to a stored random access file of the
     given name.
     The write flag is implied; without the text flag, raw is implied.
     If successful, assigns to cid the identity of the opened channel, assigns the value
     opened to res, and truncates the file to zero length.
     If a channel cannot be opened as required, the value of res indicates the reason, and
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
    cid identifies the invalid channel.
  *)

PROCEDURE IsRndFile (cid: ChanId): BOOLEAN;
  (* Tests if the channel identified by cid is open to a random access file. *)

PROCEDURE IsRndFileException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional execution state because
     of the raising of a RndFile exception; otherwise returns FALSE.
  *)

CONST
  FilePosSize = <implementation-defined whole number greater than zero>;

TYPE
  FilePos = ARRAY [1 .. FilePosSize] OF SYSTEM.LOC;

PROCEDURE StartPos (cid: ChanId): FilePos;
  (* If the channel identified by cid is not open to a random access file, the exception
     wrongDevice is raised; otherwise returns the position of the start of the file.
  *)

PROCEDURE CurrentPos (cid: ChanId): FilePos;
  (* If the channel identified by cid is not open to a random access file, the exception
     wrongDevice is raised; otherwise returns the position of the current read/write
     position.
  *)

PROCEDURE EndPos (cid: ChanId): FilePos;
  (* If the channel identified by cid is not open to a random access file, the exception
     wrongDevice is raised; otherwise returns the first position after which there have been
     no writes.
  *)

PROCEDURE NewPos (cid: ChanId; chunks: INTEGER; chunkSize: CARDINAL; from: FilePos): FilePos;
  (* If the channel identified by cid is not open to a random access file, the exception
     wrongDevice is raised; otherwise returns the position (chunks * chunkSize) relative to
     the position given by from, or raises the exception posRange if the required position
     cannot be represented as a value of type FilePos.
  *)

PROCEDURE SetPos (cid: ChanId; pos: FilePos);
  (* If the channel identified by cid is not open to a random access file, the exception
     wrongDevice is raised; otherwise sets the read/write position to the value given by
     pos.
  *)

PROCEDURE Close (VAR cid: ChanId);
  (* If the channel identified by cid is not open to a random access file, the exception
     wrongDevice is raised; otherwise closes the channel, and assigns the value identifying
     the invalid channel to cid.
  *)

END RndFile.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.5.4.2 The Procedures of RndFile

Channels opened by the module `RndFile` shall have an associated read/write position in the corresponding random-access file. The read/write position shall be at the start of the file after opening, or after a reset operation on the channel. It shall be moved forward by the number of positions occupied by data that are taken from the file by an input operation, or written to the file by an output operation.

NOTE 1 — The implementation-defined type `FilePos` has been specified in a way that enables values of this type to be read from or written to a file, while maintaining a degree of opacity for the type.

A random-access file shall have a length corresponding to the position after the highest read/write position at which data have been written. This length shall be zero if no data have been written to the file. If the read/write position is set at the current length, either implicitly on an input or output operation, or explicitly by a positioning operation, the effect of an input operation shall be as if the input stream had ended. A write at that position shall, if necessary, attempt to allocate more physical storage for the file.

In a request to open a random-access file, the flags `read`, `write`, `old`, `text`, and `raw` shall apply. If `text` is not included in the request parameter `flags`, inclusion of `raw` shall be implied.

types

Filepos = implementation-defined

annotations Values of this type represent array values of the location type with an index range from 0 to an implementation-defined value.

values

rndfile-flags = {*readFlag*, *writeFlag*, *oldFlag*, *textFlag*, *rawFlag*}

rndfile-merge-flags : *Set-value* × *Set-value* → [*Set-value*]

rndfile-merge-flags (*given*, *required*) △

let *default* = if *textFlag* ∈ *given* then { } else {*rawFlag*} in
if *given* ⊆ *rndfile-flags*
then *given* ∪ *required* ∪ *default*
else nil

rndfile-check-file : *Char** × *Set-value* → *Exception-handler* $\xrightarrow{o}$ *Enumeration-value*

rndfile-check-file (*sname*, *flags*)*handler* △

implementation-defined

annotations Returns *opened* if the requested read/write/text/raw operations are available on a random access file of the given name; otherwise returns *tooManyOpen*, *wrongPermissions*, *noRoomOnDevice*, *wrongFileType*, *noTextOperations*, *noRawOperations*, *noMixedOperations*, *alreadyOpen*, or *otherProblem*.

The procedure `OpenOld`

If successful, the call `OpenOld(cid,name,flags,res)` shall assign to `cid` the identity of a channel that is connected to a random access file specified by `name`; the value `opened` shall be assigned to `res`. The read/write position shall correspond to the start of the file.

Inclusion of the `old` flag in the parameter `flags` shall be implied; a file of the given name shall already exist if the call is to succeed.

If the `write` flag is not included in the request, inclusion of the `read` flag shall be implied.

If a channel cannot be opened as required, the value of `res` shall indicate the reason, and `cid` shall identify the invalid channel.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NOTE 2 — Distinct modes in combination with `text` and/or `raw` are given by the following equivalent sets of flags:

read from an existing file:	`read`	`old`	`read+old`
write to an existing file:	`write`	`write+old`	
read/write an existing file:	`read+write`	`read+write+old`	

```
OpenOld : Arguments → Exception-handler ⟶° ()
OpenOld (args)handler ≙
  let [vcid, name, flags, vres] = args in
  let sname = pure-string (name) in                                              579
  let oflags = rndfile-merge-flags (flags, (if writeFlag ∉ flags then {readFlag, oldFlag} else {oldFlag})) in   499
  if oflags = nil
  then (assign(vcid, badchan ()) handler;                                        150 538
        assign(vres, wrongFlags) handler                                         150
        )
  else def res = rndfile-check-old(sname, oflags) handler in                     500
       if res ≠ opened
       then (assign(vcid, badchan ()) handler;                                   150 538
             assign(vres, res) handler                                           150
             )
       else def mk-(cid, tid) = make-a-channel(RNDFILE, oflags, rndfile-ops) in  540
            if cid = badchan ()                                                  538
            then (assign(vcid, badchan ()) handler;                              150 538
                  assign(vres, outOfChannels) handler                            150
                  )
            else (def file = read-file(name) handler in                          537
                  rndfile-make-a-buffer(tid, file, sname, oflags) ;              510
                  assign(vcid, cid) handler;                                     150
                  assign(vres, opened) handler                                   150
                  )

rndfile-check-old : Char* × Set-value → Exception-handler ⟶° Enumeration-value
rndfile-check-old (sname, flags)handler ≙
  def res = if is-legal-filename (sname)                                         529
            then if is-existing-file (sname)                                     529
                 then rndfile-check-file(sname, flags) handler                   499
                 else noSuchFile
            else wrongNameFormat in
  return res
```

The procedure `OpenClean`

If successful, the call `OpenClean(cid,name,flags,res)` shall assign to `cid` the identity of a channel that is connected to a random access file specified by `name`; the value `opened` shall be assigned to `res`. The file shall be truncated to zero length.

Inclusion of the `write` flag in the parameter `flags` shall be implied; a destination of the given name shall not already exist unless the flag `old` is given.

If a channel cannot be opened as required, the value of `res` shall indicate the reason, and `cid` shall identify the invalid channel.

NOTE 3 — Distinct modes in combination with `text` and/or `raw` are given by the following equivalent sets of flags:

write to a new file:	`write`	
write to a new file or a truncated existing file:	`old`	`write+old`
write to a new file, read operations are needed:	`read`	`write+read`
write to a new file or a truncated existing file, read operations are needed:	`old+read`	`write+old+read`

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
OpenClean : Arguments → Exception-handler ⟶° ()
OpenClean (args)handler ≙
  let [vcid, name, flags, vres] = args in
  let sname = pure-string (name) in                                          579
  let cflags = rndfile-merge-flags (flags, {writeFlag}) in                   499
  if cflags = nil
  then (assign(vcid, badchan ()) handler;                                    150 538
        assign(vres, wrongFlags) handler                                     150
        )
  else def res = rndfile-check-clean(sname, cflags) handler in               501
       if res ≠ opened
       then (assign(vcid, badchan ()) handler;                               150 538
             assign(vres, res) handler                                       150
             )
       else def mk-(cid, tid) = make-a-channel(RNDFILE, cflags, rndfile-ops) in   540
            if cid = badchan ()                                              538
            then (assign(vcid, badchan ()) handler;                          150 538
                  assign(vres, outOfChannels) handler                        150
                  )
            else (rndfile-make-a-buffer(tid, {↦}, sname, cflags) ;           510
                  assign(vcid, cid) handler;                                 150
                  assign(vres, opened) handler                               150
                  )

rndfile-check-clean : Char* × Set-value → Exception-handler ⟶° Enumeration-value
rndfile-check-clean (sname, flags)handler ≙
  def res = if is-legal-filename (sname)                                     529
            then if is-existing-file (sname) ∧ oldFlag ∉ flags               529
                 then fileExists
                 else rndfile-check-file(sname, flags) handler               499
            else wrongNameFormat in
  return res
```

The procedure `IsRndFile`

The value of the call `IsRndFile(cid)` shall be *true* if the channel identified by `cid` is open to a random access file, and *false* otherwise.

```
IsRndFile : Arguments → Exception-handler ⟶° 𝔹
IsRndFile (args)handler ≙
  let [cid] = args in
  def tid = table-id(cid) in                                                 539
  def did = get-device-id(tid) in                                            540
  return did = RNDFILE
```

The procedure `IsRndFileException`

If the calling coroutine is in the state of exceptional execution because of the raising of the `RndFile` exception, the value of the call `IsRndFileException()` shall be *true*; otherwise the value shall be *false*.

```
IsRndFileException : Arguments → Exception-handler ⟶° 𝔹
IsRndFileException (args)handler ≙
  def source = rndfile-source in
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  def res = m-IsCurrentSource([source]) in                                    386
  return res
```

The procedure `StartPos`

If the channel identified by `cid` is open to a random access file, the value of the call `StartPos(cid)` shall be the position of the start of the file; otherwise the exception `wrongDevice` shall be raised.

StartPos : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Filepos*

```
StartPos (args)handler ≜
  let [cid] = args in
  def tid = table-id(cid) in                                                  539
  (vet(tid, RNDFILE) handler;                                                 529
   pos-to-filepos(tid, 1)                                                     504
   )
```

The procedure `CurrentPos`

If the channel identified by `cid` is open to a random access file, the value of the call `CurrentPos(cid)` shall be the current read/write position of the file; otherwise the exception `wrongDevice` shall be raised.

CurrentPos : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Filepos*

```
CurrentPos (args)handler ≜
  let [cid] = args in
  def tid = table-id(cid) in                                                  539
  (vet(tid, RNDFILE) handler;                                                 529
   def wpos = rndfile-get-position(tid) in                                    511
   pos-to-filepos(tid, wpos)                                                  504
   )
```

The procedure `EndPos`

If the channel identified by `cid` is open to a random access file, the value of the call `EndPos(cid)` shall be the first position in the file at or after which no data have been written; otherwise the exception `wrongDevice` shall be raised.

EndPos : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Filepos*

```
EndPos (args)handler ≜
  let [cid] = args in
  def tid = table-id(cid) in                                                  539
  (vet(tid, RNDFILE) handler;                                                 529
   (def file = rndfile-get-file(tid) in                                       511
    let end = card dom file + 1 in
    pos-to-filepos(tid, end)                                                  504
    )
   )
```

The procedure `NewPos`

If the channel identified by `cid` is open to a random access file, the value of the call `NewPos(cid,chunks,chunkSize,from)` shall be the read/write position `chunks` × `chunkSize` places relative to the position in the file given by the value of `from`; otherwise, the exception `wrongDevice` shall be raised. The `RndFile` exception shall be raised if the required position cannot be represented as a value of the type `FilePos`.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NOTE 4 — Calculation of the position in a random access file at which to issue text operations is dependent upon knowledge of the external representation of text items in a particular file; the amount by which the read/write position is moved as a result of a text operation may vary depending upon the item that is read or written.

For raw operations, the read/write position is always moved by a value equal to the storage size of variables of the type of the item read or written.

NewPos : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Filepos*

NewPos (*args*)*handler* $\triangleq$
  let [*cid*, *chunks*, *chunkSize*, *from*] = *args* in
  def *tid* = *table-id*(*cid*) in 539
  (*vet*(*tid*, RNDFILE) *handler*; 529
   let *opos* = *filepos-to-pos* (*tid*, *from*) in 504
   let *npos* = *opos* + (*chunks* × *chunkSize*) in
   def *nfpos* = *pos-to-filepos*(*tid*, *npos*) in 504
   if *npos* ≤ 0
   then *rndfile-exception*() *handler* 509
   else return *nfpos*
)

The procedure SetPos

If the channel identified by `cid` is open to a random access file, the call `SetPos(cid,pos)` shall set the read/write position for the file to the position given by the value of `pos`; otherwise the exception `wrongDevice` shall be raised.

If the position given by the value of `pos` is beyond the value returned by a call of `EndPos`, '`read <= IOChan.Flags()`' shall be *false*, and a call of an input operation shall raise the exception `notAvailable` — the value of '`write <= IOChan.Flags()`' shall be implementation-defined and shall correspond to the availability of output operations in this case. If data are subsequently written at such a position, those positions that have not been written to shall be filled with implementation-defined padding values.

NOTE 5 — Setting the read/write position beyond the value returned by `EndPos` does not of itself affect the size of the file.

SetPos : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

SetPos (*args*)*handler* $\triangleq$
  let [*cid*, *pos*] = *args* in
  def *tid* = *table-id*(*cid*) in 539
  (*vet*(*tid*, RNDFILE) *handler*; 529
   def *props* = *get-props*(*tid*) in 539
   let *wpos* = *filepos-to-pos* (*tid*, *pos*) in 504
   (def *file* = *rndfile-get-file*(*tid*) in 511
    let *end* = card dom *file* + 1 in
    def *nprops* = if *wpos* > *end*
                   then def *canwrite* = *can-write-here*(*tid*, *wpos*, *end*) in 503
                        let *remove* = if *canwrite* then {*readFlag*} else {*readFlag*, *writeFlag*} in
                        props − *remove*
                   else def *rprops* = *rndfile-get-realprops*(*tid*) in *rprops* in 511
    set-props(*tid*, *nprops*) ; 539
    rndfile-set-position(*tid*, *pos*) 511
)
)

can-write-here : *Table-id* × ℕ × ℕ $\xrightarrow{o}$ 𝔹

can-write-here (*tid*, *wpos*, *end*) $\triangleq$
  implementation-defined

annotations Returns *true* only if writing on *tid* is allowed at *wpos* > *end*.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The procedure `Close`

If the channel identified by `cid` is open to a random access file, the call `Close(cid)` shall close the channel and assign the value identifying the invalid channel to `cid`; otherwise, the exception `wrongDevice` shall be raised.

rndfile-Close : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

rndfile-Close (*args*)*handler* $\triangleq$
  let [*cid*] = *args* in
  def *tid* = *table-id*(*cid*) in 539
  (*vet*(*tid*, RNDFILE) *handler*; 529
   rndfile-free([*tid*]) *handler*; 509
   release-a-channel(*cid*, *tid*) ; 540
   assign(*cid*, *badchan* ()) *handler* 150 538
)

Auxiliaries

pos-to-filepos : *Table-id* × N $\xrightarrow{o}$ *Filepos*

pos-to-filepos (*tid*, *n*) $\triangleq$
  if ∃ *fp* : *Filepos* · *filepos-to-pos* (*tid*, *fp*) = *n* 504
  then return ι *fp* : *Filepos* · *filepos-to-pos* (*tid*, *fp*) = *n* 504
  else *rndfile-exception*() *handler* 509

annotations The iota expression selects the unique value of *Filepos* that corresponds to *n*.

filepos-to-pos : *Table-id* × *Filepos* → N

filepos-to-pos (*tid*, *fp*) $\triangleq$
  implementation-defined

9.2.5.4.3 The Device Procedures of RndFile

If supported on a channel open to a `RndFile` device, the `RndFile` device procedures shall have the semantics defined below; otherwise, they shall have the semantics of the corresponding device procedure for the invalid channel.

NOTE — Device procedures, other than *Free*, are invoked through corresponding procedures from `IOChan`.

The `RndFile` device procedure *Look*

rndfile-look : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

rndfile-look (*args*)*handler* $\triangleq$
  let [*tid*, *vch*, *vres*] = *args* in
  (*check-flags*(*tid*, {*readFlag*}) *handler*; 529
   def *wpos* = *rndfile-get-position*(*tid*) in 511
   def *file* = *rndfile-get-file*(*tid*) in 511
   def *mk-* (*ch*, *res*) = if *wpos* > card dom *file*
                then *mk-* (nil , *endOfInput*)
                else let *ins* = *map-to-seq* (*file*) in 615
                     let *s* = *rest* (*ins*, *wpos*) in 614
                     def *choice* = *a-text-item*(*tid*, *s*) *handler* in if *choice* = END-LINE 529
                     then *mk-* (nil , *endOfLine*)
                     else *mk-* (*choice*, *allRight*) in
  (*set-rdresult*(*tid*, *res*) ; 539
   assign(*vres*, *res*) *handler*; 150
   if *ch* = nil
   then let *impdep* : (*Char* | UNDEFINED) in
        assign(*vch*, *impdep*) *handler* 150

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
    else assign(vch, ch) handler                                              150
   )
  )
```

The RndFile device procedure *Skip*

```
rndfile-skip : Arguments → Exception-handler ⟶° ()
rndfile-skip (args)handler ≙
  let [tid] = args in
  (check-flags(tid, {readFlag}) handler;                                      529
   def wpos = rndfile-get-position(tid) in                                    511
   def file = rndfile-get-file(tid) in                                        511
   if wpos > card dom file
   then chan-exception(skipAtEnd) handler                                     470
   else let ins = map-to-seq (file) in                                        615
        let s = rest (ins, wpos) in                                           614
        def choice = a-text-item(tid, s) handler in                           529
        (rndfile-set-position(tid, wpos + len emit (tid, choice)) ;           511
         set-rdresult(tid, allRight)                                          539
         )
  )
```

The RndFile device procedure *SkipLook*

```
rndfile-skiplook : Arguments → Exception-handler ⟶° ()
rndfile-skiplook (args)handler ≙
  let [tid, vch, vres] = args in
  (rndfile-skip([tid]) handler;                                               505
   rndfile-look(args) handler                                                 504
   )
```

The RndFile device procedure *WriteLn*

```
rndfile-writeln : Arguments → Exception-handler ⟶° ()
rndfile-writeln (args)handler ≙
  let [tid] = args in
  (check-flags(tid, {writeFlag}) handler;                                     529
   def wpos = rndfile-get-position(tid) in                                    511
   def file = rndfile-get-file(tid) in                                        511
   let bos = emit (tid, END-LINE) in                                          529
   let os = seq-to-map (wpos, bos) in                                         615
   let upf = file † os in
   let newf = pad (upf) in                                                    509
   (rndfile-set-file(tid, newf) ;                                             511
    rndfile-set-position(tid, wpos + len bos)                                 511
    )
  )
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The RndFile device procedure *TextRead*

```
rndfile-textread : Arguments → Exception-handler →o ()
rndfile-textread (args)handler ≙
  let [tid, ato, maxChars, vcharsRead] = args in
  (check-flags(tid, {readFlag}) handler;                                    529
   if maxChars = 0
   then (set-rdresult(tid, allRight) ;                                      539
         assign(vcharsRead, 0) handler                                      150
        )
   else def wpos = rndfile-get-position(tid) in                             511
        def file = rndfile-get-file(tid) in                                 511
        if wpos > card dom file
        then (set-rdresult(tid, endOfInput) ;                               539
              assign(vcharsRead, 0) handler                                 150
             )
        else let ins = map-to-seq (file) in                                 615
             let s = rest (ins, wpos) in                                    614
             def choice = a-text-item(tid, s) handler in                    529
             if choice = END-LINE
             then (set-rdresult(tid, endOfLine) ;                           539
                   assign(vcharsRead, 0) handler                            150
                  )
             else def mk-(ans, nchars) = chars-up-to(tid, s, maxChars) handler in   530
                  def mk-(vto, cap) = make-subarray(ato, maxChars) in       529
                  let safechars = if nchars ≤ cap
                                  then nchars else cap in
                  (let chars = ans (1, ..., safechars) in
                   (for all i ∈ inds chars
                    do assign(vto (i − 1), chars (i)) handler ;             150
                    let amount = len emit (tid, chars) in
                    rndfile-set-position(tid, wpos + amount) ;              511
                    set-rdresult(tid, allRight) ;                           539
                    assign(vcharsRead, safechars) handler                   150
                    );
                   if safechars < nchars
                   then non-mandatory-detection(NONEXISTENT) handler        397
                   else skip
                   )
  )
```

The RndFile device procedure *TextWrite*

```
rndfile-textwrite : Arguments → Exception-handler →o ()
rndfile-textwrite (args)handler ≙
  let [tid, ato, maxChars] = args in
  def mk-(vfrom, cap) = make-subarray(ato, maxChars) in                     529
  (check-flags(tid, {writeFlag}) handler;                                   529
   def wpos = rndfile-get-position(tid) in                                  511
   def file = rndfile-get-file(tid) in                                      511
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  let safechars = if maxChars ≤ cap then maxChars else cap in
  (if safechars > 0
   then let frmseq = map-to-seq (vfrom) in                                    615
        let outs = frmseq (1, ..., safechars) in
        let bos = emit (tid, outs) in                                         529
        let os = seq-to-map (wpos, bos) in                                    615
        let upf = file † os in
        let newf = pad (upf) in                                               509
        (rndfile-set-file(tid, newf) ;                                        511
         rndfile-set-position(tid, wpos + len bos)                            511
         )
   else skip;
   if safechars < maxChars
   then non-mandatory-detection(NONEXISTENT) handler                          397
   else skip
   )
  )
```

The `RndFile` device procedure *RawRead*

```
rndfile-rawread : Arguments → Exception-handler →o ()
rndfile-rawread (args)handler △
  let [tid, ato, maxLocs, vlocsRead] = args in
  (check-flags(tid, {readFlag}) handler;                                      529
   if maxLocs = 0
   then (set-rdresult(tid, allRight) ;                                        539
         assign(vlocsRead, 0) handler                                         150
         )
   else def wpos = rndfile-get-position(tid) in                               511
        def file = rndfile-get-file(tid) in                                   511
        if wpos > card dom file
        then (set-rdresult(tid, endOfInput) ;                                 539
              assign(vlocsRead, 0) handler                                    150
              )
        else let ins = map-to-seq (file) in                                   615
             let s = rest (ins, wpos) in                                      614
             let ans = up-to (s, maxLocs) in                                  530
             def mk-(vto, cap) = make-subarray(ato, maxLocs) in               529
             let safelocs = if len ans ≤ cap then len ans else cap in
             (let locs = ans (1, ..., safelocs) in
              (for all i ∈ inds locs
               do assign(vto (i − 1), locs (i)) handler ;                     150
               let amount = len locs in
               rndfile-set-position(tid, wpos + amount) ;                     511
               set-rdresult(tid, allRight) ;                                  539
               assign(vlocsRead, safechars) handler                           150
               );
              if safelocs < len ans
              then non-mandatory-detection(NONEXISTENT) handler               397
              else skip
              )
  )
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The RndFile device procedure *RawWrite*

```
rndfile-rawwrite : Arguments → Exception-handler →o ()
rndfile-rawwrite (args)handler ≙
  let [tid, ato, maxLocs] = args in
  def mk-(vfrom, cap) = make-subarray(ato, maxLocs) in                    529
  (check-flags(tid, {writeFlag}) handler;                                 529
   def wpos = rndfile-get-position(tid) in                                511
   def file = rndfile-get-file(tid) in                                    511
   let safelocs = if maxLocs ≤ cap
                  then maxLocs
                  else cap in
   (if safelocs > 0
    then let frmseq = map-to-seq (vfrom) in                               615
         let outs = frmseq (1, ..., safelocs) in
         let os = seq-to-map (wpos, outs) in                              615
         let upf = file † os in
         let newf = pad (upf) in                                          509
         (rndfile-set-file(tid, newf) ;                                   511
          rndfile-set-position(tid, wpos + len bos)                       511
         )
    else skip;
    if safelocs < maxLocs
    then non-mandatory-detection(NONEXISTENT) handler                     397
    else skip
   )
  )
```

The RndFile device procedure *GetName*

```
rndfile-getname : Arguments → Exception-handler →o ()
rndfile-getname (args)handler ≙
  let [tid, vs] = args in
  def filename = rndfile-get-name(tid) in                                 511
  assign(vs, filename) handler                                            150
```

The RndFile device procedure *Reset*

```
rndfile-reset : Arguments → Exception-handler →o ()
rndfile-reset (args)handler ≙
  let [tid] = args in
  (rndfile-flush([tid]) handler;                                          508
   rndfile-set-position(tid, 1) ;                                         511
   def rprops = rndfile-get-realprops(tid) in                             511
   set-props(tid, rprops)                                                 539
  )
```

The RndFile device procedure *Flush*

```
rndfile-flush : Arguments → Exception-handler →o ()
rndfile-flush (args)handler ≙
  let [tid] = args in
  def props = get-props(tid) in                                           539
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  if writeFlag ∈ props
  then def file = rndfile-get-file(tid) in                                    511
       def filename = rndfile-get-name(tid) in                                511
       write-file(filename, file) handler                                     537
```

The RndFile device procedure *Free*

```
rndfile-free : Arguments → Exception-handler →o ()
rndfile-free (args)handler △
  let [tid] = args in
  (rndfile-flush([tid]) handler;                                              508
   rndfile-release-a-buffer(tid)                                              510
  )
```

values

```
    rndfile-ops =                          ;   rndfile-dens =
  {LOOK ↦ rndfile-look-pid,                    {rndfile-look-pid ↦ make-proper-procden (rndfile-look),          339
  SKIP ↦ rndfile-skip-pid,                     rndfile-skip-pid ↦ make-proper-procden (rndfile-skip),           339
  SKIPLOOK ↦ rndfile-skiplook-pid,             rndfile-skiplook-pid ↦ make-proper-procden (rndfile-skiplook),   339
  WRITELN ↦ rndfile-writeln-pid,               rndfile-writeln-pid ↦ make-proper-procden (rndfile-writeln),     339
  READTEXT ↦ rndfile-readtext-pid,             rndfile-readtext-pid ↦ make-proper-procden (rndfile-readtext),   339
  WRITETEXT ↦ rndfile-writetext-pid,           rndfile-writetext-pid ↦ make-proper-procden (rndfile-writetext), 339
  READBIN ↦ rndfile-readraw-pid,               rndfile-readraw-pid ↦ make-proper-procden (rndfile-readraw),     339
  WRITEBIN ↦ rndfile-writeraw-pid,             rndfile-writeraw-pid ↦ make-proper-procden (rndfile-writeraw),   339
  FLUSH ↦ rndfile-flush-pid,                   rndfile-flush-pid ↦ make-proper-procden (rndfile-flush),         339
  RESET ↦ rndfile-reset-pid,                   rndfile-reset-pid ↦ make-proper-procden (rndfile-reset),         339
  FREE ↦ rndfile-free-pid}                     rndfile-free-pid ↦ make-proper-procden (rndfile-free)}           339
```

annotations The mapping *rndfile-dens*, from elements of type *Procedure-id* to the corresponding procedure denotations, is part of the mapping *dens* used in the construction of the abstract implementation module — see Annex B.

Auxiliaries

values

```
  padding-value : Elementary-loc-value = implementation-defined

pad : File → File
pad (file) △
  let pad = padding-value in
  let padding = {n ↦ pad | n ∈ {0, ..., maxs (dom file)}} in                  614
  padding † file
```

annotations Fill any gaps in a file with implementation-defined padding.

9.2.5.4.4 Exceptions Raised by RndFile

Following the detection of the exception defined for this module, the `RndFile` exception shall be raised.

The character string associated with the `RndFile` exception shall be implementation-defined.

```
rndfile-exception : () → Exception-handler →o ()
rndfile-exception ()handler △
  def source = rndfile-source in
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  def message = rndfile-exception-string() in                                  510
  let number : N in
  m-RAISE([source, number, message]) handler                                   383
```

```
rndfile-exception-string : () →o Message
rndfile-exception-string () ≜
  implementation-defined
```

annotations The result is an implementation-defined string value.

9.2.5.4.5 The Initialization of RndFile

The initialization of the module `RndFile`, shall allocate a unique value of the exception source type to identify the `RndFile` exception.

```
rndfile-initialization : () → Exception-handler →o ()
rndfile-initialization ()handler ≜
  def source = allocate-source() in                                            342
  if source ≠ nil
  then rndfile-source := source
  else mandatory-detection(NO-EXCEPTION-SOURCE) handler                        396
```

9.2.5.4.6 The State for RndFile

types

```
  FileRnd = N →m Elementary-loc-value
```

```
state RndFile of
  rndfile-files     : Table-id →m FileRnd
  rndfile-positions : Table-id →m N
  rndfile-names     : Table-id →m Char*
  rndfile-realprops : Table-id →m Set-value
  rndfile-source    : Exception-source
  inv mk-RndFile (files, positions, names) ≜
      dom files = dom positions = dom names
  init mk-RndFile (files, positions, names, realprops) ≜
      files = {↦} ∧
      positions = {↦} ∧
      names = {↦} ∧
      realprops = {↦}
end
```

Auxiliaries

Access to the state for RndFile

```
rndfile-make-a-buffer : Table-id × FileRnd × Char* × Set-value →o ()
rndfile-make-a-buffer (tid, file, wpos, name, flags) ≜
  || (rndfile-positions := rndfile-positions ∪ {tid ↦ 1},
      rndfile-files := rndfile-files ∪ {tid ↦ file},
      rndfile-names := rndfile-names ∪ {tid ↦ name},
      rndfile-realprops := rndfile-realprops ∪ {tid ↦ flags})
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

$rndfile\text{-}release\text{-}a\text{-}buffer : Table\text{-}id \xrightarrow{o} ()$

$rndfile\text{-}release\text{-}a\text{-}buffer\,(tid) \triangleq$
$\quad \| \; (rndfile\text{-}positions := \{tid\} \lhd rndfile\text{-}positions;$
$\quad\quad rndfile\text{-}files := \{tid\} \lhd rndfile\text{-}files,$
$\quad\quad rndfile\text{-}names := \{tid\} \lhd rndfile\text{-}names,$
$\quad\quad rndfile\text{-}realprops := \{tid\} \lhd rndfile\text{-}realprops)$

$rndfile\text{-}set\text{-}position : Table\text{-}id \times \mathbf{N} \xrightarrow{o} ()$

$rndfile\text{-}set\text{-}position\,(tid, pos) \triangleq$
$\quad rndfile\text{-}positions\,(tid) := pos$

$rndfile\text{-}get\text{-}position : Table\text{-}id \xrightarrow{o} \mathbf{N}$

$rndfile\text{-}get\text{-}position\,(tid) \triangleq$
$\quad \mathsf{def}\ pos = rndfile\text{-}positions\,(tid)\ \mathsf{in}$
$\quad \mathsf{return}\ pos$

$rndfile\text{-}set\text{-}file : Table\text{-}id \times FileRnd \xrightarrow{o} ()$

$rndfile\text{-}set\text{-}file\,(tid, file) \triangleq$
$\quad rndfile\text{-}files\,(tid) := file$

$rndfile\text{-}get\text{-}file : Table\text{-}id \xrightarrow{o} FileRnd$

$rndfile\text{-}get\text{-}file\,(tid) \triangleq$
$\quad \mathsf{def}\ file = rndfile\text{-}files\,(tid)\ \mathsf{in}$
$\quad \mathsf{return}\ file$

$rndfile\text{-}set\text{-}name : Table\text{-}id \times Char^* \xrightarrow{o} ()$

$rndfile\text{-}set\text{-}name\,(tid, name) \triangleq$
$\quad rndfile\text{-}names\,(tid) := name$

$rndfile\text{-}get\text{-}name : Table\text{-}id \xrightarrow{o} Char^*$

$rndfile\text{-}get\text{-}name\,(tid) \triangleq$
$\quad \mathsf{def}\ name = rndfile\text{-}names\,(tid)\ \mathsf{in}$
$\quad \mathsf{return}\ name$

$rndfile\text{-}get\text{-}realprops : Table\text{-}id \xrightarrow{o} Set\text{-}value$

$rndfile\text{-}get\text{-}realprops\,(tid) \triangleq$
$\quad \mathsf{def}\ props = rndfile\text{-}realprops\,(tid)\ \mathsf{in}$
$\quad \mathsf{return}\ props$

9.2.5.5 The Module TermFile

The module `TermFile` provides facilities that allow elementary access to an interactive terminal.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9.2.5.5.1 The Definition Module of TermFile

```
DEFINITION MODULE TermFile;

  (* Access to the terminal device *)

  (* Channels opened by this module are connected to a single terminal device; typed
     characters are distributed between channels according to the sequence of read requests.
  *)

IMPORT IOChan, ChanConsts;

TYPE
  ChanId = IOChan.ChanId;
  FlagSet = ChanConsts.FlagSet;
  OpenResults = ChanConsts.OpenResults;

  (* Accepted singleton values of FlagSet *)

CONST
  read = FlagSet{ChanConsts.readFlag};   (* input operations are requested/available *)
  write = FlagSet{ChanConsts.writeFlag}; (* output operations are requested/available *)
  text = FlagSet{ChanConsts.textFlag};   (* text operations are requested/available *)
  raw = FlagSet{ChanConsts.rawFlag};     (* raw operations are requested/available *)
  echo = FlagSet{ChanConsts.echoFlag};   (* echoing by interactive device on reading of
                                            characters from input stream requested/applies
                                         *)

PROCEDURE Open (VAR cid: ChanId; flags: FlagSet; VAR res: OpenResults);
  (* Attempts to obtain and open a channel connected to the terminal.
     Without the raw flag, text is implied.
     Without the echo flag, line mode is requested, otherwise single character mode is
     requested.
     If successful, assigns to cid the identity of the opened channel, and assigns the value
     opened to res.
     If a channel cannot be opened as required, the value of res indicates the reason, and
     cid identifies the invalid channel.
  *)

PROCEDURE IsTermFile (cid: ChanId): BOOLEAN;
  (* Tests if the channel identified by cid is open to the terminal. *)

PROCEDURE Close (VAR cid: ChanId);
  (* If the channel identified by cid is not open to the terminal, the exception wrongDevice
     is raised; otherwise closes the channel and assigns the value identifying the invalid
     channel to cid.
  *)

END TermFile.
```

9.2.5.5.2 The Procedures of TermFile

Channels connected to the terminal device shall be opened in line mode or in single-character mode. In line mode, items shall be echoed before being added to the input stream and shall be added a line at a time.
In single character mode, items shall be added to the input stream as they are typed, and shall be echoed as they are removed from the input stream by a text read device operation — provided they have not already been echoed.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Typed characters shall be distributed between multiple channels according to the sequence of read requests.

NOTE — If all the channels open to the terminal are open in line mode, the terminal device operates exclusively in line mode; in that case, echoing might be performed by an underlying operating system. Similarly, if all the channels open to the terminal are open in single-character mode, the terminal device operates exclusively in single-character mode; in that case, echoing only occurs on reading from a channel and not on looking or skipping: this allows interactive input routines to suppress the echoing of unwanted or unexpected characters.

If an implementation allows it, there might be one or more channels open in line mode, and one or more channels open in single-character mode. In that case, echoing is postponed until the treatment of characters can be determined according to the sequence of calls of input operations. This behaviour allows programs that use the terminal in different modes to be written in a modular fashion, there being no need explicitly to save and restore the state of the terminal device.

In a request to open a channel to the terminal device, the flags `read`, `write`, `text`, `raw`, and `echo` shall apply. If `raw` is not included in the request parameter `flags`, inclusion of `text` shall be implied. If the `read` flag is not included in the request, inclusion of the `write` flag shall be implied.

values

$termfile\text{-}flags = \{readFlag, writeFlag, textFlag, rawFlag, echoFlag\}$

$termfile\text{-}merge\text{-}flags : Set\text{-}value \rightarrow [Set\text{-}value]$

$termfile\text{-}merge\text{-}flags\,(given) \triangleq$
  let $assumed = \{interactiveFlag\}$ in
  let $default1 =$ if $rawFlag \in given$ then $\{\}$ else $\{textFlag\}$ in
  let $default2 =$ if $readFlag \in given$ then $\{\}$ else $\{writeFlag\}$ in
  if $given \subseteq seqfile\text{-}flags$
  then $given \cup assumed \cup default1 \cup default2$
  else nil

$termfile\text{-}check\text{-}file : Set\text{-}value \rightarrow Exception\text{-}handler \xrightarrow{o} Enumeration\text{-}value$

$termfile\text{-}check\text{-}file\,(flags)handler \triangleq$
  implementation-defined

annotations Returns *opened* if the requested read/write/text/raw operations are available on the terminal; otherwise returns *tooManyOpen*, *wrongPermissions*, *noRawOperations*, *noMixedOperations*, *alreadyOpen*, or *otherProblem*.

The procedure Open

If successful, the call `Open(cid,flags,res)` shall assign to `cid` the identity of a channel that is connected to the terminal device.

If the `echo` flag is included in the request, single-character mode shall be available if the call is to succeed and the channel shall operate in single-character mode. Without the `echo` flag, line mode shall be available if the call is to succeed and the channel shall operate in line mode.

If a channel cannot be opened as required, the value of `res` shall indicate the reason, and `cid` shall identify the invalid channel.

$termfile\text{-}Open : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} ()$

$termfile\text{-}Open\,(args)handler \triangleq$
  let $[vcid, flags, vres] = args$ in
  let $xflags = termfile\text{-}merge\text{-}flags\,(flags)$ in 513
  if $xflags =$ nil
  then $(assign(vcid, badchan\,())\ handler;$ 150 538
    $assign(vres, wrongFlags)\ handler$ 150
)
  else def $res = termfile\text{-}check(xflags)\ handler$ in 514

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
    if res ≠ opened
    then (assign(vcid, badchan ()) handler;                                    150 538
          assign(vres, res) handler                                            150
          )
    else def mk-(cid, tid) = make-a-channel(TERMFILE, xflags, termfile-ops) in 540
         if cid = badchan ()                                                   538
         then (assign(vcid, badchan ()) handler;                               150 538
               assign(vres, outOfChannels) handler                             150
               )
         else (if writeFlag ∈ xflags then tf-inc-opened() else skip;           521
               assign(vcid, cid) handler;                                      150
               assign(vres, opened) handler                                    150
               )
```

```
termfile-check : Set-value → Exception-handler →o Enumeration-value
termfile-check (flags)handler △
  def res = if echoFlag ∈ flags
            then def cs = can-do-single-chars() in                             514
                 if ¬ cs then otherProblem else termfile-check-file(flags) handler 513
            else def cl = can-do-lines() in                                    514
                 if ¬ cl then otherProblem else termfile-check-file(flags) handler in 513
  return res
```

Auxiliaries

```
can-do-single-chars : () →o 𝔹
can-do-single-chars () △
   implementation-defined value
```

annotations Returns true if a new channel can currently be opened in single-character mode; otherwise returns false.

```
can-do-lines : () →o 𝔹
can-do-lines () △
   implementation-defined value
```

annotations Returns true if a new channel can currently be opened in line mode; otherwise returns false.

The procedure `IsTermFile`

The value of the call `IsTermFile(cid)` shall be *true* if the channel identified by `cid` is open to the terminal device, and *false* otherwise.

```
IsTermFile : Arguments → Exception-handler →o 𝔹
IsTermFile (args)handler △
  let [cid] = args in
  def tid = table-id(cid) in                                                   539
  def did = get-device-id(tid) in                                              540
  return did = TERMFILE
```

The procedure `Close`

If the channel identified by `cid` is open to the terminal device, the call `Close(cid)` shall close the channel and assign the value identifying the invalid channel to `cid`; otherwise, the exception `wrongDevice` shall be raised.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
termfile-Close : Arguments → Exception-handler ⟶° ()
termfile-Close (args)handler ≙
  let [cid] = args in
  def tid = table-id(cid) in                                   539
  (vet(tid, TERMFILE) handler;                                 529
   termfile-free([tid]) handler;                               519
   release-a-channel(cid, tid) ;                               540
   assign(cid, badchan ()) handler                             150 538
  )
```

9.2.5.5.3 The Device Procedures of TermFile

If supported on a channel open to the `TermFile` device, the `TermFile` device procedures shall have the semantics defined below; otherwise, they shall have the semantics of the corresponding device procedure for the invalid channel.

NOTE — Device procedures, other than *Free*, are invoked through corresponding procedures from `IOChan`.

The `TermFile` device procedure *Look*

```
termfile-look : Arguments → Exception-handler ⟶° ()
termfile-look (args)handler ≙
  let [tid, vch, vres] = args in
  (check-flags(tid, {readFlag}) handler;                       529
   def seen = tf-inspect-queue(tid) handler in                 520
   if seen = nil
   then tf-add-queue(tid) handler                              520
   else skip;
   def item = tf-inspect-queue(tid) handler in                 520
   def mk-(ch, res) = if item = nil
                      then mk-(nil , endOfInput)
                      else if item = END-LINE
                           then mk-(nil , endOfLine)
                           else mk-(item, allRight) in
   (set-rdresult(tid, res) ;                                   539
    assign(vres, res) handler;                                 150
    if ch = nil
    then let impdep : (Char | UNDEFINED) in
         assign(vch, impdep) handler                           150
    else assign(vch, ch) handler                               150
    )
  )
```

The `TermFile` device procedure *Skip*

```
termfile-skip : Arguments → Exception-handler ⟶° ()
termfile-skip (args)handler ≙
  let [tid] = args in
  (check-flags(tid, {readFlag}) handler;                       529
   def seen = tf-inspect-queue(tid) handler in                 520
   if seen = nil then tf-add-queue(tid) handler                520
   else skip;
   def toskip = tf-inspect-queue(tid) handler in               520
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    if toskip ≠ nil
    then (tf-skip-queue(len emit (tid, toskip)) ;                    520
          set-rdresult(tid, allRight)                                539
          )
    else chan-exception(skipAtEnd) handler                           470
    )
```

The TermFile device procedure *SkipLook*

```
termfile-skiplook : Arguments → Exception-handler -o→ ()
termfile-skiplook (args)handler ≙
  let [tid, vch, vres] = args in
  (termfile-skip([tid]) handler;                                     515
   termfile-look(args) handler                                       515
   )
```

The TermFile device procedure *WriteLn*

```
termfile-writeln : Arguments → Exception-handler -o→ ()
termfile-writeln (args)handler ≙
  let [tid] = args in
  (check-flags(tid, {writeFlag}) handler;                            529
   let output = emit (tid, END-LINE) in                              529
   term-put(output) handler                                          522
   )
```

The TermFile device procedure *TextRead*

```
termfile-textread : Arguments → Exception-handler -o→ ()
termfile-textread (args)handler ≙
  let [tid, ato, maxChars, vcharsRead] = args in
  (check-flags(tid, {readFlag}) handler;                             529
   if maxChars = 0
   then (set-rdresult(tid, allRight) ;                               539
         assign(vcharsRead, 0) handler                               150
         )
   else (def seen = tf-inspect-queue(tid) handler in                 520
         if seen = nil then tf-add-queue(tid) handler                520
         else skip;
         def item = tf-inspect-queue(tid) handler in                 520
         if item = nil
         then (set-rdresult(tid, endOfInput) ;                       539
               assign(vcharsRead, 0) handler                         150
               )
         elseif item = END-LINE
         then (set-rdresult(tid, endOfLine) ;                        539
               assign(vcharsRead, 0) handler                         150
               )
         else def mk-(ans, nchars) = tf-read-chars-up-to(tid, maxChars) handler in   520
              def mk-(vto, cap) = make-subarray(ato, maxChars) in    529
              let safechars = if nchars ≤ cap
                              then nchars else cap in
              (let chars = ans (1, ..., safechars) in
               for all i ∈ inds chars
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
            do assign(vto (i − 1), chars (i)) handler ;                                  150
            set-rdresult(tid, allRight) ;                                                539
            assign(vcharsRead, safechars) handler;                                       150
            if safechars < nchars
            then non-mandatory-detection(NONEXISTENT) handler                            397
            else skip
            )
        )
    )
```

The TermFile device procedure *TextWrite*

```
termfile-textwrite : Arguments → Exception-handler →o ()
termfile-textwrite (args)handler ≙
  let [tid, ato, maxChars] = args in
  def mk- (vfrom, cap) = make-subarray(ato, maxChars) in                                 529
  (check-flags(tid, {writeFlag}) handler;                                                529
   let safechars = if maxChars ≤ cap then maxChars else cap in
   (if safechars > 0
    then let frmseq = map-to-seq (vfrom) in                                              615
         let outs = frmseq (1, ..., safechars) in
         let bos = emit (tid, outs) in                                                   529
         term-put(bos) handler                                                           522
    else skip;
    if safechars < maxChars
    then non-mandatory-detection(NONEXISTENT) handler                                    397
    else skip
   )
  )
```

The TermFile device procedure *RawRead*

```
termfile-rawread : Arguments → Exception-handler →o ()
termfile-rawread (args)handler ≙
  let [tid, ato, maxLocs, vlocsRead] = args in
  (check-flags(tid, {readFlag}) handler;                                                 529
   if maxLocs = 0
   then (set-rdresult(tid, allRight) ;                                                   539
         assign(vlocsRead, 0) handler                                                    150
         )
   else (def queued = tf-have-queue() in                                                 521
         if ¬ queued then tf-add-queue(tid) handler                                      520
         else skip;
         def queued = tf-have-queue() in                                                 521
         if ¬ queued
         then (set-rdresult(tid, endOfInput) ;                                           539
               assign(vlocsRead, 0) handler                                              150
               )
         else def ans = tf-read-up-to(tid, maxLocs) handler in                           521
              def mk- (vto, cap) = make-subarray(ato, maxLocs) in                        529
              let safelocs = if len ans ≤ cap then len ans else cap in
              (let locs = ans (1, ..., safelocs) in
               for all i ∈ inds locs
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
            do assign(vto (i − 1), locs (i)) handler ;                                    150
            set-rdresult(tid, allRight) ;                                                 539
            assign(vlocsRead, safelocs) handler;                                          150
            if safelocs < len ans
            then non-mandatory-detection(NONEXISTENT) handler                             397
            else skip
            )
        )
    )
```

The TermFile device procedure *RawWrite*

```
termfile-rawwrite : Arguments → Exception-handler →o ()
termfile-rawwrite (args)handler △
  let [tid, ato, maxLocs] = args in
  def mk- (vfrom, cap) = make-subarray(ato, maxLocs) in                                   529
  (check-flags(tid, {writeFlag}) handler;                                                 529
   let safelocs = if maxLocs ≤ cap then maxLocs else cap in
   (if safelocs > 0
    then let frmseq = map-to-seq (vfrom) in                                               615
         let outs = frmseq (1, ..., safelocs) in
         term-put(outs) handler                                                           522
    else skip;
    if safelocs < maxLocs
    then non-mandatory-detection(NONEXISTENT) handler                                     397
    else skip
   )
  )
```

The TermFile device procedure *GetName*

```
termfile-getname : Arguments → Exception-handler →o ()
termfile-getname (args)handler △
  let [tid, vs] = args in
  def name : Char* = implementation-defined value in
  assign(vs, name) handler                                                                150
```

The TermFile device procedure *Reset*

```
termfile-reset : Arguments → Exception-handler →o ()
termfile-reset (args)handler △
  tf-clear-queue()                                                                        521
```

The TermFile device procedure *Flush*

```
termfile-flush : Arguments → Exception-handler →o ()
termfile-flush (args)handler △
  let [tid] = args in
  def props = get-props(tid) in                                                           539
  if writeFlag ∈ props
  then term-flush() handler                                                               522
  else skip
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The TermFile device procedure *Free*

```
termfile-free : Arguments → Exception-handler →o ()
termfile-free (args)handler △
  let [tid] = args in
  def props = get-props(tid) in                                    539
  if writeFlag ∈ props
  then (tf-dec-opened() ;                                          521
        def n = tf-get-opened() in                                 521
        if n = 0
        then termfile-flush([tid]) handler                         518
        else skip
        )
  else skip
```

values

```
  termfile-ops =                          termfile-dens =
  {LOOK ↦ termfile-look-pid,              {termfile-look-pid ↦ make-proper-procden (termfile-look),          339
  SKIP ↦ termfile-skip-pid,               termfile-skip-pid ↦ make-proper-procden (termfile-skip),           339
  SKIPLOOK ↦ termfile-skiplook-pid,       termfile-skiplook-pid ↦ make-proper-procden (termfile-skiplook),   339
  WRITELN ↦ termfile-writeln-pid,         termfile-writeln-pid ↦ make-proper-procden (termfile-writeln),     339
  READTEXT ↦ termfile-readtext-pid,       termfile-readtext-pid ↦ make-proper-procden (termfile-readtext),   339
  WRITETEXT ↦ termfile-writetext-pid,     termfile-writetext-pid ↦ make-proper-procden (termfile-writetext), 339
  READBIN ↦ termfile-readraw-pid,         termfile-readraw-pid ↦ make-proper-procden (termfile-readraw),     339
  WRITEBIN ↦ termfile-writeraw-pid,       termfile-writeraw-pid ↦ make-proper-procden (termfile-writeraw),   339
  FLUSH ↦ termfile-flush-pid,             termfile-flush-pid ↦ make-proper-procden (termfile-flush),         339
  RESET ↦ termfile-reset-pid,             termfile-reset-pid ↦ make-proper-procden (termfile-reset),         339
  FREE ↦ termfile-free-pid}               termfile-free-pid ↦ make-proper-procden (termfile-free)}           339
```

annotations The mapping *termfile-dens*, from elements of type *Procedure-id* to the corresponding procedure denotations, is part of the mapping *dens* used in the construction of the abstract implementation module — see Annex B.

9.2.5.5.4 The State for TermFile

```
state TermFile of
  termfile-not-echoed-queue : Elementary-loc-value*
  termfile-echoed-queue     : Elementary-loc-value*
  termfile-seeneof          : 𝔹
  termfile-opened           : ℕ

  inv mk-TermFile (nqueue, equeue, -, -) △
      ¬ (len nqueue > 0 ∧ len equeue > 0)

  init mk-TermFile (nqueue, equeue, ateof, opened) △
       nqueue = [] ∧ equeue = [] ∧ seeneof = false ∧ opened = 0
end
```

annotations *queue* corresponds to the sequence of text items obtained from the underlying system but not consumed by read or skip operations.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Auxiliaries

Access to the state for TermFile

tf-inspect-queue : *Table-id* → *Exception-handler* $\xrightarrow{o}$ [*Char* | END-LINE]

tf-inspect-queue (*tid*)*handler* $\triangleq$
  def *res* = def *nq* = *termfile-nqueue* in
    if *nq* = []
    then def *eq* = *termfile-equeue* in
      if *eq* = []
      then nil
      else *a-text-item*(*tid*, *eq*) *handler* 529
    else *a-text-item*(*tid*, *nq*) *handler* in 529
  return *res*

annotations Return the text item at the front of the queue or nil if the queue is empty.

tf-add-queue : *Table-id* → *Exception-handler* $\xrightarrow{o}$ ()

tf-add-queue (*tid*)*handler* $\triangleq$
  def *input* = *tf-getinput*() *handler* in 520
  def *props* = *get-props*(*tid*) in 539
  if *echo* $\notin$ *props*
  then (def *eq* = *termfile-equeue* in
    termfile-equeue := *eq* ↷ *input*;
    term-put(*input*) *handler*; 522
    if *input* ≠ END-LINE ∧ *input* ≠ []
    then *tf-add-queue*(*tid*) *handler* 520
)
  else *termfile-nqueue* := *input*

tf-getinput : () → *Exception-handler* $\xrightarrow{o}$ [*Elementary-loc-value**]

tf-getinput ()*handler* $\triangleq$
  def *donottry* = *termfile-seeneof* in
  def *res* = if *donottry*
    then []
    else *term-get*() *handler* in 522
  (if *res* = []
   then *termfile-seeneof* := true;
   return *res*
)

tf-skip-queue : N $\xrightarrow{o}$ ()

tf-skip-queue (*n*) $\triangleq$
  def *nq* = *termfile-nqueue* in
  def *eq* = *termfile-equeue* in
  if *nq* = []
  then *termfile-equeue* := *eq* (*n* + 1, . . . , len *eq*)
  else *termfile-nqueue* := []

tf-read-chars-up-to : *Table-id* × N → *Exception-handler* $\xrightarrow{o}$ (*Char** × N)

tf-read-chars-up-to (*tid*, *n*)*handler* $\triangleq$
  def *nq* = *termfile-nqueue* in
  def *eq* = *termfile-equeue* in
  if *nq* = []
  then def *mk-*(*chars*, *nchars*) = *chars-up-to*(*tid*, *eq*, *n*) *handler* in 530

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
      (let amount = len emit(tid, chars) in
       termfile-equeue := eq(amount + 1, ..., len eq);
       return mk-(chars, nchars)
       )
  else def char = a-text-item(tid, nq) handler in                    529
      (term-put(nq) handler;                                         522
       termfile-nqueue := [];
       return mk-([char], 1)
       )

tf-read-up-to : Table-id × N → Exception-handler ~o~> Elementary-loc-value*
tf-read-up-to(tid, n)handler ≜
  def nq = termfile-nqueue in
  def eq = termfile-equeue in
  if nq = []
  then let locs = up-to(eq, n) in                                    530
      (let amount = len locs in
       termfile-equeue := eq(amount + 1, ..., len eq);
       return locs
       )
  else (term-put(nq) handler;                                        522
       termfile-nqueue := [];
       return nq
       )

tf-have-queue : () ~o~> B
tf-have-queue() ≜
  def res = termfile-nqueue ≠ [] ∨ termfile-equeue ≠ [] in
  return res

tf-clear-queue : () ~o~> ()
tf-clear-queue() ≜
  (termfile-nqueue := [];
   termfile-equeue := [];
   termfile-seeneof := false
   )

tf-inc-opened : () ~o~> ()
tf-inc-opened() ≜
  def n = termfile-opened in
  termfile-opened := n + 1

tf-dec-opened : () ~o~> ()
tf-dec-opened() ≜
  def n = termfile-opened in
  termfile-opened := n-1

tf-get-opened : () ~o~> N
tf-get-opened() ≜
  def n = termfile-opened in
  return n
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Access to the external state for TermFile

term-get : () → *Exception-handler* $\xrightarrow{o}$ *Elementary-loc-value**

term-get ()*handler* $\triangleq$
implementation-dependent actions

annotations Obtains the external representation of the next character, removing it from the external input stream, or returns the empty sequence if there are no more characters available before an end of input indication is made by the user.

term-put : *Elementary-loc-value** → *Exception-handler* $\xrightarrow{o}$ ()

term-put (*output*)*handler* $\triangleq$
implementation-dependent actions

annotations Appends the character or characters represented externally by *output* to the external output stream.

term-flush : () → *Exception-handler* $\xrightarrow{o}$ ()

term-flush ()*handler* $\triangleq$
implementation-dependent actions

annotations Flushes any characters buffered by the underlying implementation out to the external destination.

9.2.5.6 The Module ProgramArgs

The module ProgramArgs provides a channel from which input can be taken from any arguments given to the program.

9.2.5.6.1 The Definition Module of ProgramArgs

```
DEFINITION MODULE ProgramArgs;

  (* Access to program arguments *)

IMPORT IOChan;

TYPE
  ChanId = IOChan.ChanId;

PROCEDURE ArgChan (): ChanId;
  (* Returns a value that identifies a channel for reading program arguments *)

PROCEDURE IsArgPresent (): BOOLEAN;
  (* Tests if there is a current argument to read from. If not,
     read <= IOChan.CurrentFlags() will be FALSE, and attempting to read from the argument
     channel will raise the exception notAvailable.
  *)

PROCEDURE NextArg ();
  (* If there is another argument, causes subsequent input from the argument device to come
     from the start of the next argument.  Otherwise there is no argument to read from, and
     a call of IsArgPresent will return FALSE.
  *)

END ProgramArgs.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9.2.5.6.2 The Procedures of ProgramArgs

The procedure `ArgChan`

The value of the call `ArgChan()` shall be a value identifying a channel from which the implementation-defined program arguments may be read.

ArgChan : () → *Exception-handler* $\xrightarrow{o}$ *Chan-id*
ArgChan ()*handler* $\triangleq$
  def *cid* = *pa-chan-id*() in 528
  return *cid*

The procedure `IsArgPresent`

The value of the call `IsArgPresent()` shall be *true* if there is a current argument from which to read, and *false* otherwise.

If there is no current argument, '`read <= IOChan.Flags()`' shall be *false*, and attempting to read from the argument channel shall raise the exception `notAvailable`.

IsArgPresent : () → *Exception-handler* $\xrightarrow{o}$ $\mathbb{B}$
IsArgPresent ()*handler* $\triangleq$
  def *res* = *pa-is-present*() in 527
  return *res*

The procedure `NextArg`

After the call `NextArg()`, if there is another argument, subsequent input from the argument channel shall be taken from the start of that argument; otherwise a call of `IsArgPresent()` shall return *false*.

NOTE — Provision of `NextArg` allows the treatment of arguments that contain spaces or line marks.

NextArg : () → *Exception-handler* $\xrightarrow{o}$ ()
NextArg ()*handler* $\triangleq$
  pa-skip-between() 528

9.2.5.6.3 The Device Procedures of ProgramArgs

If supported, the `ProgramArgs` device procedures shall have the semantics defined below; otherwise, they shall have the semantics of the corresponding device procedure for the invalid channel.

NOTE — Device procedures, other than *Free*, are invoked through corresponding procedures from `IOChan`.

The `ProgramArgs` device procedure *Look*

programargs-look : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()
programargs-look (*args*)*handler* $\triangleq$
  let [*tid*, *vch*, *vres*] = *args* in
  (*check-flags*(*tid*, {*readFlag*}) *handler*; 529
   def *item* = *pa-inspect-arg*(*tid*) *handler* in 526

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  def mk-(ch, res) = if item = nil
                     then mk-(nil , endOfInput)
                     else if item = END-LINE
                          then mk-(nil , endOfLine)
                          else mk-(item, allRight) in
  (set-rdresult(tid, res) ;                                          539
   assign(vres, res) handler;                                        150
   if ch = nil
   then let impdep : (Char | UNDEFINED) in
        assign(vch, impdep) handler                                  150
   else assign(vch, ch) handler                                      150
   )
  )
```

The ProgramArgs device procedure *Skip*

```
programargs-skip : Arguments → Exception-handler →o ()
programargs-skip (args)handler △
  let [tid] = args in
  (check-flags(tid, {readFlag}) handler;                             529
   def toskip = pa-inspect-arg(tid) handler in                       526
   if toskip ≠ nil
   then (pa-skip-within(len emit (tid, toskip)) ;                    528
         set-rdresult(tid, allRight)                                 539
         )
   else chan-exception(skipAtEnd) handler                            470
   )
```

The ProgramArgs device procedure *SkipLook*

```
programargs-skiplook : Arguments → Exception-handler →o ()
programargs-skiplook (args)handler △
  let [tid, vch, vres] = args in
  (programargs-skip([tid]) handler;                                  524
   programargs-look(args) handler                                    523
   )
```

The ProgramArgs device procedure *TextRead*

```
programargs-textread : Arguments → Exception-handler →o ()
programargs-textread (args)handler △
  let [tid, ato, maxChars, vcharsRead] = args in
  (check-flags(tid, {readFlag}) handler;                             529
   if maxChars = 0
   then (set-rdresult(tid, allRight) ;                               539
         assign(vcharsRead, 0) handler                               150
         )
   else def item = pa-inspect-arg(tid) handler in                    526
        if item = nil
        then (set-rdresult(tid, endOfInput) ;                        539
              assign(vcharsRead, 0) handler                          150
              )
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      elseif item = END-LINE
      then (set-rdresult(tid, endOfLine) ;                                    539
            assign(vcharsRead, 0) handler                                     150
            )
      else def mk-(ans, nchars) = pa-read-chars-up-to(tid, maxChars) handler in   526
           def mk-(vto, cap) = make-subarray(ato, maxChars) in                529
           let safechars = if nchars ≤ cap then nchars else cap in
           (let chars = ans(1, ..., safechars) in
            for all i ∈ inds chars
            do assign(vto(i − 1), chars(i)) handler ;                         150
            set-rdresult(tid, allRight) ;                                     539
            assign(vcharsRead, safechars) handler;                            150
            if safechars < nchars
            then non-mandatory-detection(NONEXISTENT) handler                 397
            else skip
      )
  )
```

The ProgramArgs device procedure *RawRead*

```
programargs-rawread : Arguments → Exception-handler →o ()
programargs-rawread (args)handler ≜
  let [tid, ato, maxLocs, vlocsRead] = args in
  (check-flags(tid, {readFlag}) handler;                                      529
   if maxLocs = 0
   then (set-rdresult(tid, allRight) ;                                        539
         assign(vlocsRead, 0) handler                                         150
         )
   else def rest = pa-arg-left() in                                           527
        if rest = []
        then (set-rdresult(tid, endOfInput) ;                                 539
              assign(vlocsRead, 0) handler                                    150
              )
        else let ans = up-to(rest, maxLocs) in                                530
             (let amount = len ans in
              pa-skip-within(amount) ;                                        528
              def mk-(vto, cap) = make-subarray(ato, maxLocs) in              529
              let safelocs = if len ans ≤ cap then len ans else cap in
              (let locs = ans(1, ..., safelocs) in
               for all i ∈ inds locs
               do assign(vto(i − 1), locs(i)) handler ;                       150
               set-rdresult(tid, allRight) ;                                  539
               assign(vlocsRead, safelocs) handler;                           150
               if safelocs < len ans
               then non-mandatory-detection(NONEXISTENT) handler              397
               else skip
               )
              )
  )
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The ProgramArgs device procedure *GetName*

```
programargs-getname : Arguments → Exception-handler ~o~> ()
programargs-getname (args)handler △
  let [tid, vs] = args in
  def name : Char* = implementation-defined value in
  assign(vs, name) handler                                          150
```

The ProgramArgs device procedure *Reset*

```
programargs-reset : Arguments → Exception-handler ~o~> ()
programargs-reset (args)handler △
  let [tid] = args in
  pa-reset-args(tid)                                                528
```

values

```
  programargs-ops =
  {LOOK ↦ programargs-look-pid,
  SKIP ↦ programargs-skip-pid,
  SKIPLOOK ↦ programargs-skiplook-pid,
  WRITELN ↦ notavail-proc-pid,
  READTEXT ↦ programargs-readtext-pid,
  WRITETEXT ↦ notavail-proc-pid,
  READBIN ↦ programargs-readraw-pid,
  WRITEBIN ↦ notavail-proc-pid,
  FLUSH ↦ notavail-proc-pid,
  RESET ↦ programargs-reset-pid,
  FREE ↦ null-proc-pid}

  programargs-dens =
{programargs-look-pid ↦ make-proper-procden (programargs-look),      339
programargs-skip-pid ↦ make-proper-procden (programargs-skip),       339
programargs-skiplook-pid ↦ make-proper-procden (programargs-skiplook),339
programargs-readtext-pid ↦ make-proper-procden (programargs-readtext),339
programargs-readraw-pid ↦ make-proper-procden (programargs-readraw),339
programargs-reset-pid ↦ make-proper-procden (programargs-reset)}      339
```

annotations The mapping *programargs-dens*, from elements of type *Procedure-id* to the corresponding procedure denotations, is part of the mapping *dens* used in the construction of the abstract implementation module — see Annex B.

Auxiliaries

```
pa-inspect-arg : Table-id → Exception-handler ~o~> [Char | END-LINE]
pa-inspect-arg ()handler △
  def res = def rest = pa-arg-left() in                             527
            if rest = []
            then nil
            else a-text-item(tid, rest) handler in                  529
  return res

pa-read-chars-up-to : Table-id × ℕ → Exception-handler ~o~> (Char* × ℕ)
pa-read-chars-up-to (tid, n)handler △
  def rest = pa-arg-left() in                                       527
  def mk-(chars, nchars) = chars-up-to(tid, rest, n) handler in     530
  (let amount = len emit (tid, chars) in
   pa-skip-within(amount) ;                                         528
   return mk-(chars, nchars)
   )
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9.2.5.6.4 The Initialization of ProgramArgs

The initialization of the module `ProgramArgs`, shall open the channel from which the implementation-defined program arguments may be read.

```
programargs-initialization : () → Exception-handler ^o→ ()
programargs-initialization ()handler △
  def paflags = pa-init-props() in                                              527
  def mk-(cid, tid) = make-a-channel(PROGRAMARGS, paflags, programargs-ops) in   540
  (programargs-chanid := cid;
   programargs-tableid := tid
  )

pa-init-props : () ^o→ Set-value
pa-init-props () △
  def present = pa-is-present() in                                              527
  let props = if present
              then {readFlag, textFlag, rawFlag}
              else {textFlag, rawFlag} in
  return props
```

9.2.5.6.5 The State for ProgramArgs

```
types

  Program-argument = Elementary-loc-value*;

  External-ProgramArgs-state :: argumentlist : Program-argument*

state ProgramArgs of
  programargs-arguments : Program-argument*
  programargs-within    : ℕ
  programargs-between   : ℕ
  programargs-chanid    : Chan-id
  programargs-tableid   : Table-id
  init mk-ProgramArgs (within, between) △
      within = 1 ∧
      between = 1
end
```

Auxiliaries

Access to the state for ProgramArgs

```
pa-is-present : () ^o→ 𝔹
pa-is-present () △
  def between = programargs-between in
  def arguments = programargs-arguments in
  return between ≤ len arguments

pa-arg-left : () ^o→ Elementary-loc-value*
pa-arg-left () △
  def within = programargs-within in
  def between = programargs-between in
  def arguments = programargs-arguments in
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  let arg = arguments(between) in
  return arg(within, ..., len arg)
```

```
pa-skip-within : N →o ()
pa-skip-within (n) ≙
  def within = programargs-within in
  programargs-within := within + n
```

```
pa-skip-between : () →o ()
pa-skip-between () ≙
  def between = programargs-between in
  def arguments = programargs-arguments in
  if between < len arguments
  then (programargs-between := between + 1;
        programargs-within := 1
        )
  else (def tid = programargs-tableid in
        set-props(tid, {textFlag, rawFlag}) ;                    539
        programargs-between := len arguments + 1
        )
```

```
pa-reset-args : Table-id →o ()
pa-reset-args (tid) ≙
  (programargs-between := 1;
   programargs-within := 1;
   def arguments = programargs-arguments in
   if len arguments ≥ 1
   then set-props(tid, {readFlag, textFlag, rawFlag})            539
   else skip
   )
```

```
pa-chan-id : () →o Chan-id
pa-chan-id () ≙
  def cid = programargs-chanid in
  return cid
```

Access to the external state for ProgramArgs

```
read-external-ProgramArgs-state : External-ProgramArgs-state →o ()
read-external-ProgramArgs-state (es) ≙
  programargs-arguments := es
```

annotations Install the argument list in the state for ProgramArgs

```
write-external-ProgramArgs-state : () →o External-ProgramArgs-state
write-external-ProgramArgs-state () ≙
  return programargs-arguments
```

annotations Return the (unaltered) argument list.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9.2.5.7 Implementation-defined Functions used in the Device Modules

$isa\text{-}Char : Table\text{-}id \times Elementary\text{-}loc\text{-}value^* \to \mathbb{B}$

$isa\text{-}Char\,(tid, s) \triangleq$
implementation-defined

annotations Return whether an external representation of data is representable as a character.

$emit : Table\text{-}id \times (Char^* \mid \text{END-LINE}) \to Elementary\text{-}loc\text{-}value^*$

$emit\,(tid, its) \triangleq$
implementation-defined

annotations Return the external representation of a sequence of characters or a line mark

$is\text{-}legal\text{-}filename : Char^* \to \mathbb{B}$

$is\text{-}legal\text{-}filename\,(name) \triangleq$
implementation-defined

$is\text{-}existing\text{-}file : Char^* \to \mathbb{B}$

$is\text{-}existing\text{-}file\,(name) \triangleq$
implementation-defined
pre $is\text{-}legal\text{-}filename\,(name)$ 529

$make\text{-}subarray : Pointer\text{-}value \times \mathbb{N} \xrightarrow{o} (Array\text{-}variable \times \mathbb{N})$

$make\text{-}subarray\ pointer, subcap \triangleq$
implementation-defined actions

annotations An implementation-defined operation that takes a pointer to a component of an array variable, *pointer*, and the required capacity of a sub-array, *cap*, with that component as its first component, and returns an array variable with component type given by the bound type of the pointer and capacity given by the maximum of cap and the number of remaining components in the array.

9.2.5.8 Auxiliary Formal Model Definitions

$check\text{-}flags : Table\text{-}id \times Set\text{-}value \to Exception\text{-}handler \xrightarrow{o} ()$

$check\text{-}flags\,(tid, props) handler \triangleq$
def $dtbl = devtables\,(tid)$ in
let $mk\text{-}Device\text{-}table\,(\text{-}, properties, \text{-}, \text{-}, \text{-}) = dtbl$ in
if $props \subseteq properties$
then skip
else $chan\text{-}exception(notAvailable)\ handler$ 470

$vet : Table\text{-}id \times Dev\text{-}id \times Set\text{-}value \xrightarrow{o} ()$

$vet\,(tid, did, props) \triangleq$
def $dtbl = devtables\,(tid)$ in
let $mk\text{-}Device\text{-}table\,(devid, \text{-}, \text{-}, \text{-}, \text{-}) = dtbl$ in
if $did \neq devid$
then $chan\text{-}exception(wrongDevice)\ handler$ 470
else skip

annotations Check that the driver has created the channel

$a\text{-}text\text{-}item : Table\text{-}id \times Elementary\text{-}loc\text{-}value^* \to Exception\text{-}handler \xrightarrow{o} Char \mid \text{END-LINE}$

$a\text{-}text\text{-}item\,(tid, s) handler \triangleq$
let $possible\text{-}items = \{front\,(s,i) \mid i \in \mathbb{N} \cdot isa\text{-}Char\,(tid, front\,(s,i)) \vee front\,(s,i) = \text{END-LINE}\}$ in 614 529 614
if $possible\text{-}items = \{\,\}$
then $chan\text{-}exception(textParseError)\ handler$ 470

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

else let *choice* : *Char* | END-LINE be st $\{choice\} \subseteq possible\text{-}items$ in
return *choice*

annotations Check there is a character or line mark at the head of the input, and return it.

chars-up-to : $Table\text{-}id \times Elementary\text{-}loc\text{-}value^* \times \mathbb{N} \to Exception\text{-}handler \xrightarrow{o} (Char^* \times \mathbb{N})$

chars-up-to $(tid, s, n)handler \triangleq$
let *possibles* $= \{ans \mid ans : Char^* \cdot$ len $ans \leq n \wedge ans \neq [] \wedge$
END-LINE $\notin$ elems $ans \wedge$
$isa\text{-}prefix\,(emit\,(tid, ans), s)\}$ in
if *possibles* $= \{\}$
then *chan-exception*(*textParseError*) *handler*
else let *choice* : $Char^*$ be st $\{choice\} \subseteq possibles$ in
return *mk-*$(choice,$ len $choice)$

530 529
470

annotations Return up to n characters from the front of s.

up-to : $Value^* \times \mathbb{N} \to Value^*$

up-to $(s, max) \triangleq$
let *ans* : $Value^*$ be st $isa\text{-}prefix\,(ans, s) \wedge$ len $ans \leq max \wedge ans \neq []$ in
ans

530

annotations Return a non empty subsequence up to length max from the front of s.

isa-prefix : $Value^* \times Value^* \to \mathbb{B}$

isa-prefix $(s1, s2) \triangleq$
$\exists i \in$ inds $s2 \cdot s1 = s2\,(1, \ldots, i)$

annotations Returns whether $S1$ is a prefix of $S2$

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.6 The Interface to Channels for New Device Modules

Additional device modules may be provided to allow the library to be used with other input sources and output destinations. These might include, for example, files opened with host-specific options or parameters or with host-specific behaviour, a windowing system, or a speech output device.

9.2.6.1 The Module IOLink

The module `IOLink` provides facilities that allow a user to provide specialized device modules for use with channels, following the pattern of the rest of the library.

A device needs to identify itself in order to allow a check to be made that device-dependent operations are applied only for channels opened to that device. To this end, values of the type `DeviceId` are used to identify new device modules, and are normally obtained by them during their initialization by a call to the procedure `AllocateDeviceId`.

A device module procedure provided for opening a channel can obtain a new channel by calling the procedure `MakeChan`. If a channel is allocated, but the call of the device module open procedure is not successful for some reason, the device module should release the channel by calling the procedure `UnMakeChan`, and return the value identifying the invalid channel to its client.

A call to `UnMakeChan` is also made on a successful call of a device module procedure provided for closing a channel.

If a call of a device module 'open' procedure is successful, then by calling the function procedure `DeviceTablePtrValue`, a device module can obtain a pointer (of the type `DeviceTablePtr`) to a 'device table' (of a record type `DeviceTable`) for the channel. The fields of this record will have been initialized by `MakeChan`, but the procedure can then change any fields of the device table needed to install its own values for the device data, supported operations, and flags.

Device tables have:

a) a field in which the device module can store private data,

b) a field in which the value identifying the device module is stored,

c) a field in which the value identifying the channel is stored,

d) a field in which the read result is stored,

e) a field in which device error numbers are stored prior to the raising of a device error exception,

f) a field in which flags are stored indicating those which apply,

g) a field for each device procedure.

(The fields are initialized by `MakeChan` to the values shown in the definition module below.)

By calling the function procedure `IsDevice`, a device module can enquire whether it was responsible for opening a given channel. This allows it to implement a corresponding enquiry function that is exported from the device module itself.

Client modules may raise appropriate exceptions; to support this facility, the type `DevExceptionRange` and the procedure `RAISEdevException` can be used.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9.2.6.1.1 The Definition Module of IOLink

```
DEFINITION MODULE IOLink;

  (* Types and procedures for the standard implementation of channels *)

IMPORT IOChan, IOConsts, ChanConsts, SYSTEM;

TYPE
  DeviceId;
    (* Values of this type are used to identify new device modules, and are normally
       obtained by them during their initialization.
    *)

PROCEDURE AllocateDeviceId (VAR did: DeviceId);
  (* Allocates a unique value of type DeviceId, and assigns this value to did. *)

PROCEDURE MakeChan (did: DeviceId; VAR cid: IOChan.ChanId);
  (* Attempts to make a new channel for the device module identified by did.  If no more
     channels can be made, the identity of the invalid channel is assigned to cid.
     Otherwise, the identity of a new channel is assigned to cid.
  *)

PROCEDURE UnMakeChan (did: DeviceId; VAR cid: IOChan.ChanId);
  (* If the device module identified by did is not the module that made the channel
     identified by cid, the exception wrongDevice is raised; otherwise the channel is
     deallocated, and the value identifying the invalid channel is assigned to cid.
  *)

TYPE
  DeviceTablePtr = POINTER TO DeviceTable;
    (* Values of this type are used to refer to device tables *)

TYPE
  LookProc =      PROCEDURE (DeviceTablePtr, VAR CHAR, VAR IOConsts.ReadResults);
  SkipProc =      PROCEDURE (DeviceTablePtr);
  SkipLookProc =  PROCEDURE (DeviceTablePtr, VAR CHAR, VAR IOConsts.ReadResults);
  WriteLnProc =   PROCEDURE (DeviceTablePtr);
  TextReadProc =  PROCEDURE (DeviceTablePtr, SYSTEM.ADDRESS, CARDINAL, VAR CARDINAL);
  TextWriteProc = PROCEDURE (DeviceTablePtr, SYSTEM.ADDRESS, CARDINAL);
  RawReadProc =   PROCEDURE (DeviceTablePtr, SYSTEM.ADDRESS, CARDINAL, VAR CARDINAL);
  RawWriteProc =  PROCEDURE (DeviceTablePtr, SYSTEM.ADDRESS, CARDINAL);
  GetNameProc =   PROCEDURE (DeviceTablePtr, VAR ARRAY OF CHAR);
  ResetProc =     PROCEDURE (DeviceTablePtr);
  FlushProc =     PROCEDURE (DeviceTablePtr);
  FreeProc =      PROCEDURE (DeviceTablePtr);
      (* Carry out the operations involved in closing the corresponding channel, including
         flushing buffers, but do not unmake the channel.
      *)

TYPE
  DeviceData = SYSTEM.ADDRESS;

  DeviceTable =
    RECORD                          (* Initialized by MakeChan to: *)
      cd: DeviceData;               (* the value NIL *)
      did: DeviceId;                (* the value given in the call of MakeChan *)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      cid: IOChan.ChanId;          (* the identity of the channel *)
      result: IOConsts.ReadResults;(* the value notKnown *)
      errNum: IOChan.DeviceErrNum; (* undefined *)
      flags: ChanConsts.FlagSet;   (* ChanConsts.FlagSet{} *)
      doLook: LookProc;            (* raise exception notAvailable *)
      doSkip: SkipProc;            (* raise exception notAvailable *)
      doSkipLook: SkipLookProc;    (* raise exception notAvailable *)
      doLnWrite: WriteLnProc;      (* raise exception notAvailable *)
      doTextRead: TextReadProc;    (* raise exception notAvailable *)
      doTextWrite: TextWriteProc;  (* raise exception notAvailable *)
      doRawRead: RawReadProc;      (* raise exception notAvailable *)
      doRawWrite: RawWriteProc;    (* raise exception notAvailable *)
      doGetName: GetNameProc;      (* return the empty string *)
      doReset: ResetProc;          (* do nothing *)
      doFlush: FlushProc;          (* do nothing *)
      doFree: FreeProc;            (* do nothing *)
    END;

PROCEDURE DeviceTablePtrValue (cid: IOChan.ChanId; did: DeviceId): DeviceTablePtr;
  (* If the device module identified by did is not the module that made the channel
     identified by cid, the exception wrongDevice is raised; otherwise returns a pointer to
     the device table for the channel.
  *)

PROCEDURE IsDevice (cid: IOChan.ChanId; did: DeviceId): BOOLEAN;
  (* Tests if the device module identified by did is the module that made the channel
     identified by cid.
  *)

TYPE
  DevExceptionRange = [IOChan.notAvailable .. IOChan.textParseError];

PROCEDURE RAISEdevException (cid: IOChan.ChanId; did: DeviceId;
                             x: DevExceptionRange; s: ARRAY OF CHAR);
  (* If the device module identified by did is not the module that made the channel
     identified by cid, the exception wrongDevice is raised; otherwise the given exception
     is raised, and the string value in s is included in the exception message.
  *)

PROCEDURE IsIOException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional execution state
     because of the raising of an exception from ChanExceptions;
     otherwise returns FALSE.
  *)

PROCEDURE IOException (): IOChan.ChanExceptions;
  (* If the current coroutine is in the exceptional execution state because of the
     raising of an exception from ChanExceptions, returns the corresponding
     enumeration value, and otherwise raises an exception.
  *)

END IOLink.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.6.1.2 The Procedures of IOLink

The procedure `AllocateDeviceId`

The call `AllocateDeviceId(did)` shall allocate a unique value of the type `DeviceId`, and assign this value to `did`.

AllocateDeviceId : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

AllocateDeviceId (*args*)*handler* ≙
  let [*did*] = *args* in
  def *newid* = *allocate-device-id*() in 539
  assign(*did*, *newdid*) *handler* 150

The procedure `MakeChan`

The call `MakeChan(did,cid)` shall attempt to allocate a new channel for the device module identified by `did`. If no more channels can be allocated, the value identifying the invalid channel shall be assigned to `cid`. Otherwise, the call shall assign a value identifying a new initialized channel to `cid`.

MakeChan : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

MakeChan (*args*)*handler* ≙
  let [*did*, *cid*] = *args* in
  def *mk-*(*newcid*, -) = *make-a-channel*(*did*, { }, {↦}) in 540
  assign(*cid*, *newcid*) *handler* 150

The procedure `UnMakeChan`

Provided the device module identified by `did` is the module that made the channel identified by `cid`, the call `UnMakeChan(did,cid)` shall deallocate the channel identified by `cid`, and shall assign the value identifying the invalid channel to `cid`; otherwise the exception `wrongDevice` shall be raised.

UnMakeChan : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

UnMakeChan (*args*)*handler* ≙
  let [*did*, *cid*] = *args* in
  def *tid* = *table-id*(*cid*) *handler* in 539
  def *realdid* = *get-device-id*(*tid*) in 540
  if *realdid* = *did*
  then (*release-a-channel*(*cid*, *tid*) ; 540
    assign(*cid*, *badchan* ()) *handler* 150 538
)
  else *chan-exception*(*wrongDevice*) *handler* 470

The procedure `DeviceTablePtrValue`

Provided that the device module identified by `did` is the module that made the channel identified by `cid`, the value of the call `DeviceTablePtrValue(cid,did)` shall be a pointer to the device table for the channel identified by `cid`; otherwise the exception `wrongDevice` shall be raised.

DeviceTablePtrValue : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Table-id*

DeviceTablePtrValue (*args*)*handler* ≙
  let [*cid*, *did*] = *args* in
  def *tid* = *table-id*(*cid*) *handler* in 539
  def *realdid* = *get-device-id*(*tid*) in 540
  if *realdid* = *did* then return *tid*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  else chan-exception(wrongDevice) handler                                    470
```

The procedure `IsDevice`

The value of the call `IsDevice(cid,did)` shall be *true* if the device module identified by `did` is the module that made the channel identified by `cid`, and otherwise shall be *false*.

```
IsDevice : Arguments → Exception-handler →° 𝔹
IsDevice (args)handler △
  let [cid, did] = args in
  def tid = table-id(cid) handler in                                          539
  def realdid = get-device-id(tid) in                                         540
  return realdid = did
```

The procedure `RAISEdevException`

Provided that the device module identified by `did` is the module that made the channel identified by `cid`, the call `RAISEdevException(cid,did,x,s)` shall raise the exception given by `x`, and shall include the string value in `s` in the exception message; otherwise the exception `wrongDevice` shall be raised.

```
RAISEdevException : Arguments → Exception-handler →° ()
RAISEdevException (args)handler △
  let [cid, did, x, s] = args in
  def tid = table-id(cid) handler in                                          539
  def realdid = get-device-id(tid) in                                         540
  if realdid = did then chan-exception-using(x, pure-string (s)) handler      471 579
  else chan-exception(wrongDevice) handler                                    470
```

The procedure `IsIOException`

If the calling coroutine is in the state of exceptional execution because of the raising of an exception from `IOChan.ChanExceptions`, the value of the call `IsIOException()` shall be *true*; otherwise the value shall be *false*.

```
IsIOException : Arguments → Exception-handler →° 𝔹
IsIOException (args)handler △
  def source = iolib-source in
  def res = m-IsCurrentSource([source]) in                                    386
  return res
```

The procedure `IOException`

If the calling coroutine is in the state of exceptional execution because of the raising of an exception from `IOChan.ChanExceptions`, the value of the call `IOException()` shall be the value that identifies the raised exception; otherwise the language exception `exException` shall be raised.

```
IOException : Arguments → Exception-handler →° Value
IOException (args)handler △
  def source = iolib-source in
  def n = m-CurrentNumber([source]) handler in                                386
  let res = val-enumeration (n, chanexceptions) in                            330
  return res
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NOTE — A single value of `EXCEPTIONS.ExceptionSource` is used to identify the source of input/output library exceptions corresponding to `IOChan.ChanExceptions`. The procedures `IsIOException` and `IOException` are included so that this value need not be exported for corresponding procedures to be provided through the `IOChan` interface.

9.2.7 The State for the Input/Output Library

Domains

types

$$\mathit{Std\text{-}chan\text{-}id} = \text{STDIN} \mid \text{STDOUT} \mid \text{STDERR} \mid \text{NULLCHAN} \mid \text{BADCHAN};$$

$$\mathit{New\text{-}chan\text{-}id} = \text{token}_{\mathit{new\text{-}chan\text{-}id}};$$

$$\mathit{Chan\text{-}id} = \mathit{Std\text{-}chan\text{-}id} \mid \mathit{New\text{-}chan\text{-}id};$$

$$\mathit{Table\text{-}id} = \text{token}_{\mathit{table\text{-}id}};$$

$$\mathit{Std\text{-}dev\text{-}id} = \text{STDFILE} \mid \text{STREAMFILE} \mid \text{SEQFILE} \mid \text{RNDFILE} \mid \text{TERMFILE} \mid \text{PROGRAMARGS};$$

$$\mathit{New\text{-}dev\text{-}id} = \text{token}_{\mathit{new\text{-}dev\text{-}id}};$$

$$\mathit{Dev\text{-}id} = \mathit{Std\text{-}dev\text{-}id} \mid \mathit{New\text{-}dev\text{-}id};$$

$$\mathit{Operation} = \text{LOOK} \mid \text{SKIP} \mid \text{SKIPLOOK} \mid \text{WRITELN} \mid \text{WRITETEXT} \mid \text{WRITERAW} \mid \text{READTEXT} \mid \text{READRAW} \mid \text{NAME} \mid \text{FLUSH} \mid \text{RESET} \mid \text{FREE};$$

$$\begin{aligned}\mathit{Device\text{-}table} :: \mathit{id} &: \mathit{Dev\text{-}id}\\ \mathit{properties} &: \mathit{Set\text{-}value}\\ \mathit{ops} &: \mathit{Operation} \xrightarrow{m} \mathit{Procedure\text{-}id}\\ \mathit{error} &: \mathbb{Z}\\ \mathit{result} &: \mathit{Enumeration\text{-}value};\end{aligned}$$

$$\mathit{File} = \mathit{FileStream} \mid \mathit{FileSeq} \mid \mathit{FileRnd};$$

$$\mathit{File\text{-}id} = \text{token}_{\mathit{file\text{-}id}};$$

$$\mathit{External\text{-}IOLib\text{-}state} :: \mathit{filestore} : \mathit{File\text{-}id} \xrightarrow{m} \mathit{File}$$

state *IOLib* of

$$\begin{aligned}\mathit{iolib\text{-}external} &: \mathit{File\text{-}id} \xrightarrow{m} \mathit{File}\\ \mathit{iolib\text{-}channels} &: \mathit{Chan\text{-}id} \xleftrightarrow{m} \mathit{Table\text{-}id}\\ \mathit{iolib\text{-}devtables} &: \mathit{Table\text{-}id} \xleftrightarrow{m} \mathit{Device\text{-}table}\\ \mathit{iolib\text{-}devids} &: \mathit{New\text{-}dev\text{-}id}\text{-set}\\ \mathit{iolib\text{-}defin} &: \mathit{Chan\text{-}id}\\ \mathit{iolib\text{-}defout} &: \mathit{Chan\text{-}id}\\ \mathit{iolib\text{-}deferr} &: \mathit{Chan\text{-}id}\\ \mathit{iolib\text{-}source} &: \mathit{Exception\text{-}source}\end{aligned}$$

$$\text{inv } \mathit{mk\text{-}IOLib}(\mathit{channels}, \mathit{devtables}, \text{-}, \text{-}, \text{-}, \text{-}) \triangleq \text{rng } \mathit{channels} = \text{dom } \mathit{devtables}$$

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  init mk-IOLib (channels, devtables, defin, defout, deferr, -) ≜
      let bad-tid, null-tid, in-tid, out-tid, err-tid : Table-id be st
              is-uniques ([bad-tid, null-tid, in-tid, out-tid, err-tid]) in
      channels = {BADCHAN ↦ bad-tid,
       NULLCHAN ↦ null-tid,
       STDIN ↦ in-tid,
       STDOUT ↦ out-tid,
       STDERR ↦ err-tid} ∧
      devtables = {bad-tid ↦ bad-table,
       null-tid ↦ null-table,
       in-tid ↦ in-table,
       out-tid ↦ out-table,
       err-tid ↦ err-table} ∧
      defin = STDIN ∧
      defout = STDOUT ∧
      deferr = STDERR
end
```

614

Auxiliaries

Access to the external state for the input/output library

read-external-IOLib-state : *External-IOLib-state* $\xrightarrow{o}$ ()

read-external-IOLib-state (*es*) ≜
  iolib-external := *es*

annotations Install the external filestore in the library state.

write-external-IOLib-state : () $\xrightarrow{o}$ *External-IOLib-state*

write-external-IOLib-state () ≜
  return *iolib-external*

annotations Return the final state of the external filestore.

read-file : *Value* $\xrightarrow{o}$ *File*

read-file (*name*) ≜
  implementation-defined

annotations Read in a file named by the given value by translating that value into a value of type *File-id* and recovering the file associated with that *file-id* from the *iolib-external* component of the state.

write-file : *Value* × *File* $\xrightarrow{o}$ ()

write-file (*name*, *f*) ≜
  implementation-defined

annotations Write file *f* to the destination named by the given value by translating that value into a value of type *File-id* and updating the file associated with that *file-id* in the *iolib-external* component of the state.

Access to the state for the input/output library

stdinchan : () → *Chan-id*

stdinchan () ≜
  STDIN

annotations Return the id for the standard input channel.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

stdoutchan : () → *Chan-id*

stdoutchan () ≙
  STDOUT

annotations Return the id for the standard output channel.

stderrchan : () → *Chan-id*

stderrchan () ≙
  STDERR

annotations Return the id for the standard error channel.

nullchan : () → *Chan-id*

nullchan () ≙
  NULLCHAN

annotations Return the id for the null channel.

badchan : () → *Chan-id*

badchan () ≙
  BADCHAN

annotations Return the id for the invalid channel.

inchan : () $\xrightarrow{o}$ *Chan-id*

inchan () ≙
  def *cid* = *iolib-defin* in
  return *cid*

annotations Return the id for the current default input channel.

setinchan : *Chan-id* $\xrightarrow{o}$ ()

setinchan (*cid*) ≙
  iolib-defin := *cid*

annotations Set cid to be the id for the current default input channel.

outchan : () $\xrightarrow{o}$ *Chan-id*

outchan () ≙
  def *cid* = *iolib-defout* in
  return *cid*

annotations Return the id for the current default output channel.

setoutchan : *Chan-id* $\xrightarrow{o}$ ()

setoutchan (*cid*) ≙
  iolib-defout := *cid*

annotations Set cid to be the id for the current default output channel.

errchan : () $\xrightarrow{o}$ *Chan-id*

errchan () ≙
  def *cid* = *iolib-deferr* in
  return *cid*

annotations Return the id for the current default error channel.

seterrchan : *Chan-id* $\xrightarrow{o}$ ()

seterrchan (*cid*) ≙
  iolib-deferr := *cid*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

annotations Set cid to be the id for the current default error channel.

$table\text{-}id : Chan\text{-}id \to Exception\text{-}handler \xrightarrow{o} Table\text{-}id$

$table\text{-}id\,(cid)handler \triangleq$
  if $cid \notin \text{dom } channels$
  then $chan\text{-}exception(notAChannel)\ handler$ 470
  else def $chans = iolib.channels$ in
    return $chans\,(cid)$

annotations Return the table id corresponding to a channel id.

$device\text{-}op : Table\text{-}id \times Operation \xrightarrow{o} Procedure\text{-}id$

$device\text{-}op\,(tid, oper) \triangleq$
  def $dtbl = iolib\text{-}devtables\,(tid)$ in
  return $dtbl.ops\,(oper)$

annotations Return the procedure value corresponding to the desired device operation.

$set\text{-}props : Table\text{-}id \times Set\text{-}value \xrightarrow{o} ()$

$set\text{-}props\,(tid, props) \triangleq$
  def $dtbl = iolib\text{-}devtables\,(tid)$ in
  $iolib\text{-}devtables\,(tid) := \mu\,(dtbl, properties \mapsto props)$

$get\text{-}props : Table\text{-}id \xrightarrow{o} Set\text{-}value$

$get\text{-}props\,(tid) \triangleq$
  def $dtbl = iolib\text{-}devtables\,(tid)$ in
  let $mk\text{-}Device\text{-}table\,(\text{-}, props, \text{-}, \text{-}, \text{-}) = dtbl$ in
  return $props$

$get\text{-}error : Table\text{-}id \xrightarrow{o} \mathbb{N}$

$get\text{-}error\,(tid) \triangleq$
  def $dtbl = iolib\text{-}devtables\,(tid)$ in
  let $mk\text{-}Device\text{-}table\,(\text{-}, \text{-}, \text{-}, error, \text{-}) = dtbl$ in
  return $error$

$rdresult : Table\text{-}id \xrightarrow{o} Enumeration\text{-}value$

$rdresult\,(tid) \triangleq$
  def $dtbl = iolib\text{-}devtables\,(tid)$ in
  let $mk\text{-}Device\text{-}table\,(\text{-}, \text{-}, \text{-}, \text{-}, res) = dtbl$ in
  return res

$set\text{-}rdresult : Table\text{-}id \times Enumeration\text{-}value \xrightarrow{o} ()$

$set\text{-}rdresult\,(tid, res) \triangleq$
  def $dtbl = iolib\text{-}devtables\,(tid)$ in
  $iolib\text{-}devtables\,(tid) := \mu\,(dtbl, result \mapsto res)$

$allocate\text{-}device\text{-}id : () \xrightarrow{o} New\text{-}dev\text{-}id$

$allocate\text{-}device\text{-}id\,() \triangleq$
  def $devids = iolib\text{-}devids$ in
  let $newdid : New\text{-}dev\text{-}id$ be st $\neg \exists\, d \in devids \cdot d = newdid$ in
  $(iolib\text{-}devids := devids \cup \{newid\};$
   return $newid$
)

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
get-device-id : Table-id ⟶° Dev-id
get-device-id (tid) ≙
  def dtbl = iolib-devtables (tid) in
  let mk-Device-table (realdid, -, -, -, -) = dtbl in
  return realdid

make-a-channel : Dev-id × Set-value × (Operation ⟶ᵐ Procedure-id) ⟶° (Chan-id × [Table-id])
make-a-channel (did, props, ops) ≙
  def canmake = can-make-a-channel() in                                    540
  if ¬ canmake
  then return mk-(BADCHAN, nil )
  else def channels = iolib-channels in
       def devtables = iolib-devtables in
       let cid : New-chan-id be st cid ∉ dom channels in
       let tid : Table-id be st tid ∉ rng channels in
       let dtbl = mk-Device-table (did, props, default-ops † ops, -, notKnown) in
       ( || (iolib-devtables := devtables ∪ {tid ↦ dtbl},
             iolib-channels := channels ∪ {cid ↦ tid});
        return mk-(cid, tid)
        )

release-a-channel : Chan-id × Table-id ⟶° ()
release-a-channel (cid, tid) ≙
  def devtables = iolib-devtables in
  def channels = iolib-channels in
  || (iolib-devtables := {tid} ⩤ devtables,
      iolib-channels := {cid} ⩤ channels)

can-make-a-channel : () ⟶° 𝔹
can-make-a-channel () ≙
  return implementation-dependent
```

annotations Returns *true* if and only if there are sufficient resources to make a new channel.

Values for the pre-opened channels

values

in-props : *Set-value* = implementation-defined;

in-ops : *Operation* ⟶ᵐ *Procedure-id* = implementation-defined;

in-table = *mk-Device-table* (STDFILE, *in-props*, *in-ops*, -, *notKnown*);

out-props : *Set-value* = implementation-defined;

out-ops : *Operation* ⟶ᵐ *Procedure-id* = implementation-defined;

out-table = *mk-Device-table* (STDFILE, *out-props*, *out-ops*, -, *notKnown*);

err-props : *Set-value* = implementation-defined;

err-ops : *Operation* ⟶ᵐ *Procedure-id* = implementation-defined;

err-table = *mk-Device-table* (STDFILE, *err-props*, *err-ops*, -, *notKnown*)

The invalid channel shall be a channel on which no data transfer operations are available. Enquiries on the invalid channel shall indicate that no operations are available.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

values

bad-ops =
{LOOK ↦ *notavail-proc-pid*,
SKIP ↦ *notavail-proc-pid*,
SKIPLOOK ↦ *notavail-proc-pid*,
WRITELN ↦ *notavail-proc-pid*,
READTEXT ↦ *notavail-proc-pid*,
WRITETEXT ↦ *notavail-proc-pid*,
READRAW ↦ *notavail-proc-pid*,
WRITERAW ↦ *notavail-proc-pid*,
NAME ↦ *imp-name-pid*,
FLUSH ↦ *notavail-proc-pid*,
RESET ↦ *notavail-proc-pid*}

;

bad-dens =
{*notavail-proc-pid* ↦ *make-proper-procden* (*notavail-proc*), 339
imp-name-pid ↦ *make-proper-procden* (*imp-name*)} 339

bad-props = { };

badtable = *mk-Device-table* (-, *badprops*, *bad-ops*, -, -)

The default read and write operations on a new channel shall raise the exception `notAvailable`; the default *GetName* operation shall return the empty string; the default *Reset*, *Flush*, and *Free* operations shall do nothing.

values

default-ops =
{LOOK ↦ *notavail-proc-pid*,
SKIP ↦ *notavail-proc-pid*,
SKIPLOOK ↦ *notavail-proc-pid*,
WRITELN ↦ *notavail-proc-pid*,
READTEXT ↦ *notavail-proc-pid*,
WRITETEXT ↦ *notavail-proc-pid*,
READRAW ↦ *notavail-proc-pid*,
WRITERAW ↦ *notavail-proc-pid*,
NAME ↦ *def-nameop-pid*,
RESET ↦ *null-proc-pid*,
FLUSH ↦ *null-proc-pid*,
FREE ↦ *null-proc-pid*}

;

default-dens =
{*def-nameop-pid* ↦ *make-proper-procden* (*def-nameop*)} 339

The null device shall support all operations by discarding all data written to it, or by giving an immediate end of input indication on reading.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

values

null-ops =
{LOOK ↦ *null-look-pid*,
SKIP ↦ *null-skip-pid*,
SKIPLOOK ↦ *null-skiplook-pid*,
WRITELN ↦ *null-proc-pid*,
READTEXT ↦ *null-readtext-pid*,
WRITETEXT ↦ *null-proc-pid*,
READRAW ↦ *null-readraw-pid*,
WRITERAW ↦ *null-proc-pid*,
NAME ↦ *null-name-pid*,
FLUSH ↦ *null-proc-pid*,
RESET ↦ *null-proc-pid*,
FREE ↦ *null-proc-pid*}

;

null-dens =
{*null-look-pid* ↦ *make-proper-procden* (*null-look*), 339
null-skip-pid ↦ *make-proper-procden* (*null-skip*), 339
null-skiplook-pid ↦ *make-proper-procden* (*null-skiplook*), 339
null-proc-pid ↦ *make-proper-procden* (*null-proc*), 339
null-readtext-pid ↦ *make-proper-procden* (*null-readtext*), 339
null-readraw-pid ↦ *make-proper-procden* (*null-readraw*), 339
}

;

null-props = {*readFlag*, *writeFlag*, *textFlag*, *rawFlag*};

null-table = *mk-Device-table* (STDFILE, *null-props*, *null-ops*, -, *notKnown*)

Device procedures for the pre-opened channels

notavail-proc : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()
notavail-proc (*args*)*handler* ≙
  chan-exception(*notAvailable*) *handler* 470

imp-nameop : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()
imp-nameop (*args*)*handler* ≙
  let [*tid*, *s*] = *args* in
  def *name* : *Char** = implementation-defined in
  assign(*s*, *name*) *handler* 150

def-nameop : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()
def-nameop (*args*)*handler* ≙
  let [*tid*, *s*] = *args* in
  assign(*s*, [ ]) *handler* 150

null-look : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()
null-look (*args*)*handler* ≙
  let [*tid*, *ch*, *res*] = *args* in
  (*set-rdresult*(*tid*, *endOfInput*) ; 539
   assign(*res*, *endOfInput*) *handler* 150
)

null-skip : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()
null-skip (*args*)*handler* ≙
  chan-exception(*skipAtEnd*) *handler* 470

null-skiplook : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()
null-skiplook (*args*)*handler* ≙
  chan-exception(*skipAtEnd*) *handler* 470

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
null-readtext : Arguments → Exception-handler ⁰→ ()
null-readtext (args)handler ≙
  let [tid, -, -, nRead] = args in
  (set-rdresult(tid, endOfInput) ;                                    539
   assign(nRead, 0) handler                                           150
   )

null-readraw : Arguments → Exception-handler ⁰→ ()
null-readraw (args)handler ≙
  let [tid, -, -, nRead] = args in
  (set-rdresult(tid, endOfInput) ;                                    539
   assign(nRead, 0) handler                                           150
   )

null-name : Arguments → Exception-handler ⁰→ ()
null-name (args)handler ≙
  let [-, s] = args in
  def name : Char* = implementation-defined in
  assign(s, name) handler                                             150

null-proc : Arguments → Exception-handler ⁰→ ()
null-proc (args)handler ≙
  skip
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.3 Concurrent Programming

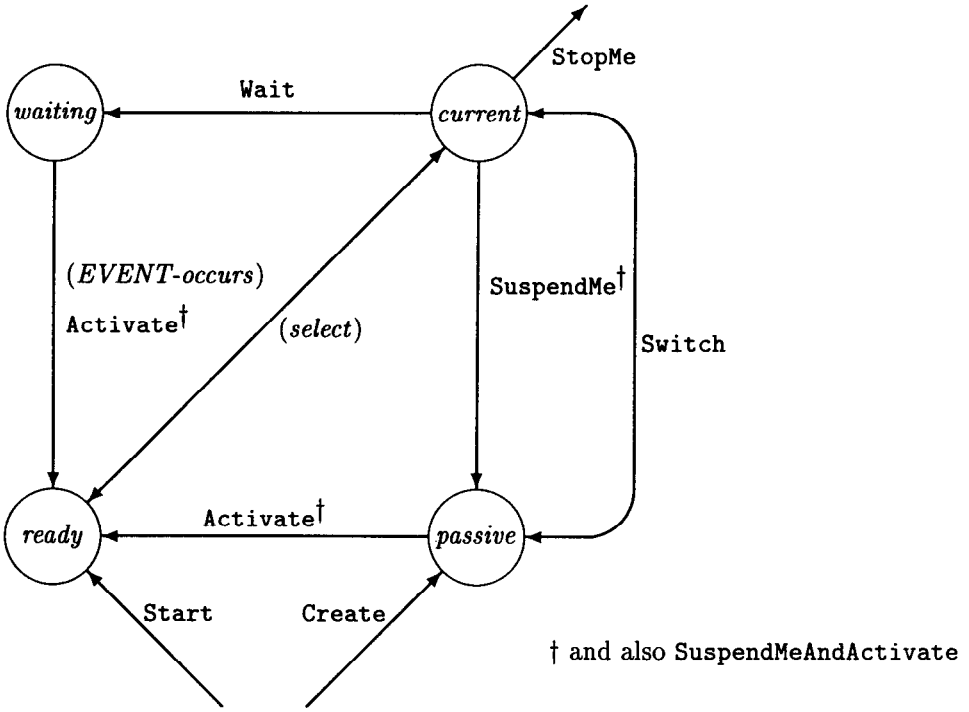
9.3.1 The Module Processes

The module `Processes` provides a basic set of facilities for use in concurrent programs. These can be used on their own, or in conjunction with those from the module `Semaphores` which provide for potentially parallel parts of the program to exclude one another from regions of interaction.

A concurrent program consists of a number of processes, each of which may potentially run in parallel with the others but is distinguishable from them. At any one time, a process may be in one of four states: It may be *ready*, that is, eligible to use the processor but not actually doing so. It may be *current*, that is, actually using the processor. It may be *passive*, that is, ineligible to use the processor until another process makes it eligible. Lastly, it may be *waiting*, that is, ineligible to use the processor until the occurrence of one of a set of events for which it is waiting.

At all times there must be at least one process using the processor, or, if no process is eligible, there must be at least one process waiting for some external event.

The relationships between these states, together with the operations that lead to a change of state, are depicted in the following figure. Operations given in parentheses are not directly invokable from a program.



Two general styles of use are envisaged, and both may be present in a single program.

In the first style (using `Switch`), the model is of a set of closely coupled processes, which explicitly choose which of them is to run next, and which pass information between themselves as part of the choice. The intention is to provide a higher level coroutine-like facility between processes of the same urgency.

In the second style (using `Activate` and `SuspendMe`), the processes are written to be less dependent on one another, and the choice of which of them runs is left to an internal scheduler, which is invoked whenever one process is suspended or another one is reactivated. This internal scheduler makes use of the fact that each process has an associated static 'urgency', specified by an `INTEGER` parameter when it is first created. The scheduler ensures that it cannot be the case that a process eligible to use the processor has an urgency greater than one of the processes currently doing so. The 'main process' (the parent program) is given a default urgency of zero; for other processes, the more positive the value of urgency, the more urgent is the process.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Those processes that suspend themselves to wait for external events must first associate themselves with one or more sources of such events. This part of ISO/IEC 10514 does not prescribe how events occur, or what the sources of events must be, other than to require that they be mapped in an implementation-defined way to values of the pervasive type `CARDINAL`, and to require that a source of events cannot be connected to more than one process simultaneously.

NOTES

1 There is no requirement that pre-emptive scheduling be employed, although an implementation is free to incorporate such scheduling if this is desired.

2 Although this part of ISO/IEC 10514 applies to single-processor machines, if a pre-emptive (time-sliced) scheduler is present there may conceptually be more than one 'current' process; the description above is phrased so as to emphasize this.

3 A program that uses the module `Processes` should not make explicit use of coroutines (except, perhaps, in the implementation of `Processes` itself).

4 This is a partial specification: Various of the procedures in the modules `Processes` and `Semaphores` have semantics expressed in terms of preconditions. Their behaviour in situations where these preconditions are not met is deliberately not specified; in particular, the raising of exceptions is not required. This is a deliberate decision, and allows for maximum efficiency in the implementation of these modules.

CHANGE — This module is not based on the module `Processes` described in *Programming in Modula-2*.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.3.1.1 The Definition Module of Processes

```
DEFINITION MODULE Processes;

  (* This module allows concurrent algorithms to be expressed using processes.  A process is
     a unit of a program that has the potential to run in parallel with other processes.
  *)

IMPORT SYSTEM;

TYPE
  ProcessId;                        (* Used to identify processes *)
  Parameter     = SYSTEM.ADDRESS;   (* Used to pass data between processes *)
  Body          = PROC;             (* Used as the type of a process body *)
  Urgency       = INTEGER;          (* Used by the internal scheduler *)
  Sources       = CARDINAL;         (* Used to identify event sources *)
  ProcessesExceptions =             (* Exceptions raised by this module *)
    (passiveProgram, processError);

(* The following procedures create processes and switch control between them. *)

PROCEDURE Create (procBody: Body; extraSpace: CARDINAL; procUrg: Urgency;
                  procParams: Parameter; VAR procId: ProcessId);
  (* Creates a new process with procBody as its body, and with urgency and parameters
     given by procUrg and procParams.  At least as much workspace (in units of
     SYSTEM.LOC) as is specified by extraSpace is allocated to the process.
     An identity for the new process is returned in procId.
     The process is created in the passive state; it will not run until activated.
  *)

PROCEDURE Start (procBody: Body; extraSpace: CARDINAL; procUrg: Urgency;
                 procParams: Parameter; VAR procId: ProcessId);
  (* Creates a new process, with parameters as for Create.
     The process is created in the ready state; it is eligible to run immediately.
  *)

PROCEDURE StopMe ();
  (* Terminates the calling process.
     The process must not be associated with a source of events.
  *)

PROCEDURE SuspendMe ();
  (* Causes the calling process to enter the passive state.  The procedure only returns
     when the calling process is again activated by another process.
  *)

PROCEDURE Activate (procId: ProcessId);
  (* Causes the process identified by procId to enter the ready state, and thus to become
     eligible to run again.
  *)

PROCEDURE SuspendMeAndActivate (procId: ProcessId);
  (* Executes an atomic sequence of SuspendMe() and Activate(procId). *)

PROCEDURE Switch (procId: ProcessId; VAR info: Parameter);
  (* Causes the calling process to enter the passive state; the process identified by procId
     becomes the currently executing process.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
      info is used to pass parameter information from the calling to the activated process.
      On return, info will contain information from the process that chooses to switch back to
      this one (or will be NIL if Activate or SuspendMeAndActivate are used instead of
      Switch).
  *)

PROCEDURE Wait ();
  (* Causes the calling process to enter the waiting state.  The procedure will return when
     the calling process is activated by another process, or when one of its associated
     eventSources has generated an event.
  *)

(* The following procedures allow the association of processes with sources of external
   events.
*)

PROCEDURE Attach (eventSource: Sources);
  (* Associates the specified eventSource with the calling process. *)

PROCEDURE Detach (eventSource: Sources);
  (* Dissociates the specified eventSource from the program. *)

PROCEDURE IsAttached (eventSource: Sources): BOOLEAN;
  (* Returns TRUE if and only if the specified eventSource is currently associated with
     one of the processes of the program.
  *)

PROCEDURE Handler (eventSource: Sources): ProcessId;
  (* Returns the identity of the process, if any, that is associated with the specified
     eventSource.
  *)

(* The following procedures allow processes to obtain their identity, parameters, and
   urgency.
*)

PROCEDURE Me (): ProcessId;
  (* Returns the identity of the calling process (as assigned when the process was first
     created).
  *)

PROCEDURE MyParam (): Parameter;
  (* Returns the value specified as procParams when the calling process was created. *)

PROCEDURE UrgencyOf (procId: ProcessId): Urgency;
  (* Returns the urgency established when the process identified by procId was first
     created.
  *)

(* The following procedure provides facilities for exception handlers. *)

PROCEDURE ProcessesException (): ProcessesExceptions;
  (* If the current coroutine is in the exceptional execution state because of the raising
     of a Processes exception, returns the corresponding enumeration value, and
     otherwise raises an exception.
  *)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
PROCEDURE IsProcessesException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional execution state
     because of the raising of an exception in a routine from this module; otherwise
     returns FALSE.
  *)

END Processes.
```

9.3.1.2 The Procedures of Processes

The semantics of certain of the procedures of `Processes` require that a process be selected from the set of processes that are eligible to run, and be scheduled for execution. The selection algorithm shall guarantee that no process eligible to use the processor has an urgency greater than one of the processes currently doing so.

NOTES

1 The procedures in this category are `Start`, `StopMe`, `SuspendMe`, `Activate`, `SuspendMeAndActivate` and `Wait`.

2 The urgency of a process is specified, when it is created, by a value of the type `INTEGER`, and cannot be changed dynamically. The more positive the value of urgency, the more urgent the process.

3 The selection algorithm is modelled in the formal definition by *select* — see 9.3.1.6.

Certain of the procedures of `Processes` require that a process be associated with a source of external events. There shall be an implementation-defined mapping of values of the type `CARDINAL` to such sources of events.

A source of events shall not be associated with more than one process at any instant.

NOTES

4 This part of ISO/IEC 10514 does not prescribe how events occur, or what the sources of events must be, other than in terms of this implementation-defined mapping.

5 This part of ISO/IEC 10514 does not specify the consequences if a value (of the type `Sources`) that is not mapped to a source of events is passed as an actual parameter to any of the procedures `Attach`, `Detach`, `IsAttached` or `Handler`.

The procedure `Create`

The call `Create(procBody,extraSpace,procUrg,procParams,procId)` shall create a new process. A unique value of the type `ProcessId` shall be assigned to `procId` as an identity for the process. The urgency and parameters for the created process shall be those given by `procUrg` and `procParams` respectively. `extraSpace` shall specify the amount of workspace (in units of `SYSTEM.LOC`) that is required by the process, *above any fixed overhead* needed by the implementation of `Processes`.

The process shall be ineligible to run (i.e. it is created in the *passive* state). When the process is first activated, it shall start execution by invoking the procedure that is denoted by `procBody`.

The usage made of the workspace shall be implementation-dependent.

If the end of this procedure body is reached, or if a return statement is executed in the procedure body, then the effect shall be the same as calling the protection domain exit procedure (if any), followed by an explicit call of `StopMe`.

NOTES

6 The process will be activated when another process makes it eligible by calling `Switch` or `Activate` or `SuspendMeAndActivate`.

7 The standard library makes no provision to handle exceptions generated by created processes, which must handle all exceptions themselves if it is wished to avoid exceptional termination of a program.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Create : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

Create (*args*)*handler* $\triangleq$
  let [-, -, *procurg*, *procparams*, *procid*] = *args* in
  def *pid* = *new-passive-process*(*procurg*, *procparams*) in 556
  assign(*procid*, *pid*) *handler* 150

annotations The formal definition does not model the association of a new process with the procedure given as its body. Neither does it model the effect of a return from that procedure.

The procedure Start

The call `Start(procBody,extraSpace,procUrg,procParams,procId)` shall have an identical effect to the call `Create(procBody,extraSpace,procUrg,procParams,procId)`, except that the created process shall be eligible to run immediately (i.e. it is created in the *ready* state).

Start : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

Start (*args*)*handler* $\triangleq$
  let [-, -, *procurg*, *procparams*, *procid*] = *args* in
  def *pid* = *new-passive-process*(*procurg*, *procparams*) in 556
  (*assign*(*procid*, *pid*) *handler*; *make-ready*(*pid*) ; *select*()) 150 556 557

The procedure StopMe

If the calling process is not be associated with any source of events, the call `StopMe()` shall cause the calling process to be terminated and removed from the system. The procedure shall not return; the calling process shall not again become eligible to run. If there are no other processes, then normal termination of the program shall be initiated. If there are other processes eligible to run, then one of them shall be selected for execution. The exception `passiveProgram` shall be raised if there are other processes, but none of them is eligible to run and none of them is waiting for an event to occur.

NOTES

8 If the main process stops, the other processes will continue to run.

9 The behaviour of `StopMe` in situations where the calling process is associated with a source of events is implementation-dependent.

StopMe : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

StopMe (*args*)*handler* $\triangleq$
  def *nonpassive* = *processes-current* ∪ *processes-ready* ∪ *processes-waiting*,
    passive = *processes-passive*,
    me = *whoami*() in 557
  if *nonpassive* ∪ *passive* = {*me*}
  then (*kill*(*me*) ; *note-HALT*()) 556 341
  else if *nonpassive* = {*me*}
    then *processes-exception*(*passiveProgram*) *handler* 554
    else (*kill*(*me*) ; *select*()) 556 557

annotations The definition of *kill* makes no distinction between the main process and other processes. If the main process stops, the other processes will continue to run.

The procedure SuspendMe

The call `SuspendMe()` shall cause the calling process to become ineligible to run (i.e. to enter the *passive* state). If there are other processes eligible to run, then one of them shall be selected for execution. The exception `passiveProgram` shall be raised if no other process is eligible to run and no other process is waiting for an event to occur.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NOTE 10 — The suspended process can be reactivated when another process again makes it eligible by calling `Switch` or `Activate` or `SuspendMeAndActivate`.

SuspendMe : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

SuspendMe (*args*)*handler* $\triangleq$
  (def *me* = *whoami*() in 557
   make-passive(*me*) ; *select*() ; 556 557
   if *current* ∪ *waiting* = { }
   then *processes-exception*(*passiveProgram*) *handler*) 554

The procedure `Activate`

If the process identified by `procId` is *passive* or *waiting*, the call `Activate(procId)` shall cause that process to become eligible to run (i.e. to enter the *ready* state); otherwise the call shall have no effect.

NOTE 11 — If the designated process was suspended by a call of `Wait`, it will become ready in the same way as if the event had occurred. Thus, if the procedure `Activate` (or `SuspendMeAndActivate`) is used to reactivate a waiting process, further checking will usually be required in that process to determine whether or not the event for which it was waiting had actually taken place.

Activate : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

Activate (*args*)*handler* $\triangleq$
  let [*procid*] = *args* in
  def *waiting* = *processes-waiting*,
      passive = *processes-passive* in
  if *procid* ∈ *waiting* ∪ *passive*
  then (*make-ready*(*procid*) ; *select*()) 556 557
  else skip

The procedure `SuspendMeAndActivate`

If the process identified by `procId` is *passive*, *waiting*, or is the calling process, the call `SuspendMeAndActivate(procId)` shall cause the calling process to become ineligible to run (i.e. to enter the *passive* state), and shall cause the process identified by `procId` to become eligible to run (i.e. to enter the *ready* state); otherwise the call shall have no effect.

NOTE 12 — `SuspendMeAndActivate(procId)` effectively performs an atomic (i.e. 'indivisible') sequence of the calls `SuspendMe()` and `Activate(procId)`. If applied to the identity of the calling process, the effect is to force a scheduling operation.

SuspendMeAndActivate : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

SuspendMeAndActivate (*args*)*handler* $\triangleq$
  let [*procid*] = *args* in
  def *waiting* = *processes-waiting*,
      passive = *processes-passive*,
      me = *whoami*() in 557
  if *procid* ∈ *waiting* ∪ *passive* ∪ {*me*}
  then (*make-passive*(*me*) ; *make-ready*(*procid*) ; *select*()) 556 556 557
  else skip

The procedure `Switch`

If the calling process has an urgency no greater than that of the process identified by `procId`, the call `Switch(procId,info)` shall cause the calling process to become ineligible to run (i.e. to enter the *passive* state),

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

and shall resume execution of the process identified by `procId`. The exception `processError` shall occur (but need not be raised) if this process is already eligible to run. `info` shall be passed as a parameter to the process identified by `procId`. If the calling process is reactivated as a result of another process calling `Switch`, then `info` shall be assigned the value passed by that other call. If the calling process is reactivated as a result of another process calling `Activate` or `SuspendMeAndActivate`, then `info` shall be assigned the value `NIL`.

NOTE 13 — `Switch` is intended to allow a high-level coroutine facility for use within concurrent programs. Several consequences follow:

a) The process that is resumed shall only be able to retrieve the parameter passed to it as `info` if it was ineligible to run by virtue of itself having called `Switch`.

b) The behaviour of `Switch` in situations where the urgency of the calling process is greater than the urgency of the process identified by `procId` is implementation-dependent.

c) While a call of `Switch` may be used instead of `Activate` to activate a process, p, of a higher urgency than the caller, p will not be able to use `Switch` if it wishes to reactivate that caller — it will be obliged to use `Activate` or `SuspendMeAndActivate`.

d) If the designated process was suspended by a call of `Wait`, it will become ready in the same way as if the event had occurred. Thus, if the procedure `Switch` (or `Activate` or `SuspendMeAndActivate`) is used to reactivate a waiting process, further checking will usually be required in that process to determine whether or not the event for which it was waiting had actually taken place.

Switch : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

Switch (*args*)*handler* $\triangleq$
  let [*procid*, *info*] = *args* in
  def *waiting* = *processes-waiting*,
    passive = *processes-passive*,
    me = *whoami*() in 557
  if *procid* ∈ *waiting* ∪ *passive*
  then (*make-passive*(*me*) ; def *data* = *value-of-a-variable*(*info*) in 556 224
    make-current(*procid*, *data*) ; def *infos* = *processes-info* in 556
    assign(*info*, *infos* (*me*)) *handler*) 150
  else *processes-exception*(*processError*) *handler* 554

The procedure `Wait`

If the calling process is associated with a source of events, the call `Wait()` shall cause the calling process to become ineligible to run (i.e. to enter the *waiting* state). If there are other processes eligible to run, then one of them shall be selected for execution.

NOTES

14 One of the ready processes is selected for execution to replace the caller of `Wait`. This is on the assumption that, if there are ready processes that are not executing, the scheduler imposes a fixed upper limit on the number of executing processes.

15 The process will remain ineligible to run until an event occurs from one of the sources to which it is attached, or until it is made eligible by another process calling `Switch` or `Activate` or `SuspendMeAndActivate`.

16 If a process waiting for an event is reactivated by virtue of a call being made by another process to `Switch` or `Activate` or `SuspendMeAndActivate`, it will become ready in the same way as if the event had occurred. Thus, in such situations, further checking will usually be required to determine whether or not the event for which it was waiting had actually taken place.

17 The behaviour of `Wait` if the calling process has not been attached to a source of events, or if another process is attached to the event source after the calling process has called `Wait` is implementation-dependent. In such circumstances the calling process may never again become eligible to run, and the system could deadlock.

Wait : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

Wait (*args*)*handler* $\triangleq$
  (def *me* = *whoami*() in 557
  make-waiting(*me*) ; *select*()) 557 557

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The procedure `Attach`

The call `Attach(eventSource)` shall associate the source of events given by `eventSource` with the calling process. If the source of events is already associated with a process, then that association shall first be broken.

Attach : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

Attach (*args*)*handler* $\triangleq$
  let [*eventsource*] = *args* in
  (*disconnect*(*eventsource*) ; def *me* = *whoami*() in 557 557
   connect(*eventsource*, *me*)) 557

The procedure `Detach`

The call `Detach(eventSource)` shall dissociate the source of events given by `eventSource` from the program.

NOTE 18 — The call has no effect if the program is not associated with `eventSource`.

Detach : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

Detach (*args*)*handler* $\triangleq$
  let [*eventsource*] = *args* in
  disconnect(*eventsource*) ; 557

The procedure `IsAttached`

The value of the call `IsAttached(eventSource)` shall be *true* if and only if the source of events given by `eventSource` is associated with one of the processes in the program.

IsAttached : *Arguments* → *Exception-handler* $\xrightarrow{o}$ $\mathbb{B}$

IsAttached (*args*)*handler* $\triangleq$
  let [*eventsource*] = *args* in
  def *attached* = *processes-attached* in
  return *eventsource* ∈ dom *attached*

The procedure `Handler`

If the source of events identified by `eventSource` is associated with a process, the value of the call `Handler(eventSource)` shall be the identity of that process.

NOTE 19 — The value of the call `Handler(eventSource)` is implementation-dependent in the situation where the value of `eventSource` is not mapped to any real source of events.

Handler : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Pid*

Handler (*args*)*handler* $\triangleq$
  let [*eventsource*] = *args* in
  def *attached* = *processes-attached* in
  return *attached* (*eventsource*)
pre *eventsource* ∈ dom *processes-attached*

The procedure `Me`

The value of the call `Me()` shall be the identity of the calling process.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Me : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Pid*

Me (*args*)*handler* $\triangleq$
  def *me* = *whoami*() in
  return *me*

The procedure `MyParam`

The value of the call `MyParam()` shall be the value of the parameter denoted by `procParams` at the time the process was created.

MyParam : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Procdata*

MyParam (*args*)*handler* $\triangleq$
  def *me* = *whoami*(),
      params = *processes-params* in
  return *params* (*me*)

The procedure `UrgencyOf`

The value of the call `UrgencyOf(procId)` shall be the urgency of the process identified by `procId`.

NOTE 20 — This urgency of a process is statically assigned when the process is created; it is not possible for a process to alter its urgency dynamically.

UrgencyOf : *Arguments* → *Exception-handler* $\xrightarrow{o}$ $\mathbb{Z}$

UrgencyOf (*args*)*handler* $\triangleq$
  let [*procid*] = *args* in
  def *urgency* = *processes-urgency* in
  return *urgency* (*procid*)
pre $P \in$ dom *processes-urgency*

The procedure `IsProcessesException`

If the calling coroutine is in the state of exceptional execution because of the raising of a `Processes` exception, the value of the call `IsProcessesException()` shall be *true*; otherwise the value shall be *false*.

IsProcessesException : *Arguments* → *Exception-handler* $\xrightarrow{o}$ $\mathbb{B}$

IsProcessesException (*args*)*handler* $\triangleq$
  def *source* = *processes-source* in
  def *res* = *m-IsCurrentSource*([*source*]) in
  return *res*

The procedure `ProcessesException`

If the calling coroutine is in the state of exceptional execution because of the raising of a `Processes` exception, the value of the call `ProcessesException()` shall be the value that identifies the raised exception; otherwise the language exception `exException` shall be raised.

ProcessesException : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Value*

ProcessesException (*args*)*handler* $\triangleq$
  def *source* = *processes-source* in
  def *n* = *m-CurrentNumber*([*source*]) *handler* in
  let *res* = *val-enumeration* (*n*, *processesexceptions*) in
  return *res*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9.3.1.3 Exceptions Raised by Processes

The exceptions raised by `Processes` shall be identified by the values of the enumeration type `ProcessesExceptions`.

The detection of the exception `processError` shall be implementation-defined.

The character string associated with each of the exceptions raised by `Processes` shall be implementation-defined.

values

$processesexceptions = [\texttt{passiveProgram}, \texttt{processError}];$

$passiveProgram = mk\text{-}Enumeration\text{-}value\,(\texttt{passiveProgram}, processesexceptions);$

$processError = mk\text{-}Enumeration\text{-}value\,(\texttt{processError}, processesexceptions)$

$processes\text{-}exception : Enumeration\text{-}value \to Exception\text{-}handler \xrightarrow{o} ()$
$processes\text{-}exception\,(pe)handler \triangleq$
  if $processes\text{-}exception\text{-}detected\,(pe)$ 554
  then $raise\text{-}processes\text{-}exception(pe)\ handler$ 554
  else implementation-dependent

$processes\text{-}exception\text{-}detected : Enumeration\text{-}value \to \mathbb{B}$
$processes\text{-}exception\text{-}detected\,(pe) \triangleq$
  cases pe :
    $passiveProgram$ → true,
    $processError$ → implementation-defined
  end

$raise\text{-}processes\text{-}exception : Enumeration\text{-}value \to Exception\text{-}handler \xrightarrow{o} ()$
$raise\text{-}processes\text{-}exception\,(pe)handler \triangleq$
  def $source = processes\text{-}source$ in
  def $message = processes\text{-}exception\text{-}string(pe)$ in 554
  let $number \in$ inds $processesexceptions$ be st
      $se = val\text{-}enumeration\,(number, processesexceptions)$ in 330
  $m\text{-}RAISE([source, number, message])\ handler$ 383

$processes\text{-}exception\text{-}string : Enumeration\text{-}value \xrightarrow{o} Message$
$processes\text{-}exception\text{-}string\,(pe) \triangleq$
  implementation-defined

annotations The result is an implementation-defined string value.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9.3.1.4 The State for Processes

Domains

types

$Pid = \text{MAIN-ID} \mid \text{token}_{process\text{-}id}$;

$Event\text{-}Source = \text{token}_{event\text{-}source}$;

$Procdata = Pointer\text{-}value$

state *Processes* of
  processes-ready : *Pid*-set
  processes-current : *Pid*-set
  processes-passive : *Pid*-set
  processes-waiting : *Pid*-set
  processes-urgency : $Pid \xrightarrow{m} \mathbb{Z}$
  processes-attached : $Event\text{-}source \xrightarrow{m} Pid$
  processes-params : $Pid \xrightarrow{m} Procdata$
  processes-info : $Pid \xrightarrow{m} Procdata$
  processes-source : *Exception-source*

  inv *mk-Processes*(*Ready*, *Current*, *Passive*, *Waiting*, *Urgency*, *Attached*, -, -, -) $\triangleq$
    is-disj ({*Ready*, *Current*, *Passive*, *Waiting*}) $\wedge$ 614
    $Current = \{\} \Rightarrow Ready = \{\} \wedge$
    $Current \cup Waiting \neq \{\} \wedge$
    $\forall p \in Current \cdot \forall q \in Ready \cdot Urgency(p) \geq Urgency(q) \wedge$
    rng *Attached* $\subseteq$ dom *Urgency* $= Ready \cup Current \cup Passive \cup Waiting$
end

annotations *params* and *info* are used to model the passing of information between processes. *info* is used in the definition of `Switch` and *params* is used in the definition of `Create` and `Start`.

This formal definition does not model the association of a process with the program for that process. Neither does it describe the allocation and use of the workspace (see 6.12.8.3).

9.3.1.5 The Initialization of Processes

The initialization of the module `Processes` shall allocate a unique value of the exception source type to identify exceptions raised by the module.

After the module is initialized, there shall be exactly one process, known as the 'main process'. This process shall have an urgency of 0 and a parameter of `NIL`; initially, it shall not be associated with any source of events.

processes-initialization : $() \rightarrow Exception\text{-}handler \xrightarrow{o} ()$
processes-initialization ()*handler* $\triangleq$
  $\|$ (*processes-ready* := {}, *processes-passive* := {}, *processes-waiting* := {}, *processes-current* := {MAIN-ID},
    processes-urgency := {MAIN-ID $\mapsto$ 0},
    processes-attached := {$\mapsto$},
    processes-params := {MAIN-ID $\mapsto$ nil }, *processes-info* := {$\mapsto$},
    def *source* = *allocate-source*() in 342
    if *source* $\neq$ nil
    then *processes-source* := *source*
    else *mandatory-detection*(NO-EXCEPTION-SOURCE) *handler*) 396

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9.3.1.6 Auxiliary Formal Model Definitions

new-passive-process ($Urg : \mathbb{Z}, Parameter : Procdata$) $P : Pid$
ext wr $processes\text{-}passive : Pid$-set
wr $processes\text{-}urgency : Pid \xrightarrow{m} \mathbb{Z}$
wr $processes\text{-}params : Pid \xrightarrow{m} Procdata$
pre true
post let $P : Pid$ be st $P \notin \text{dom } \overleftarrow{processes\text{-}urgency}$ in
$processes\text{-}passive = \overleftarrow{processes\text{-}passive} \cup \{P\} \wedge$
$processes\text{-}urgency = \overleftarrow{processes\text{-}urgency} \cup \{P \mapsto Urg\} \wedge$
$processes\text{-}params = \overleftarrow{processes\text{-}params} \cup \{P \mapsto Parameter\}$

annotations Create a new process and make it ineligible to be selected to run.

Every process has a mapping from its identity to an urgency. The first line in the post-condition guarantees that the identity of each process is different from all the others that exist *at this time.* No requirement is imposed concerning the re-use of a process id belonging to a process that has been removed.

kill ($P : Pid$)
ext wr $processes\text{-}current : Pid$-set
wr $processes\text{-}urgency : Pid \xrightarrow{m} \mathbb{Z}$
rd $processes\text{-}attached : Event\text{-}source \xrightarrow{m} Pid$
wr $processes\text{-}params, processes\text{-}info : Pid \xrightarrow{m} Procdata$
pre $P \in processes\text{-}current \wedge P \notin \text{rng } processes\text{-}attached$
post $processes\text{-}current = \overleftarrow{processes\text{-}current} - \{P\} \wedge$
$processes\text{-}urgency = \{P\} \lhd \overleftarrow{processes\text{-}urgency} \wedge$
$processes\text{-}params = \{P\} \lhd \overleftarrow{processes\text{-}params} \wedge$
$processes\text{-}info = \{P\} \lhd \overleftarrow{processes\text{-}info}$

annotations Remove a process, permanently. The process must not be associated with a source of events. This definition requires no special action in the case $P =$ MAIN-ID.

make-passive ($P : Pid$)
ext wr $processes\text{-}current, processes\text{-}passive : Pid$-set
pre $P \in processes\text{-}current$
post $processes\text{-}current = \overleftarrow{processes\text{-}current} - \{P\} \wedge processes\text{-}passive = \overleftarrow{processes\text{-}passive} \cup \{P\}$

annotations Make a process ineligible to be selected to run.

make-ready ($P : Pid$)
ext wr $processes\text{-}ready, processes\text{-}passive, processes\text{-}waiting : Pid$-set
wr $processes\text{-}info : Pid \xrightarrow{m} Procdata$
pre $P \in processes\text{-}passive \vee P \in processes\text{-}waiting$
post $processes\text{-}ready = \overleftarrow{processes\text{-}ready} \cup \{P\} \wedge$
$processes\text{-}info = \overleftarrow{processes\text{-}info} \dagger \{P \mapsto \text{nil}\} \wedge$
$((P \in \overleftarrow{processes\text{-}passive} \wedge processes\text{-}passive = \overleftarrow{processes\text{-}passive} - \{P\} \wedge processes\text{-}waiting = \overleftarrow{processes\text{-}waiting}) \vee$
$(P \in \overleftarrow{processes\text{-}waiting} \wedge processes\text{-}waiting = \overleftarrow{processes\text{-}waiting} - \{P\} \wedge processes\text{-}passive = \overleftarrow{processes\text{-}passive}))$

annotations Make a process eligible to be selected to run.

make-current ($P : Pid, I : Procdata$)
ext wr $processes\text{-}current, processes\text{-}passive, processes\text{-}waiting : Pid$-set
rd $processes\text{-}urgency : Pid \xrightarrow{m} \mathbb{Z}$
wr $processes\text{-}info : Pid \xrightarrow{m} Procdata$
pre $P \in processes\text{-}passive \cup processes\text{-}waiting \wedge urgency(me) \leq urgency(P)$

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

post $processes\text{-}passive = \overleftarrow{processes\text{-}passive} - \{P\} \wedge$
$processes\text{-}waiting = \overleftarrow{processes\text{-}waiting} - \{P\} \wedge$
$processes\text{-}current = \overleftarrow{processes\text{-}current} \cup \{P\} \wedge$
$processes\text{-}info = \overleftarrow{processes\text{-}info} \dagger \{P \mapsto I\}$

annotations Allow a nominated process to run. No *select* operation is required to preserve *inv-Processes* because of the pre-condition.

whoami () $P : Pid$
ext rd $processes\text{-}current : Pid$-set
pre true
post $((\text{card } processes\text{-}current = 1) \wedge (P \in processes\text{-}current)) \vee$
$((\text{card } processes\text{-}current > 1) \wedge (P = scheduler\text{-}running))$

annotations If there is only one executing process, it is the caller. Otherwise, on a single-processor machine, the caller is that process selected for execution by a time-sliced scheduler. *scheduler-running* is a component of the state of such a scheduler, giving the identity of the running process.

select ()
ext wr $processes\text{-}ready, processes\text{-}current : Pid$-set
rd $processes\text{-}urgency : Pid \xrightarrow{m} \mathbb{Z}$
pre true
post $\forall p \in processes\text{-}ready \cdot (\neg \exists q \in processes\text{-}current \cdot (urgency(p) > urgency(q))) \wedge$
$(processes\text{-}current \cup processes\text{-}ready) = (\overleftarrow{processes\text{-}current} \cup \overleftarrow{processes\text{-}ready}) \wedge$
$\overleftarrow{processes\text{-}ready} \neq \{\} \Rightarrow processes\text{-}current \neq \{\}$

annotations The *select* operation chooses which of the eligible processes shall run. More than one process may be chosen to run if the implementation provides multi-processing, for example using a time-sliced scheduler.

connect ($E : Event\text{-}source, P : Pid$)
ext wr $processes\text{-}attached : Event\text{-}source \xrightarrow{m} Pid$
pre *true*
post $processes\text{-}attached = \overleftarrow{processes\text{-}attached} \dagger \{E \mapsto P\}$

annotations Associate a process with a source of events. If another process was associated with that source of events it is not associated afterwards.

disconnect ($E : Event\text{-}source$)
ext wr $processes\text{-}attached : Event\text{-}source \xrightarrow{m} Pid$
pre *true*
post $processes\text{-}attached = \{E\} \lhd \overleftarrow{processes\text{-}attached}$

annotations Disassociate a source of events from a process. After the operation, no process will be associated with the source of events.

make-waiting ($P : Pid$)
ext wr $processes\text{-}current, processes\text{-}waiting : Pid$-set
pre $P \in processes\text{-}current \wedge$
$P \in \text{rng } processes\text{-}attached$
post $processes\text{-}current = \overleftarrow{processes\text{-}current} - \{P\} \wedge processes\text{-}waiting = \overleftarrow{processes\text{-}waiting} \cup \{P\}$

annotations Make a process ineligible to be selected to run. The process must be associated with at least one source of events.

EVENT-occurs ($E : Event\text{-}source$)
ext wr $processes\text{-}waiting, processes\text{-}ready, processes\text{-}current : Pid$-set
rd $processes\text{-}attached : Event\text{-}source \xrightarrow{m} Pid$
pre $E \in \text{dom } processes\text{-}attached \wedge processes\text{-}attached(E) \in processes\text{-}waiting$

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

post let $P = processes\text{-}attached(E)$ in
$post\text{-}make\text{-}ready(P) \wedge post\text{-}select$

annotations The *EVENT-occurs* primitive signals the occurrence of an event. After the event, the state of the process will be *ready* or *current*.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.3.2 The Module Semaphores

The module **Semaphores** provides facilities for potentially parallel parts of a program (i.e. processes) to exclude one another from regions of interaction by using the semaphore mechanism first proposed by Dijkstra. The semaphores provided by the module are *general* or *counting* semaphores (as opposed to *binary* semaphores).

9.3.2.1 The Definition Module of Semaphores

```
DEFINITION MODULE Semaphores;

  (* Provides mutual exclusion facilities for use by processes. *)

TYPE
  SEMAPHORE;

PROCEDURE Create (VAR s: SEMAPHORE; initialCount: CARDINAL );
  (* Creates and returns s as the identity of a new semaphore that has its associated count
     initialized to initialCount, and has no processes yet waiting on it, or raises an
     exception if a new semaphore cannot be created.
  *)

PROCEDURE Destroy (VAR s: SEMAPHORE);
  (* Provided that no process is waiting for s to become free, recovers the resources used
     to implement the semaphore s.
  *)

PROCEDURE Claim (s: SEMAPHORE);
  (* If the count associated with the semaphore s is non-zero, decrements this count and
     allows the calling process to continue; otherwise suspends the calling process until
     s is released.
  *)

PROCEDURE Release (s: SEMAPHORE);
  (* If there are any processes waiting on the semaphore s, allows one of them
     to enter the ready state; otherwise increments the count associated with s.
  *)

PROCEDURE CondClaim (s: SEMAPHORE): BOOLEAN;
  (* Returns FALSE if the call Claim(s) would cause the calling process to be suspended;
     in this case the count associated with s is not changed. Otherwise returns TRUE and
     the associated count is decremented.
  *)

PROCEDURE IsSemaphoresException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional execution state because
     of the raising of the Semaphores exception; otherwise returns FALSE.
  *)

END Semaphores.
```

9.3.2.2 The Procedures of Semaphores

Each semaphore shall have a unique identity. Associated with each semaphore shall be a non-negative count, and a set of zero or more processes waiting for it to become free. A semaphore is said to be 'free' if its associated count is non-zero.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NOTES

1 For convenience, the semantics of this module are illustrated in terms of the semantics of the `Processes` module. However, there is no requirement that an implementation provide `Processes` in addition to providing `Semaphores`.

2 The behaviour of the procedures `Destroy`, `Claim`, `Release` and `CondClaim` in situations where the actual parameter passed is not a valid semaphore (i.e. is not the identity of a semaphore allocated by a call to `Create`) is implementation-dependent.

The procedure `Create`

The call `Create(s,initialCount)` shall create a new semaphore, if there are sufficient resources to do so, and shall assign its identity to `s`; otherwise the `Semaphores` exception shall be raised. The count associated with the semaphore shall be initialized to `initialCount`, and there shall then be no process waiting for the semaphore to be free.

Semaphores-Create : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

Semaphores-Create (*args*)*handler* $\triangleq$
  def *cancreate* = *can-create-sema*() in
  if *cancreate*
  then let [*s*, *initialcount*] = *args* in
    def *sid* = *createsem*(*initialcount*) in
    assign(*s*, *sid*) *handler*
  else *semaphores-exception*() *handler*

createsem (*i* : N) *s* : *Valid-sid*
ext wr *semaphores-count* : *Valid-sid* $\xrightarrow{m}$ N
  wr *semaphores-waiters* : *Valid-sid* $\xrightarrow{m}$ *Pid*-set
post let *s* : *Valid-sid* be st $s \notin \text{dom } \overleftarrow{semaphores\text{-}count}$ in
  $semaphores\text{-}count = \overleftarrow{semaphores\text{-}count} \dagger \{s \mapsto i\} \wedge semaphores\text{-}waiters = \overleftarrow{semaphores\text{-}waiters} \dagger \{s \mapsto \{\}\}$

annotations The first line of the post-condition guarantees that the identity of each semaphore is different from all the others that exist *at this time*. No requirement is imposed concerning the re-use of the identity of a semaphore that has been destroyed.

can-create-sema : () $\xrightarrow{o}$ $\mathbb{B}$

can-create-sema () $\triangleq$
  return implementation-dependent value

annotations Return *true* if there are sufficient resources to create a new semaphore; otherwise return *false*.

The procedure `Destroy`

Provided that no process is waiting for the semaphore identified by the value of `s` to become free, the call `Destroy(s)` shall remove that semaphore and shall recover the resources used to implement it.

The variable `s` shall be set to a value that is invalid for semaphore operations.

NOTE 3 — The behaviour of `Destroy(s)` in situations where there are processes waiting for `s` to become free is implementation-dependent.

Destroy : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

Destroy (*args*)*handler* $\triangleq$
  let [*s*] = *args* in
  def *sem* = *value-of-a-variable*(*s*) in
  def *sid* = *destroysem*(*sem*) in
  assign(*s*, *sid*) *handler*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

destroysem (*s* : *Valid-sid*) *ns* : *Sid*
ext wr *semaphores-count* : *Valid-sid* $\xrightarrow{m}$ N
wr *semaphores-waiters* : *Valid-sid* $\xrightarrow{m}$ *Pid*-set
pre $s \in \text{dom } semaphores\text{-}waiters \wedge semaphores\text{-}waiters(s) = \{\}$
post $(semaphores\text{-}waiters = s \lhd \overleftarrow{semaphores\text{-}waiters}) \wedge (semaphores\text{-}count = s \lhd \overleftarrow{semaphores\text{-}count}) \wedge (ns = \text{INVALID-SID})$

The procedure `Claim`

The call `Claim(s)` shall claim the semaphore identified by `s`. If the count associated with `s` is non-zero, then it shall be decremented, and the calling process shall continue execution; otherwise the calling process shall become ineligible to run, and shall be added to the set waiting for `s` to become free.

Claim : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()
Claim (*args*)*handler* $\triangleq$
let $[s] = args$ in
def *count* = *semaphores-count* in
if *count* $(s) > 0$
then *decrease-count*(s) 563
else *make-me-wait-for*(s) 563
pre $s \in \text{dom } semaphores\text{-}waiters$

The procedure `Release`

The call `Release(s)` shall unclaim the semaphore identified by `s`. If no process is waiting on `s`, the count associated with `s` shall be incremented; otherwise one process shall be selected from those waiting for `s` to become free; this process shall be removed from the waiting set, and shall become eligible to run (i.e. enter the *ready* state).

NOTES

4 No requirement is imposed about *which* of the waiting processes is chosen for activation.

5 Preemption will occur if the newly eligible process has an urgency greater than that of the calling process.

Release : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()
Release (*args*)*handler* $\triangleq$
let $[s] = args$ in
def *waiters* = *semaphores-waiters* in
if *waiters* $(s) \neq \{\}$
then *make-one-process-active-in*(s) 563
else *increase-count*(s) 563
pre $s \in \text{dom } semaphores\text{-}waiters$

The procedure `CondClaim`

If the call `Claim(s)` would have caused the calling process to become ineligible to run, the value of the call `CondClaim(s)` shall be *false*, and the count associated with `s` shall be unchanged. Otherwise the count associated with `s` shall be decremented, and the value of the call shall be *true*.

NOTE 6 — The calling process is never suspended as a result of calling `CondClaim`.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

CondClaim : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

CondClaim (*args*)*handler* $\triangleq$
  let [*s*] = *args* in
  def *count* = *semaphores-count* in
  if *count* (*s*) > 0
  then (*decrease-count*(*s*) ; return true) 563
  else return false
pre *s* ∈ dom *semaphores-waiters*

The procedure `IsSemaphoresException`

If the calling coroutine is in the state of exceptional execution because of the raising of the `Semaphores` exception, the value of the call `IsSemaphoresException()` shall be *true*; otherwise the value shall be *false*.

IsSemaphoresException : *Arguments* → *Exception-handler* $\xrightarrow{o}$ $\mathbb{B}$

IsSemaphoresException (*args*)*handler* $\triangleq$
  def *source* = *semaphores-source* in
  def *res* = *m-IsCurrentSource*([*source*]) in 386
  return *res*

9.3.2.3 Exceptions Raised by Semaphores

Following the detection of the exception defined for this module, the `Semaphores` exception shall be raised.

The character string associated with the `Semaphores` exception shall be implementation-defined.

semaphores-exception : () → *Exception-handler* $\xrightarrow{o}$ ()

semaphores-exception ()*handler* $\triangleq$
  def *source* = *semaphores-source* in
  def *message* = *semaphores-exception-string*() in 562
  let *number* : N in
  m-RAISE([*source*, *number*, *message*]) *handler* 383

semaphores-exception-string : () $\xrightarrow{o}$ *Message*

semaphores-exception-string () $\triangleq$
  implementation-defined

annotations The result is an implementation-defined string value.

9.3.2.4 The State for Semaphores

Domains

types

Sid = INVALID-SID | *Valid-sid*;

Valid-sid = token$_{semaphore\text{-}id}$

state *Semaphores* of
  semaphores-count : *Valid-sid* $\xrightarrow{m}$ N
  semaphores-waiters : *Valid-sid* $\xrightarrow{m}$ *Pid*-set
  semaphores-source : *Exception-source*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

inv *mk-Semaphores* (*Count*, *Waiters*) $\triangleq$
(dom *Count* = dom *Waiters*) $\wedge$ ($\forall s \in$ dom *Count* $\cdot$ *Waiters*(*s*) $\neq \{\}$ $\Rightarrow$ *Count*(*s*) = 0)
end

9.3.2.5 The Initialization of Semaphores

The initialization of the module **Semaphores**, shall allocate a unique value of the exception source type to identify the **Semaphores** exception.

After the module has been initialized, no semaphores shall be in existence.

semaphores-initialization : () $\rightarrow$ *Exception-handler* $\xrightarrow{o}$ ()
semaphores-initialization ()*handler* $\triangleq$
|| (def *source* = *allocate-source*() in 342
if *source* $\neq$ nil
then *semaphores-source* := *source*
else *mandatory-detection*(NO-EXCEPTION-SOURCE) *handler* , 396
semaphores-waiters := {$\mapsto$},
semaphores-count := {$\mapsto$})

9.3.2.6 Auxiliary Formal Model Definitions

NOTE — This definition refers to functions defined in 9.3.1.6. The functions concerned are *make-passive*, *make-ready*, *select*, and *whoami*.

increase-count (*s* : *Sid*)
ext wr *semaphores-count* : *Valid-sid* $\xrightarrow{m}$ N
pre *s* $\in$ dom *semaphores-count*
post *semaphores-count*(*s*) = $\overleftarrow{\textit{semaphores-count}}(s) + 1$

decrease-count (*s* : *Sid*)
ext wr *semaphores-count* : *Valid-sid* $\xrightarrow{m}$ N
pre *s* $\in$ dom *semaphores-count*
post *semaphores-count*(*s*) = $\overleftarrow{\textit{semaphores-count}}(s) - 1$

make-me-wait-for (*s* : *Sid*)
ext wr *semaphores-waiters* : *Valid-sid* $\xrightarrow{m}$ *Pid*-set
pre *s* $\in$ dom *semaphores-waiters*
post def *me* = *whoami*() in 557
semaphores-waiters(*s*) = $\overleftarrow{\textit{semaphores-waiters}}(s) \cup \{me\} \wedge$ *post-make-passive*(*me*) $\wedge$ *post-select*

make-one-process-active-in (*s* : *Sid*)
ext wr *semaphores-waiters* : *Valis-sid* $\xrightarrow{m}$ *Pid*-set
pre *s* $\in$ dom *semaphores-waiters*
post let *p* $\in$ $\overleftarrow{\textit{semaphores-waiters}}(s)$ in
semaphores-waiters(*s*) = $\overleftarrow{\textit{waiters}}(s) - \{p\} \wedge$ *post-make-ready*(*p*) $\wedge$ *post-select*

annotations One process is chosen from the set of processes that are waiting for the semaphore, and is made eligible to run.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9.4 The Module Strings

The module `Strings` provides facilities for manipulating character arrays as representations of strings. The procedures provided accept any character array type, but manipulate all as if their index types were whole numbers and zero-based.

The module also provides predicates that check whether an operation to assign, delete, insert, replace or append strings or characters will work without loss of information. These predicates check that parameters indexing the concrete representation of a string (i.e. the character array containing the string value) fall within its length, thereby allowing the programmer to maintain the string abstraction.

A general-purpose string type (`String1`) is provided for convenience when handling single characters by using a value constructor. An enumeration type (`CompareResults`) is provided for use when comparing string values.

9.4.1 The Definition Module of Strings

```
DEFINITION MODULE Strings;

  (* Facilities for manipulating strings *)

TYPE
  String1 = ARRAY [0..0] OF CHAR;
    (* String1 is provided for constructing a value of a single-character string type from a
       single character value in order to pass CHAR values to ARRAY OF CHAR parameters.
    *)

PROCEDURE Length (stringVal: ARRAY OF CHAR): CARDINAL;
  (* Returns the length of stringVal (the same value as would be returned by the
     standard function LENGTH).
  *)


(* The following seven procedures construct a string value, and attempt to assign it to a
   variable parameter.  They all have the property that if the length of the constructed string
   value exceeds the capacity of the variable parameter, a truncated value is assigned, while
   if the length of the constructed string value is less than the capacity of the variable
   parameter, a string terminator is appended before assignment is performed.
*)

PROCEDURE Assign (source: ARRAY OF CHAR; VAR destination: ARRAY OF CHAR);
  (* Copies source to destination *)

PROCEDURE Extract (source: ARRAY OF CHAR; startPos, numberToExtract: CARDINAL;
                   VAR destination: ARRAY OF CHAR);
  (* Copies at most numberToExtract characters from source to destination, starting at position
     startPos in source.
  *)

PROCEDURE Delete (VAR stringVar: ARRAY OF CHAR; startPos, numberToDelete: CARDINAL);
  (* Deletes at most numberToDelete characters from stringVar, starting at position
     startPos.
  *)

PROCEDURE Insert (source: ARRAY OF CHAR; startPos: CARDINAL;
                  VAR destination: ARRAY OF CHAR);
  (* Inserts source into destination at position startPos *)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
PROCEDURE Replace (source: ARRAY OF CHAR; startPos: CARDINAL;
                   VAR destination: ARRAY OF CHAR);
  (* Copies source into destination, starting at position startPos. Copying stops when
     all of source has been copied, or when the last character of the string value in
     destination has been replaced.
  *)

PROCEDURE Append (source: ARRAY OF CHAR; VAR destination: ARRAY OF CHAR);
  (* Appends source to destination. *)

PROCEDURE Concat (source1, source2: ARRAY OF CHAR; VAR destination: ARRAY OF CHAR);
  (* Concatenates source2 onto source1 and copies the result into destination. *)

(* The following predicates provide for pre-testing of the operation-completion
   conditions for the procedures above.
*)

PROCEDURE CanAssignAll (sourceLength: CARDINAL; VAR destination: ARRAY OF CHAR): BOOLEAN;
  (* Returns TRUE if a number of characters, indicated by sourceLength, will fit into
     destination; otherwise returns FALSE.
  *)

PROCEDURE CanExtractAll (sourceLength, startPos, numberToExtract: CARDINAL;
                         VAR destination: ARRAY OF CHAR): BOOLEAN;
  (* Returns TRUE if there are numberToExtract characters starting at startPos and
     within the sourceLength of some string, and if the capacity of destination is
     sufficient to hold numberToExtract characters; otherwise returns FALSE.
  *)

PROCEDURE CanDeleteAll (stringLength, startPos, numberToDelete: CARDINAL): BOOLEAN;
  (* Returns TRUE if there are numberToDelete characters starting at startPos and
     within the stringLength of some string; otherwise returns FALSE.
  *)

PROCEDURE CanInsertAll (sourceLength, startPos: CARDINAL;
                        VAR destination: ARRAY OF CHAR): BOOLEAN;
  (* Returns TRUE if there is room for the insertion of sourceLength characters from
     some string into destination starting at startPos; otherwise returns FALSE.
  *)

PROCEDURE CanReplaceAll (sourceLength, startPos: CARDINAL;
                         VAR destination: ARRAY OF CHAR): BOOLEAN;
  (* Returns TRUE if there is room for the replacement of sourceLength characters in
     destination starting at startPos; otherwise returns FALSE.
  *)

PROCEDURE CanAppendAll (sourceLength: CARDINAL; VAR destination: ARRAY OF CHAR): BOOLEAN;
  (* Returns TRUE if there is sufficient room in destination to append a string of
     length sourceLength to the string in destination; otherwise returns FALSE.
  *)

PROCEDURE CanConcatAll (source1Length, source2Length: CARDINAL;
                        VAR destination: ARRAY OF CHAR): BOOLEAN;
  (* Returns TRUE if there is sufficient room in destination for a two strings of
     lengths source1Length and source2Length; otherwise returns FALSE.
  *)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
(* The following type and procedures provide for the comparison of string values, and for the
   location of substrings within strings.
*)

TYPE
  CompareResults = (less, equal, greater);

PROCEDURE Compare (stringVal1, stringVal2: ARRAY OF CHAR): CompareResults;
  (* Returns less, equal, or greater, according as stringVal1 is lexically less than,
     equal to, or greater than stringVal2.
  *)

PROCEDURE Equal (stringVal1, stringVal2: ARRAY OF CHAR): BOOLEAN;
  (* Returns Strings.Compare(stringVal1, stringVal2) = Strings.equal *)

PROCEDURE FindNext (pattern, stringToSearch: ARRAY OF CHAR; startPos: CARDINAL;
                    VAR patternFound: BOOLEAN; VAR posOfPattern: CARDINAL);
  (* Looks forward for next occurrence of pattern in stringToSearch, starting the search at
     position startPos. If startPos < LENGTH(stringToSearch) and pattern is found,
     patternFound is returned as TRUE, and posOfPattern contains the start position in
     stringToSearch of pattern. Otherwise patternFound is returned as FALSE, and posOfPattern
     is unchanged.
  *)

PROCEDURE FindPrev (pattern, stringToSearch: ARRAY OF CHAR; startPos: CARDINAL;
                    VAR patternFound: BOOLEAN; VAR posOfPattern: CARDINAL);
  (* Looks backward for the previous occurrence of pattern in stringToSearch and returns the
     position of the first character of the pattern if found. The search for the pattern
     begins at startPos. If pattern is found, patternFound is returned as TRUE, and
     posOfPattern contains the start position in stringToSearch of pattern in the range
     [0..startPos]. Otherwise patternFound is returned as FALSE, and posOfPattern is unchanged.
  *)

PROCEDURE FindDiff (stringVal1, stringVal2: ARRAY OF CHAR;
                    VAR differenceFound: BOOLEAN; VAR posOfDifference: CARDINAL);
  (* Compares the string values in stringVal1 and stringVal2 for differences. If they
     are equal, differenceFound is returned as FALSE, and TRUE otherwise. If
     differenceFound is TRUE, posOfDifference is set to the position of the first
     difference; otherwise posOfDifference is unchanged.
  *)

PROCEDURE Capitalize (VAR stringVar: ARRAY OF CHAR);
  (* Applies the function CAP to each character of the string value in stringVar. *)

END Strings.
```

NOTES

1 The procedures provided accept any character array type (i.e. with any index type) but because of the use of open array parameters the procedures manipulate all as if their index types were whole numbers and zero-based.

2 The array type String1 may be used as the array type identifier of an array constructor when constructing an array value from a value of the character type. The constructed array value may be used as actual parameter in calls of procedures having a formal open array value parameter, for example to assign, insert, replace, append, concatenate, or find a single character.

3 Since function procedures cannot return open arrays, many of the operations that might more logically have been provided as function procedures have to be provided as proper procedures.

4 Predicates with the prefix 'Can' and the suffix 'All' are provided to check the operation-completion condition of string

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

operations (e.g. `CanInsertAll` checks the operation-completion condition for `Insert`). Failure to satisfy the operation-completion condition of a string handling procedure does not lead to an exception; i.e. the semantics of string operations are defined for all well-formed parameters.

5 Value parameters are used where a string value is not changed by a procedure. This is not just a matter of programming clarity, but allows the parameterization of programs using constants.

6 Because string constants cannot be assigned to, nor appended to, nor have characters replaced or inserted, the predicates that test the operation completion conditions of procedures use `VAR`-parameters for the string parameters.

7 This part of ISO/IEC 10514 has not adopted a change to Modula-2 (described in the fourth edition of *Programming in Modula-2*) of always requiring a string terminator for a string value.

9.4.2 The Procedures of Strings

The procedure `Length`

The value of the call

```
Length(stringVal)
```

shall be the length of the string value in `stringVal`. This shall be the same as the value of `LENGTH(stringVal)` — see 6.10.4.10.

$Length : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} \mathbb{N}$

$Length\,(args)handler \triangleq$
  let $[stringVal] = args$ in
  return len $pure\text{-}string\,(stringVal)$

The procedure `CanAssignAll`

The value of the call

```
CanAssignAll(sourceLength, destination)
```

shall be the value of the Boolean expression

```
sourceLength <= HIGH(destination) + 1
```

NOTES

1 The function procedure `CanAssignAll` may be used to check whether complete assignment of a string value to a string variable will be possible using, for example, the procedure `Assign`.

2 Since a string variable must have at least one element, single-character string assignment is always valid.

$CanAssignAll : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} \mathbb{B}$

$CanAssignAll\,(args)handler \triangleq$
  let $[sourceLength, destination] = args$ in
  let $cap = capacity\,(destination)$ in
  return $sourceLength \leq cap$

The procedure `Assign`

The call

```
Assign(source, destination)
```

shall assign the string value in `source` to `destination`. If the length of `source` exceeds the capacity of `destination`,

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

the assigned value shall be truncated to the capacity of `destination`. If the length of `source` is less than the capacity of `destination`, a string terminator shall be appended to `source` when assignment takes place.

$Assign : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} ()$

$Assign\,(args)handler \triangleq$
  let $[source, destination] = args$ in
  $array\text{-}variable\text{-}assignment(destination, source)\ handler$ 150

EXAMPLE 1 — String assignment.

In these, and later, examples the following declarations are assumed:

```
VAR
  small: ARRAY [0 .. 4] OF CHAR;
  large: ARRAY [0 .. 255] OF CHAR;
  alpha: ARRAY ['A' .. 'E'] OF CHAR;
  ch: CHAR;
  found, areDiff: BOOLEAN;
  pos: CARDINAL;
```

a)
```
ch := "X";
Assign(String1{ch}, small)
```
results in:
`small[0]` having value "X"
`small[1]` having value ""

b)
```
Assign("pq", small)
```
results in:
`small[0]` having value "p"
`small[1]` having value "q"
`small[2]` having value ""

c)
```
Assign("", small)
```
results in:
`small[0]` having value ""

d)
```
Assign("Hello!", small)
```
results in:
`small[0]` having value "H"
`small[1]` having value "e"
`small[2]` having value "l"
`small[3]` having value "l"
`small[4]` having value "o"

e) the call
```
CanAssignAll(6, small)
```
returns the value:
`FALSE`

f)
```
Assign("Go", alpha)
```
results in:
`alpha['A']` having value "G"
`alpha['B']` having value "o"
`alpha['C']` having value ""

g)
```
small := "Hello"; large := "";
IF CanAssignAll(LENGTH(small), large)
THEN
  Assign(small, large)
END
```
results in:
`large[0]` having value "H"
`large[1]` having value "e"
`large[2]` having value "l"
`large[3]` having value "l"
`large[4]` having value "o"
`large[5]` having value ""

The procedure `CanExtractAll`

The value of the call

```
CanExtractAll(sourceLength, startPos, numberToExtract, destination)
```

shall be the value of the Boolean expression

```
(startPos + numberToExtract <= sourceLength) AND (numberToExtract <= HIGH(destination) + 1)
```

NOTE 3 — The function procedure `CanExtractAll` may be used to check whether complete extraction of a substring from a string variable will be possible using, for example, the procedure `Extract`.

$CanExtractAll : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} \mathbb{B}$

$CanExtractAll\,(args)handler \triangleq$
  let $[sourceLength, startPos, numberToExtract, destination] = args$ in
  let $cap = capacity\,(destination)$ in
  return $startPos + numberToExtract \leq sourceLength \wedge cap \geq numberToExtract$ 579

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The procedure `Extract`

The call

```
Extract(source, startPos, numberToExtract, destination)
```

shall create a new string value of at most `numberToExtract` characters extracted from `source`. Extraction shall start at position `startPos` in `source`, and shall continue as long as there are characters left to extract from `source` and no more than `numberToExtract` characters have been extracted. If the length of the created string value exceeds the capacity of `destination`, the string value shall be truncated to the capacity of `destination`, and the truncated value shall be assigned to `destination`. If the length of the created string value is less than the capacity of `destination`, a string terminator shall be appended to the string value, and the resulting value shall be assigned to `destination`. An empty string value shall be extracted if `startPos` is greater than or equal to `LENGTH(source)`.

$Extract : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} ()$

$Extract\,(args)handler \triangleq$
  let $[source, startPos, numberToExtract, destination] = args$ in
  let $repstr = pure\text{-}string\,(source)$ in 579
  let $substr = subseq\,(repstr, startPos + 1, startPos + numberToExtract)$ in 579
  $array\text{-}variable\text{-}assignment(destination, substr)\ handler$ 150

annotations The index values in the procedure calls range from zero upwards; the corresponding index values for the VDM-SL character sequences range from 1 upwards. This is most apparent in evaluations of the *subseq* function.

EXAMPLE 2 — String extraction.

a)
```
large := "ABCDE"; small := "";
IF CanExtractAll(LENGTH (large), 2, 3, small) THEN
  Extract(large, 2, 3, small)
END
```
results in:
`small[0]` having value `"C"`
`small[1]` having value `"D"`
`small[2]` having value `"E"`
`small[3]` having value `""`

b)
```
large := "ABCDE"; small := "";
Extract(large, 2, 3, small)
```
results in:
`small[0]` having value `"C"`
`small[1]` having value `"D"`
`small[2]` having value `"E"`
`small[3]` having value `""`

The procedure `CanDeleteAll`

The value of the call

```
CanDeleteAll(stringLength, startPos, numberToDelete)
```

shall be the value of the Boolean expression

```
startPos + numberToDelete <= stringLength
```

NOTE 4 — The function procedure `CanDeleteAll` may be used to check whether complete deletion of a substring value from a string variable will be possible using, for example, using the procedure `Delete`.

$CanDeleteAll : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} \mathbb{B}$

$CanDeleteAll\,(args)handler \triangleq$
  let $[stringLength, startPos, numberToDelete] = args$ in
  return $startPos + numberToDelete \leq stringLength$

The procedure `Delete`

The call

```
Delete(stringVar, startPos, numberToDelete)
```

shall create a new string value by deleting at most `numberToDelete` characters from `stringVar`. Deletion shall start

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

at position `startPos` in `stringVar` and shall continue as long as there are characters left to delete in `stringVar` and no more than `numberToDelete` characters have been deleted. If any characters are deleted, a string terminator shall be appended to the created string value, and the resulting value shall be assigned to `stringVar`. The string value in `stringVar` shall not be altered if `startPos` is greater than or equal to `LENGTH(stringVar)`.

Delete : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

Delete (*args*)*handler* $\triangleq$
  let [*stringVar*, *startPos*, *numberToDelete*] = *args* in
  def *str* = *obtain-string*(*stringVar*) in 579
  let *part1* = *subseq* (*str*, 1, *startPos*) in 579
  let *part2* = *subseq* (*str*, *startPos* + *numberToDelete* + 1, len *str*) in 579
  let *newstr* = *part1* ⌢ *part2* in
  array-variable-assignment(*stringVar*, *newstr*) *handler* 150

EXAMPLE 3 — String deletion.

a)
```
small := "ABCDE";
IF CanDeleteAll(LENGTH(small), 2, 2)
THEN
  Delete(small, 2, 2)
END
```
results in:
`small[0]` having value "A"
`small[1]` having value "B"
`small[2]` having value "E"
`small[3]` having value ""

b) after the assignment
```
small := "ABC";
```
the call
```
CanDeleteAll(3, 2, 2)
```
returns the value:
```
FALSE
```

c)
```
small := "ABC";
Delete(small, 2, 2)
```
results in:
`small[0]` having value "A"
`small[1]` having value "B"
`small[2]` having value ""

The procedure `CanInsertAll`

The value of the call

```
CanInsertAll(sourceLength, startPos, destination)
```

shall be the value of the Boolean expression

```
(startPos <= LENGTH(destination)) AND
(sourceLength + LENGTH(destination) <= HIGH(destination) + 1)
```

NOTE 5 — The function procedure `CanInsertAll` may be used to check whether complete insertion of a string value into a string variable will be possible using, for example, the procedure `Insert`.

CanInsertAll : *Arguments* → *Exception-handler* $\xrightarrow{o}$ $\mathbb{B}$

CanInsertAll (*args*)*handler* $\triangleq$
  let [*sourceLength*, *startPos*, *destination*] = *args* in
  def *str* = *obtain-string*(*destination*) in 579
  let *cap* = *capacity* (*destination*) in 579
  return *startPos* ≤ len *str* ∧ *sourceLength* + len *str* ≤ *cap*

The procedure `Insert`

The call

```
Insert(source, startPos, destination)
```

shall create a new string value by inserting the substring `source` into `destination`. The string in `destination` shall first be split at the position given by `startPos`; the created string value shall then be the concatenation of the first part of `destination`, the substring `source`, and the second part of `destination`. If the length of the created string value exceeds the capacity of `destination`, the string value shall be truncated to the capacity of `destination`, and the truncated value shall be assigned to `destination`. If the length of the created string value is less than the capacity of `destination`, a string terminator shall be appended to the string value, and the resulting value shall be

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

assigned to `destination`. The string value in `destination` shall not be altered if `startPos` is greater than or equal to `LENGTH(destination)`.

Insert : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

Insert (*args*)*handler* ≙
  let [*source*, *startPos*, *destination*] = *args* in
  def *oldstr* = *obtain-string*(*destination*) in 579
  let *insstr* = *pure-string* (*source*) in 579
  let *part1* = *subseq* (*oldstr*, 1, *startPos*) in 579
  let *part2* = *subseq* (*oldstr*, *startPos* + 1, len *oldstr*) in 579
  let *newstr* = *part1* ⌢ *insstr* ⌢ *part2* in
  array-variable-assignment(*destination*, *newstr*) *handler* 150

annotations Obtain abstract string values from source and destination; split destination at start index; concatenate beginning of destination, source, and end of destination, and turn result back into concrete string value. The function *array-variable-assignment* truncates if required.

EXAMPLE 4 — String insertion.

a) after the assignment
```
small := "ABCD"
```
the call
```
CanInsertAll(LENGTH("XYZ"), 2, small)
```
returns the value:
```
FALSE
```

b)
```
small := "ABCD"
Insert("XYZ", 2, small)
```
results in:
```
small[0] having value "A"
small[1] having value "B"
small[2] having value "X"
small[3] having value "Y"
small[4] having value "Z"
```

c)
```
large := "ABC";
IF CanInsertAll(3, 2, large) THEN
  Insert("XYZ", 2, large)
END
```
results in:
```
large[0] having value "A"
large[1] having value "B"
large[2] having value "X"
large[3] having value "Y"
large[4] having value "Z"
large[5] having value "C"
large[6] having value ""
```

d)
```
large := "ABCD"; ch := "X"
Insert(String1{ch}, 2, large)
```
results in:
```
large[0] having value "A"
large[1] having value "B"
large[2] having value "X"
large[3] having value "C"
large[4] having value "D"
large[5] having value ""
```

The procedure `CanReplaceAll`

The value of the call

```
CanReplaceAll(sourceLength, startPos, destination)
```

shall be the value of the Boolean expression

```
sourceLength + startPos <= LENGTH(destination)
```

NOTES

6 The function procedure `CanReplaceAll` may be used to check whether complete replacement of a substring value within a string variable will be possible using, for example, the procedure `Replace`.

7 The preservation of the string abstraction is taken as the goal of the string module. This means that the operation completion condition of `CanReplaceAll` only tests whether the proposed replacement is valid within the given string length; the procedure `Replace` will always preserve the length of its destination string.

CanReplaceAll : *Arguments* → *Exception-handler* $\xrightarrow{o}$ $\mathbb{B}$

CanReplaceAll (*args*)*handler* ≙
  let [*sourceLength*, *startPos*, *destination*] = *args* in
  def *str* = *obtain-string*(*destination*) in 579

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

return *startPos* + *sourceLength* $\leq$ len *str*

The procedure Replace

The call

```
Replace(source, startPos, destination)
```

shall modify the string value from `destination` by overwriting characters in `destination` with characters extracted from the string value in `source`. Overwriting shall begin at the position given by `startPos` and shall continue as long as there are characters left to overwrite in `destination` and characters left to extract from `source`. The string value in `destination` shall not be altered if `startPos` is greater than or equal to `LENGTH(destination)`.

NOTE 8 — The length of the string value in `destination` is always preserved by `Replace`.

Replace : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

Replace (*args*)*handler* $\triangleq$
  let [*source*, *startPos*, *destination*] = *args* in
  def *oldstr* = *obtain-string*(*destination*) in 579
  let *repstr* = *pure-string* (*source*) in 579
  let *lenrep* = len *repstr* in
  let *lenold* = len *oldstr* in
  let *part1* = *subseq* (*oldstr*, 1, *startPos*) in 579
  let *part2* = *subseq* (*oldstr*, *startPos* + *lenrep* + 1, *lenold*) in 579
  let *truncstr* = *subseq* (*part1* $\curvearrowright$ *repstr* $\curvearrowright$ *part2*, 1, *lenold*) in 579
  array-variable-assignment(*destination*, *truncstr*) *handler* 150

EXAMPLE 5 — String replacement.

a) after the assignment
```
small := "ABC"
```
the call
```
CanReplaceAll(LENGTH("XY"), 2, small)
```
returns the value:
```
FALSE
```

b)
```
small := "ABC";
Replace("XY", 2, small)
```
results in:
```
small[0] having value "A"
small[1] having value "B"
small[2] having value "X"
small[3] having value ""
```

c)
```
large := "ABCDEF";
IF CanReplaceAll(3, 2, large)
THEN
  Replace("XYZ", 2, large)
END
```
results in:
```
large[0] having value "A"      large[4] having value "Z"
large[1] having value "B"      large[5] having value "F"
large[2] having value "X"      large[6] having value ""
large[3] having value "Y"
```

The procedure CanAppendAll

The value of the call

```
CanAppendAll(sourceLength, destination)
```

shall be the value of the Boolean expression

```
LENGTH(destination) + sourceLength <= HIGH(destination) + 1
```

NOTE 9 — The function procedure `CanAppendAll` may be used to check whether it will be possible to append a string value to another string value held within a string variable using, for example, the procedure `Append`.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

CanAppendAll : *Arguments* → *Exception-handler* → 𝔹

CanAppendAll (*args*)*handler* ≙
  let [*sourceLength*, *destination*] = *args* in
  def *str* = *obtain-string*(*destination*) in 579
  let *cap* = *capacity* (*destination*) in 579
  return len *str* + *sourceLength* ≤ *cap*

The procedure Append

The call

```
Append(source, destination)
```

shall create a new string value by appending the string value in `source` onto `destination`. If the length of the created string value exceeds the capacity of `destination`, the string value shall be truncated to the capacity of `destination`, and the truncated value shall be assigned to `destination`. If the length of the created string value is less than the capacity of `destination`, a string terminator shall be appended to the string value, and the resulting value shall be assigned to `destination`.

Append : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

Append (*args*)*handler* ≙
  let [*source*, *destination*] = *args* in
  def *oldstr* = *obtain-string*(*destination*) in 579
  let *appstr* = *pure-string* (*source*) in 579
  let *newstr* = *oldstr* $\curvearrowright$ *appstr* in
  array-variable-assignment(*destination*, *newstr*) *handler* 150

EXAMPLE 6 — Appending to strings.

a) after the assignment
```
small := "pqr"
```
the call
```
CanAppendAll(LENGTH("XYZ"), small)
```
returns the value:
```
FALSE
```

b)
```
small := "pqr";
Append("XYZ", small)
```
results in:
```
small[0] having value "p"
small[1] having value "q"
small[2] having value "r"
small[3] having value "X"
small[4] having value "Y"
```

c)
```
small := "pqr";
ch := "s";
Append(String1{ch}, small)
```
results in:
```
small[0] having value "p"     small[3] having value "s"
small[1] having value "q"     small[4] having value ""
small[2] having value "r"
```

The procedure CanConcatAll

The value of the call

```
CanConcatAll(source1Length, source2Length, destination)
```

shall be the value of the Boolean expression

```
source1Length + source2Length <= HIGH(destination) + 1
```

NOTE 10 — The function procedure `CanConcatAll` may be used to check whether complete concatenation of two string values will be possible within the capacity of a specified string variable using, for example, the procedure `Concat`.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

CanConcatAll : *Arguments* → *Exception-handler* $\xrightarrow{o}$ $\mathbb{B}$

CanConcatAll (*args*)*handler* $\triangleq$
  let [*source1Length*, *source2Length*, *destination*] = *args* in
  let *cap* = *capacity* (*destination*) in 579
  return *source1Length* + *source2Length* ≤ *cap*

The procedure `Concat`

The call

```
Concat(source1, source, destination)
```

shall create a new string value by appending the substring value `source2` onto `source1`. If the length of the created string value exceeds the capacity of `destination`, the string value shall be truncated to the capacity of `destination`, and the truncated value shall be assigned to `destination`. If the length of the created string value is less than the capacity of `destination`, a string terminator shall be appended to the string value, and the resulting value shall be assigned to `destination`.

Concat : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

Concat (*args*)*handler* $\triangleq$
  let [*source1*, *source2*, *destination*] = *args* in
  let *str1* = *pure-string* (*source1*) in 579
  let *str2* = *pure-string* (*source2*) in 579
  let *newstr* = *str1* $\curvearrowright$ *str2* in
  array-variable-assignment(*destination*, *newstr*) *handler* 150

EXAMPLE 7 — String concatenation.

a) after the assignment
```
small := "pqr"
```
the call
```
CanConcatAll(4, LENGTH(small), small)
```
returns the value:
```
FALSE
```

b)
```
small := "pqr";
Concat("WXYZ", small, small)
```
results in:
```
small[0]  having value  "W"
small[1]  having value  "X"
small[2]  having value  "Y"
small[3]  having value  "Z"
small[4]  having value  "p"
```

c)
```
small := "jk"; large := "";
ch := "s";
Concat(String1{ch}, small, large)
```
results in:
```
large[0]  having value  "s"        large[2]  having value  "k"
large[1]  having value  "j"        large[3]  having value  ""
```

The procedure `Capitalize`

The call

```
Capitalize(stringVar)
```

shall apply the standard function `CAP` to each character of the string value in `stringVar`.

NOTE 11 — The procedure `Capitalize` may be used to achieve case-insensitive use of the procedures `Compare`, `FindNext`, `FindPrev` and `FindDiff`.

Capitalize : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

Capitalize (*args*)*handler* $\triangleq$
  let [*stringVar*] = *args* in
  def *original* = *obtain-string*(*stringVar*) in 579
  let *capped* = [*i* ↦ *cap* (*original*(*i*)) | *i* ∈ inds *original*] in 575
  array-variable-assignment(*stringVar*, *capped*) *handler* 150

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

$cap : Char \rightarrow Char$

$cap\,(ch) \triangleq$
  if $ch \in \mathsf{dom}\ capitalizations$
  then $capitalizations\,(ch)$
  else ch

EXAMPLE 8 — String capitalization.

The following example assumes a capitalization mapping which maps `p` to `P`, `q` to `Q` and `r` to `R`.

```
small := "pqr";
Capitalize(small)
results in:
small[0] having value "P"
small[1] having value "Q"
small[2] having value "R"
small[3] having value ""
```

The procedure `Compare`

The call

```
Compare(stringVal1, stringVal2)
```

shall return a value of the enumeration type `CompareResults` depending on the lexical ordering of the type `CHAR` — see 6.10.2.5. The value returned shall be `less`, `equal` or `greater` according as the string value in `stringVal1` is lexically less than, equal to, or greater than the string value in `stringVal2`.

NOTES

12 The result of this function is dependent upon the full collating sequence of character values. This sequence is implementation-defined, although it must have certain properties — see 6.10.2.5.

13 In general, the result of this function is case-sensitive.

$Compare : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} Value$

$Compare\,(args)handler \triangleq$
  let $[stringVal1, stringVal2] = args$ in
  let $s1 = pure\text{-}string\,(stringVal1)$ in 579
  let $s2 = pure\text{-}string\,(stringVal2)$ in 579
  return $(s1 = s2 \rightarrow equal,$
  $is\text{-}less\text{-}than\,(s1, s2) \rightarrow less,$
  $\mathsf{others} \rightarrow greater)$

$is\text{-}less\text{-}than : Char^* \times Char^* \rightarrow \mathbb{B}$

$is\text{-}less\text{-}than\,(s1, s2) \triangleq$
  $(s1 = [] \rightarrow s2 \neq [],$
  $s2 = [] \rightarrow \mathsf{false},$
  $\mathsf{hd}\ s1 < \mathsf{hd}\ s2 \rightarrow \mathsf{true},$
  $\mathsf{hd}\ s1 > \mathsf{hd}\ s2 \rightarrow \mathsf{false},$
  $\mathsf{hd}\ s1 = \mathsf{hd}\ s2 \rightarrow is\text{-}less\text{-}than\,(\mathsf{tl}\ s1, \mathsf{tl}\ s2))$ 575

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLE 9 — String comparison.

a) `Compare("", "") returns equal`
b) `Compare("", "abc") returns less`
c) `Compare("abc", "") returns greater`
d) `Compare("pqr", "pqr") returns equal`
e) `Compare("pqr", "pqrstuv") returns less`
f) `Compare("pqrstuv", "pqr") returns greater`
g) `Compare("abc", "pqr") returns less`
h) `Compare("pqr", "abc") returns greater`
i) `Compare("abcdef", "p") returns less`
j) `Compare("p", "abcdef") returns greater`

The procedure `Equal`

The value of the call

```
Equal(stringVal1, stringVal2)
```

shall be the value of the Boolean expression

```
Strings.Compare(stringVal1, stringVal2) = Strings.equal
```

$Equal : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} \mathbb{B}$

$Equal\,(args)handler \triangleq$
  let $[stringVal1, stringVal2] = args$ in
  return $pure\text{-}string\,(stringVal1) = pure\text{-}string\,(stringVal2)$ 579 579

The procedure `FindNext`

The call

```
FindNext(pattern, stringToSearch, startPos, patternFound, posOfPattern)
```

shall search forwards for the next occurrence of `pattern` in `stringToSearch`, starting the search in `stringToSearch` at position `startPos`. If `pattern` is found, the value *true* shall be assigned to `patternFound`, and `posOfPattern` shall contain the starting position of `pattern` in `stringToSearch`; `posOfPattern` shall be in the range `[startPos..LENGTH(stringToSearch)-1]`. Otherwise the value *false* shall be assigned to `patternFound` and `posOfPattern` shall be unchanged.

NOTES

14 The pattern might be found at the given value of `startPos`.

15 If `startPos > LENGTH(stringToSearch)-LENGTH(pattern)` then `patternFound` is returned as *false*.

$FindNext : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} ()$

$FindNext\,(args)handler \triangleq$
  let $[pattern, stringToSearch, startPos, patternFound, posOfPattern] = args$ in
  let $patstr = pure\text{-}string\,(pattern)$ in 579
  let $fullstr = pure\text{-}string\,(stringToSearch)$ in 579
  let $searchstr = subseq\,(fullstr, startix + 1, \mathsf{len}\ fullstr)$ in 579
  let $(f, n) = find\,(patstr, searchstr, startPos + 1)$ in 577
  $(assign(patternFound, f)\ handler$; 150
   if f
   then $assign(posOfPattern, n - 1)\ handler$ 150
   else skip)

annotations The actual parameter corresponding to the position of the pattern is to be left unchanged if the pattern is not found. Hence, although *find* returns (*false*, nil) the actual parameter, *pos*, is only updated if the pattern is found (i.e. $f = true$).

Note also that the start index (*startix*) to both *subseq* and *find*, is incremented by 1 because VDM-SL sequences are indexed from 1 onwards.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

$find : Char^* \times Char^* \times \mathbb{N} \rightarrow (\mathbb{B} \times [\mathbb{N}])$

$find\,(pat, str, start) \triangleq$
  if len $str <$ len pat
  then $mk\text{-}(\text{false}, \text{nil})$
  else if $str\,(1, \ldots, \text{len}\ pat) = pat$
    then $mk\text{-}(\text{true}, start)$
    else $find\,(pat, \text{tl}\ str, start + 1)$ 577

EXAMPLE 10 — Forwards string search.

a)
```
large := "Hello hello hello";
FindNext("ll", large, 0, found, pos)
```
results in:
`found` having value `TRUE`
`pos` having value 2

b)
```
large := "Hello hello hello";
FindNext("ll", large, 0, found, pos);
FindNext("ll", large, pos+1, found, pos)
```
results in:
`found` having value `TRUE`
`pos` having value 8

c)
```
large := "abcdefghijklmnopqrstuvwxyz";
ch := "x";
FindNext(String1{ch}, large, 0, found, pos)
```
results in:
`found` having value `TRUE`
`pos` having value 23

d)
```
large := "abcdefghijklmnopqrstuvwxyz";
ch := "x";
FindNext(String1{ch}, large, 26, found, pos)
```
results in:
`found` having value `FALSE`
`pos` remains unchanged

The procedure FindPrev

The call

```
FindPrev(pattern, stringToSearch, startPos, patternFound, posOfPattern)
```

shall look backwards for an occurrence of pattern in the string value in `stringToSearch`, starting the search in `stringToSearch` at position `startPos`. If `pattern` is found, the value *true* shall be assigned to `patternFound`, and `posOfPattern` shall contain the starting position of `pattern` in `stringToSearch`; `posOfPattern` shall be in the range `[0 .. startPos]`. Otherwise the value *false* shall be assigned to `patternFound` and `posOfPattern` shall be unchanged.

NOTES

16 The pattern might be found at the given value of `startPos`.

17 If `startPos > LENGTH(stringToSearch)-LENGTH(pattern)` the whole string value is searched.

$FindPrev : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} ()$

$FindPrev\,(args)handler \triangleq$
  let $[pattern, stringToSearch, startPos, patternFound, posOfPattern] = args$ in
  let $patstr = pure\text{-}string\,(pattern)$ in 579
  let $fullstr = pure\text{-}string\,(stringToSearch)$ in 579
  let $searchstr = subseq\,(fullstr, 1, startPos + \text{len}\ patstr)$ in 579
  let $revstr = revstring\,(searchstr)$ in 577
  let $revpat = revstring\,(patstr)$ in 577
  let $mk\text{-}(f, n) = find\,(revpat, revstr, 1)$ in 577
  $(assign(patternFound, f)\ handler;$ 150
   if f
   then $assign(posOfPattern, \text{len}\ revstr - (n-1) - (\text{len}\ patstr - 1) - 1)\ handler$ 150
   else skip)

annotations The actual parameter corresponding to the position of the pattern is to be left unchanged if the pattern is not found. Hence, although *find* returns (*false*, nil) the actual parameter, *pos*, is only updated if the pattern is found (i.e. $f = true$).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
revstring : Char* → Char*
revstring (s) △
  if s = [ ]
  then [ ]
  else revstring (tl s) ↷ [hd s]                                    577
```

EXAMPLE 11 — Backwards string search.

a)
```
large := "aabbbcccc";
FindPrev("cc", large, 200, found, pos)
```
results in:
`found` having value `TRUE`
`pos` having value `7`

b)
```
large := "aabbbcccc";
FindPrev("cc", large, 200, found, pos);
FindPrev("cc", large, pos-1, found, pos)
```
results in:
`found` having value `TRUE`
`pos` having value `6`

c)
```
large := "Maybe this makes sense";
FindPrev("se", large, 200, found, pos)
```
results in:
`found` having value `TRUE`
`pos` having value `20`

d)
```
large := "Maybe this makes sense";
FindPrev("se", large, 20, found, pos);
FindPrev("se", large, pos-1, found, pos)
```
results in:
`found` having value `TRUE`
`pos` having value `17`

The procedure `FindDiff`

The call

```
FindDiff(stringVal1, stringVal2, differenceFound, posOfDifference)
```

shall compare the string values in `stringVal1` and `stringVal2`. The value *false* shall be assigned to `differenceFound` if the string values are equal and *true* otherwise. If `differenceFound` is *true*, the position of the first difference between the string values shall be assigned to `posOfDifference`; otherwise `posOfDifference` shall be unchanged.

```
FindDiff : Arguments → Exception-handler →o ()
FindDiff (args)handler △
  let [stringVal1, stringVal2, differenceFound, posOfDifference] = args in
  let s1 = pure-string (stringVal1) in                              579
  let s2 = pure-string (stringVal2) in                              579
  let (f, n) = diff (s1, s2, 1) in                                  578
  (assign(differenceFound, f) handler ;                             150
   if f
   then assign(posOfDifference, n − 1) handler                      150
   else skip)

diff : Char* × Char* × ℕ → (𝔹 × [ℕ])
diff (s1, s2, pos) △
  (s1 = [ ] ∧ s2 = [ ] → mk-()false, nil ,
   s1 = [ ] ∧ s2 ≠ [ ] → mk-()true, pos,
   s1 ≠ [ ] ∧ s2 = [ ] → mk-()true, pos,
   hd s1 ≠ hd s2       → mk-()true, pos,
   hd s1 = hd s2       → diff (tl s1, tl s2, pos + 1))              578
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLE 12 — Finding string differences.

a) `FindDiff("", "", areDiff, pos)`
results in:
`areDiff` having value `FALSE`
`pos` being unchanged

b) `FindDiff("abc", "", areDiff, pos)`
results in:
`areDiff` having value `TRUE`
`pos` having value `0`

c) `FindDiff("", "abc", areDiff, pos)`
`areDiff` having value `TRUE`
`pos` having value `0`

d) `FindDiff("pqr", "pqt", areDiff, pos)`
`areDiff` having value `TRUE`
`pos` having value `2`

e) `FindDiff("pqr", "pqrstuv", areDiff, pos)`
`areDiff` having value `TRUE`
`pos` having value `3`

f) `FindDiff("pqrstuv", "pqr", areDiff, pos)`
`areDiff` having value `TRUE`
`pos` having value `3`

9.4.3 Auxiliary Formal Model Definitions

The following functions and values are used in defining the module `Strings`.

$$capacity : Array\text{-}variable \to \mathbb{N}$$

$$capacity\,(avar) \triangleq$$
let $mk\text{-}Array\text{-}variable(v) = avar$ in
card dom v

annotations Obtain the number of components in a concrete string variable.

$$subseq : Char^* \times \mathbb{N}_1 \times \mathbb{N} \to Char^*$$

$$subseq\,(s,i,j) \triangleq$$
$(i \leq \text{len } s \wedge 1 \leq j \leq \text{len } s \to s\,(i,\ldots,j),$
$i \leq \text{len } s \wedge j > \text{len } s \to s\,(i,\ldots,\text{len } s),$
$i > \text{len } s \vee j = 0 \to [])$

annotations The function *subseq* is used to select a subsequence of a string, even when the start and end indexes are outside the length of the string (the standard VDM-SL subsequence operator has an over-strong pre-condition for the purposes of this definition).

The first index of the abstract string must not be negative, because Modula-2 start indexes have a minimum ordinal value of zero, and will always be incremented before a call of subseq.

$$obtain\text{-}string : Variable \xrightarrow{o} Char^*$$

$$obtain\text{-}string\,(svar) \triangleq$$
def $sval = value\text{-}of\text{-}a\text{-}variable(svar)$ in 224
return $pure\text{-}string\,(sval)$ 579

annotations The value in the concrete string variable is obtained. Concrete string variables are array variables whose values are a map. The auxiliary function *value-of-a-variable* returns a value of type *String-value*, which is a mapping from ordinal values to characters. The range of this mapping may contain a string terminator and any undefined values that the array value contains. The use of *pure-string* removes these values, leaving a VDM-SL character sequence that represents (the abstract) string value.

$$pure\text{-}string : String\text{-}value \to Char^*$$

$$pure\text{-}string\,(sval) \triangleq$$
let $str = map\text{-}to\text{-}seq\,(sval)$ in 615
if END-OF-STRING-CHAR $\in$ elems str
then let $head, tail : Char^*$ be st $head \frown tail = str \wedge$
END-OF-STRING-CHAR $\notin$ elems $head \wedge$
hd $tail$ = END-OF-STRING-CHAR in
$head$
else str

annotations This function strips any string terminator and any character values after the first string terminator found in a sequence obtained from the concrete string value (i.e. the character array value).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

values

compareresults = [**less**, **equal**, **greater**];

less = *mk-Enumeration-value* (**less**, *compareresults*);

equal = *mk-Enumeration-value* (**equal**, *compareresults*);

greater = *mk-Enumeration-value* (**greater**, *compareresults*)

annotations These are the enumeration values for use in the formal specification of the abstract implementation module.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.5 The String Conversions Library

The string conversions library defined in this clause allows the conversion of the values of numeric data types to and from character string representations. The modules `WholeStr`, `RealStr`, and `LongStr` provide simple high-level facilities for converting to and from strings and whole number and real number data types. Low-level facilities are provided by the corresponding modules `WholeConv`, `RealConv`, and `LongConv`. Common data types and values that are used in the definition modules are defined by the module `ConvTypes`.

The formats for string conversions correspond to those for numeric input and output, except that the numeric output routines provide for (right) alignment in a specified field width — see 9.2.2.2 and 9.2.2.3.

9.5.1 Common Data Types

The module `ConvTypes` defines types and values that are used in the high-level and low-level string conversion definition modules. Where appropriate, the conversion modules define types in terms of those defined in `ConvTypes`. Direct import from this module is not normally necessary in modules that are clients of the high-level conversion modules.

9.5.1.1 The Module ConvTypes

The module `ConvTypes` defines the enumeration type `ConvResults` with values for expressing the format of strings that are interpreted as representing values of numeric data types. The module also defines the types `ScanClass` and `ScanState`, which the low-level conversion modules use in the definition of procedures that control lexical scanning.

The module `ConvTypes` has, directly or by import, no state, no initialization, and no termination.

9.5.1.1.1 The Definition Module of ConvTypes

```
DEFINITION MODULE ConvTypes;

  (* Common types used in the string conversion modules *)

TYPE
  ConvResults =     (* Values of this type are used to express the format of a string *)
  (
    strAllRight,    (* the string format is correct for the corresponding conversion *)
    strOutOfRange,  (* the string is well-formed but the value cannot be represented *)
    strWrongFormat, (* the string is in the wrong format for the conversion *)
    strEmpty        (* the given string is empty *)
  );

  ScanClass =  (* Values of this type are used to classify input to finite state scanners *)
  (
    padding,    (* a leading or padding character at this point in the scan - ignore it *)
    valid,      (* a valid character at this point in the scan - accept it *)
    invalid,    (* an invalid character at this point in the scan - reject it *)
    terminator (* a terminating character at this point in the scan (not part of token) *)
  );

  ScanState =  (* The type of lexical scanning control procedures *)
    PROCEDURE (CHAR, VAR ScanClass, VAR ScanState);

END ConvTypes.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9.5.2 High-Level String Conversion Modules

Separate high-level string conversion modules are defined for the whole number types (INTEGER and CARDINAL), and for the real number types (REAL and LONGREAL). These all use decimal notation.

In calls of procedures converting from strings, the source parameter str is assumed to contain a string value (which is terminated by the string terminator character if the length of the string is less than the capacity of the array). While leading spaces are ignored, the entire remainder of the string has to be in the correct format for the conversion to take place.

In calls of procedures converting to strings, the destination parameter str is assigned a string value (which is terminated by the string terminator character if the length of the string is less than the capacity of the array). If the destination parameter is of insufficient capacity, the string is truncated. Users may determine whether truncation will occur by using procedures such as LengthCard from the low-level string conversion modules — see 9.5.3.1.2.

NOTE — The string conversion procedures may be used with strings read by STextIO.ReadToken or TextIO.ReadToken, if it is required that entire space-character delimited tokens are in the correct format — see 9.2.2.1.2.

EXAMPLE — Conversion of strings read by ReadToken.

This example shows how space-character delimited tokens that cannot be converted to values of type CARDINAL can be replaced by descriptive text:

```
firstOnLine := TRUE;
STextIO.ReadToken(inStr);
WHILE SIOResult.ReadResult() <> SIOResult.endOfInput DO
  WholeStr.StrToCard(inStr, inCard, inRes);
  CASE inRes OF
  | ConvTypes.strAllRight .. ConvTypes.strWrongFormat:
     IF firstOnLine THEN
       firstOnLine := FALSE
     ELSE
       STextIO.WriteString(" ")
     END
  | ConvTypes.strEmpty:
     firstOnLine := TRUE;
     STextIO.SkipLine
  END;
  CASE inRes OF
  | ConvTypes.strAllRight:
     STextIO.WriteString(inStr)
  | ConvTypes.strOutOfRange:
     STextIO.WriteString("out-of-range")
  | ConvTypes.strWrongFormat:
     STextIO.WriteString("wrong-format")
  | ConvTypes.strEmpty:
     STextIO.WriteLn
  END;
  STextIO.ReadToken(inStr)
END
```

It is assumed here that the capacity of the character array variable inStr is sufficient to accommodate the longest token in the input stream.

9.5.2.1 The Module WholeStr

The module WholeStr provides for the conversion of values of the type INTEGER and the type CARDINAL to and from strings.

The module WholeStr has, directly or by import, no state, no initialization, and no termination.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9.5.2.1.1 The Definition Module of WholeStr

```
DEFINITION MODULE WholeStr;

  (* Whole-number/string conversions *)

IMPORT
  ConvTypes;

TYPE
  ConvResults = ConvTypes.ConvResults; (* strAllRight, strOutOfRange, strWrongFormat, strEmpty *)

(* the string form of a signed whole number is
     ["+" | "-"], decimal digit, {decimal digit}
*)

PROCEDURE StrToInt (str: ARRAY OF CHAR; VAR int: INTEGER; VAR res: ConvResults);
  (* Ignores any leading spaces in str. If the subsequent characters in str are in the
     format of a signed whole number, assigns a corresponding value to int. Assigns a
     value indicating the format of str to res.
  *)

PROCEDURE IntToStr (int: INTEGER; VAR str: ARRAY OF CHAR);
  (* Converts the value of int to string form and copies the possibly truncated result to str. *)

(* the string form of an unsigned whole number is
     decimal digit, {decimal digit}
*)

PROCEDURE StrToCard (str: ARRAY OF CHAR; VAR card: CARDINAL; VAR res: ConvResults);
  (* Ignores any leading spaces in str. If the subsequent characters in str are in the
     format of an unsigned whole number, assigns a corresponding value to card.
     Assigns a value indicating the format of str to res.
  *)

PROCEDURE CardToStr (card: CARDINAL; VAR str: ARRAY OF CHAR);
  (* Converts the value of card to string form and copies the possibly truncated result to str. *)

END WholeStr.
```

9.5.2.1.2 The Procedures of WholeStr

The string form of a signed whole number shall be

```
["+" | "-"], decimal digit, {decimal digit}
```

The string form of an unsigned whole number shall be

```
decimal digit, {decimal digit}
```

The procedure `StrToInt`

The call `StrToInt(str,int,res)` shall ignore leading spaces in `str` and, with assignment to `int` as defined below, shall assign to `res` the value:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

`strAllRight`	if the remainder of `str` represents a complete signed whole number in the range of the type `INTEGER` — the value of this number shall be assigned to `int`;
`strOutOfRange`	if the remainder of `str` represents a complete signed whole number but its value is out of the range of the type `INTEGER` — the value `MAX(INTEGER)` or `MIN(INTEGER)` shall be assigned to `int` according to the sign of the number;
`strWrongFormat`	if there are remaining characters in `str` but these are not in the form of a complete signed whole number — the value of `int` is not defined;
`strEmpty`	if there are no remaining characters in `str` — the value of `int` is not defined.

```
StrToInt : Arguments → Exception-handler ^o→ ()
StrToInt (args)handler ≙
  let [str, int, res] = args in
  let string = pure-string (str) in                                   579
  let remainder = no-leading-spaces (string) in                       451
  let mk-(value, result) = strint (remainder) in                      584
  (if value = nil
   then let impdep : (SIGNED-TYPE | undefined ) in
        assign(int, impdep) handler                                   150
   else assign(int, value) handler ;                                  150
   assign(res, convenum (result)) handler )                           150 606
```

Domains

types

```
  Conv-result = ALL-RIGHT | OUT-OF-RANGE | WRONG-FORMAT | EMPTY
```

```
strint : Char* → ([ℤ] × Conv-result)
strint (string) ≙
  let sign, digits : Char* be st
        sign ↷ digits = string ∧ digits = no-leading-sign (string) in 451
  if is-digits (digits)                                               605
  then let value = if is-optminus(sign)
                   then − numeric-value (digits)                      606
                   else numeric-value (digits) in                     606
       let result = if value > max-signed-value ∨ value < min-signed-value
                    then OUT-OF-RANGE
                    else ALL-RIGHT in
       let rvalue = if result = ALL-RIGHT
                    then value
                    else if value > 0 then max-signed-value else min-signed-value in
       mk-(rvalue, result)
  else let result = if string = [ ]
                    then EMPTY
                    else WRONG-FORMAT in
       mk-(nil , result)
```

The procedure `IntToStr`

The call `IntToStr(int,str)` shall assign to `str` the possibly truncated string corresponding to the value of `int`. A sign shall be included only for negative values.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
IntToStr : Arguments → Exception-handler ⟶° ()
IntToStr (args)handler ≜
  let [int, str] = args in
  let number = intstr (int) in                                   585
  array-variable-assignment(str, number) handler                 150

intstr : ℤ → Char*
intstr (int) ≜
  let sign, digits : Char* be st
        sign = if int < 0 then [minus-character] else [] ∧
        numeric-value (digits) = abs int in                      606
  sign ↷ digits
```

The procedure StrToCard

The call `StrToCard(str,card,res)` shall ignore leading spaces in `str` and, with assignment to `card` as defined below, shall assign to `res` the value:

`strAllRight`	if the remainder of `str` represents a complete unsigned whole number in the range of the type `CARDINAL` — the value of this number shall be assigned to `card`;
`strOutOfRange`	if the remainder of `str` represents a complete unsigned whole number but its value is out of the range of the type `CARDINAL` — the value `MAX(CARDINAL)` shall be assigned to `card`.
`strWrongFormat`	if there are remaining characters in `str` but these are not in the form of a complete unsigned whole number — the value of `card` is not defined;
`strEmpty`	if there are no remaining characters in `str` — the value of `card` is not defined.

```
StrToCard : Arguments → Exception-handler ⟶° ()
StrToCard (args)handler ≜
  let [str, card, res] = args in
  let string = pure-string (str) in                              579
  let remainder = no-leading-spaces (string) in                  451
  let mk-(value, result) = strcard (remainder) in                585
  (if value = nil
   then let impdep : (UNSIGNED-TYPE | undefined ) in
        assign(card, impdep) handler                             150
   else assign(card, value) handler ;                            150
   assign(res, convenum (result)) handler )                      150 606

strcard : Char* → ([ℕ] × Conv-result)
strcard (string) ≜
  if is-digits (string)                                          605
  then let value = numeric-value (string) in                     606
       let result = if value > max-unsigned-value
                    then OUT-OF-RANGE
                    else ALL-RIGHT in
       let rvalue = if result = ALL-RIGHT then value else max-unsigned-value in
       mk-(rvalue, result)
  else let result = if string = []
                    then EMPTY
                    else WRONG-FORMAT in
       mk-(nil , result)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The procedure CardToStr

The call CardToStr(card,str) shall assign to str the possibly truncated string corresponding to the value of card.

CardToStr : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

CardToStr (*args*)*handler* $\triangleq$
  let [*card*, *str*] = *args* in
  let *number* = *cardstr* (*card*) in 586
  array-variable-assignment(*str*, *number*) *handler* 150

cardstr : N → *Char**

cardstr (*card*) $\triangleq$
  let *number* : *Char** be st
    numeric-value (*number*) = *card* in 606
  number

9.5.2.2 The Modules RealStr and LongStr

The modules RealStr and LongStr provide for the conversion of real numbers to and from strings. In the case of RealStr, real number parameters are of the type REAL. In the case of LongStr, real number parameters are of the type LONGREAL.

The modules RealStr and LongStr have, directly or by import, no state, no initialization, and no termination.

9.5.2.2.1 The Definition Module of RealStr

```
DEFINITION MODULE RealStr;

  (* REAL/string conversions *)

IMPORT
  ConvTypes;

TYPE
  ConvResults = ConvTypes.ConvResults; (* strAllRight, strOutOfRange, strWrongFormat, strEmpty *)

(* the string form of a signed fixed-point real number is
     ["+" | "-"], decimal digit, {decimal digit}, [".", {decimal digit}]
*)

(* the string form of a signed floating-point real number is
     signed fixed-point real number, "E"|"e", ["+" | "-"], decimal digit, {decimal digit}
*)

PROCEDURE StrToReal (str: ARRAY OF CHAR; VAR real: REAL; VAR res: ConvResults);
  (* Ignores any leading spaces in str. If the subsequent characters in str are in the
     format of a signed real number, assigns a corresponding value to real.
     Assigns a value indicating the format of str to res.
  *)

PROCEDURE RealToFloat (real: REAL; sigFigs: CARDINAL; VAR str: ARRAY OF CHAR);
  (* Converts the value of real to floating-point string form, with sigFigs significant
     figures, and copies the possibly truncated result to str.
  *)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
PROCEDURE RealToEng (real: REAL; sigFigs: CARDINAL; VAR str: ARRAY OF CHAR);
  (* Converts the value of real to floating-point string form, with sigFigs significant
     figures, and copies the possibly truncated result to str. The number is scaled with
     one to three digits in the whole number part and with an exponent that is a
     multiple of three.
  *)

PROCEDURE RealToFixed (real: REAL; place: INTEGER; VAR str: ARRAY OF CHAR);
  (* Converts the value of real to fixed-point string form, rounded to the given place
     relative to the decimal point, and copies the possibly truncated result to str.
  *)

PROCEDURE RealToStr (real: REAL; VAR str: ARRAY OF CHAR);
  (* Converts the value of real as RealToFixed if the sign and magnitude can be shown
     within the capacity of str, or otherwise as RealToFloat, and copies the possibly
     truncated result to str. The number of places or significant digits depend on the
     capacity of str.
  *)

END RealStr.
```

9.5.2.2.2 The Definition Module of LongStr

```
DEFINITION MODULE LongStr;

  (* LONGREAL/string conversions *)

IMPORT
  ConvTypes;

TYPE
  ConvResults = ConvTypes.ConvResults; (* strAllRight, strOutOfRange, strWrongFormat, strEmpty *)

(* the string form of a signed fixed-point real number is
     ["+" | "-"], decimal digit, {decimal digit}, [".", {decimal digit}]
*)

(* the string form of a signed floating-point real number is
     signed fixed-point real number, "E"|"e", ["+" | "-"], decimal digit, {decimal digit}
*)

PROCEDURE StrToReal (str: ARRAY OF CHAR; VAR real: LONGREAL; VAR res: ConvResults);
  (* Ignores any leading spaces in str. If the subsequent characters in str are in the
     format of a signed real number, assigns a corresponding value to real.
     Assigns a value indicating the format of str to res.
  *)

PROCEDURE RealToFloat (real: LONGREAL; sigFigs: CARDINAL; VAR str: ARRAY OF CHAR);
  (* Converts the value of real to floating-point string form, with sigFigs significant
     figures, and copies the possibly truncated result to str.
  *)

PROCEDURE RealToEng (real: LONGREAL; sigFigs: CARDINAL; VAR str: ARRAY OF CHAR);
  (* Converts the value of real to floating-point string form, with sigFigs significant
     figures, and copies the possibly truncated result to str. The number is scaled with
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
    one to three digits in the whole number part and with an exponent that is a
    multiple of three.
  *)

PROCEDURE RealToFixed (real: LONGREAL; place: INTEGER; VAR str: ARRAY OF CHAR);
  (* Converts the value of real to fixed-point string form, rounded to the given place
    relative to the decimal point, and copies the possibly truncated result to str.
  *)

PROCEDURE RealToStr (real: LONGREAL; VAR str: ARRAY OF CHAR);
  (* Converts the value of real as RealToFixed if the sign and magnitude can be shown
    within the capacity of str, or otherwise as RealToFloat, and copies the possibly
    truncated result to str. The number of places or significant digits depend on the
    capacity of str.
  *)

END LongStr.
```

9.5.2.2.3 The Procedures of RealStr and LongStr

The semantics of the two modules shall be the same, except that when module `RealStr` refers to the pervasive type `REAL`, the corresponding procedure in `LongStr` shall refer to the pervasive type `LONGREAL`.

NOTE — The above statement is merely to avoid needless repetition of the semantics for `LongStr`.

The string form of a signed fixed-point real number shall be

```
["+" | "-"], decimal digit, {decimal digit},
[".", {decimal digit}]
```

The string form of a signed floating-point real number shall be

```
signed fixed-point real number,
"E"|"e", ["+" | "-"], decimal digit, {decimal digit}
```

The procedure `StrToReal`

The call `StrToReal(str,real,res)` shall ignore leading spaces in `str` and shall assign values to `res` and `real` as follows:

`strAllRight`	if the remainder of `str` represents a complete signed real number in the range of the type of `real` — the value of this number shall be assigned to `real`;
`strOutOfRange`	if the remainder of `str` represents a complete signed real number but its value is out of the range of the type of `real` — the maximum or minimum value of the type of `real` shall be assigned to `real` according to the sign of the number;
`strWrongFormat`	if there are remaining characters in `str` but these are not in the form of a complete signed real number — the value of `real` is not defined;
`strEmpty`	if there are no remaining characters in `str` — the value of `real` is not defined.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
StrToReal : Arguments → Exception-handler →° ()
StrToReal (args)handler ≙
  let [str, real, res] = args in
  let string = pure-string (str) in                                   579
  let remainder = no-leading-spaces (string) in                       451
  let mk-(value, result) = strreal (remainder) in                     589
  (if value = nil
   then let impdep : (REAL-TYPE | undefined ) in
        assign(real, impdep) handler                                  150
   else assign(real, value) handler ;                                 150
   assign(res, convenum (result)) handler )                           150 606

strreal : Char* → ([ℝ] × Conv-result)
strreal (string) ≙
  let sign, rest1 : Char* be st
        sign ↷ rest1 = string ∧ rest1 = no-leading-sign (string) in   451
  let whole, rest2 : Char* be st
        whole ↷ rest2 = rest1 ∧ rest2 = no-leading-digits (rest1) in  451
  let rest3 = no-leading-char (rest2, decimal-point-character) in     451
  let frac, rest4 : Char* be st
        frac ↷ rest4 = rest3 ∧ rest4 = no-leading-digits (rest3) in   451
  let e, rest5 : Char* be st
        e ↷ rest5 = rest4 ∧ rest5 = no-leading-exponent (rest4) in    451
  let esign, exp : Char* be st esign ↷ exp = rest5 ∧ exp = no-leading-sign (rest5) in  451
  if is-digits (whole) ∧ (rest4 = [] ∨ (e ≠ [] ∧ is-digits (exp)))    605 605
  then let scale = if exp = [] then 1.0
                   else if is-optminus (esign)                        606
                        then 10.0 ↑ −numeric-value (exp)              606
                        else 10.0 ↑ numeric-value (exp) in            606
       let value = if is-optminus (sign)                              606
                   then − fixed-point-value (whole, frac) × scale     606
                   else fixed-point-value (whole, frac) × scale in    606
       let result = if value > max-real-value ∨ value < min-real-value
                    then OUT-OF-RANGE
                    else ALL-RIGHT in
       let rvalue = if result = ALL-RIGHT
                    then value
                    else if value > 0 then max-real-value else min-real-value in
       mk-(rvalue, result)
  else let result = if string = []
                    then EMPTY
                    else WRONG-FORMAT in
       mk-(nil , result)
```

The procedure `RealToFloat`

The call `RealToFloat(real,sigFigs,str)` shall assign to `str` the possibly truncated string corresponding to the value of `real` in floating-point form. A sign shall be included only for negative values. One significant digit shall be included in the whole number part. The signed exponent part shall be included only if the exponent value is not 0. If the value of `sigFigs` is greater than 0, that number of significant digits shall be included, otherwise an implementation-defined number of significant digits shall be included. The decimal point shall not be included if there are no significant digits in the fractional part.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

RealToFloat : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

RealToFloat (*args*)*handler* $\triangleq$
  let [*real*, *sigfigs*, *str*] = *args* in
  let *number* = *floatstr* (*real*, *sigfigs*) in 590
  array-variable-assignment(*str*, *number*) *handler* 150

values

  default-figs : $\mathbb{N}_1$ = implementation-defined

floatstr : $\mathbb{R} \times \mathbb{N} \to Char^*$

floatstr (*value*, *figs*) $\triangleq$
  let *mantissa* : $\mathbb{R}$, *exp* : $\mathbb{N}$ be st *mantissa* × 10 ↑ *exp* = abs *value* ∧ *mantissa* < 10.0 ∧ *mantissa* ≥ 0.0 in
  let *sig* = if *figs* = 0 then *default-figs* else *figs* in
  let *round* : $\mathbb{N}$ be st abs (*round* − *mantissa* × 10 ↑ (*sig* − 1)) ≤ 0.5 in
  let *whole*, *frac* : *Char** be st *fixed-point-value* (*whole*, *frac*) × 10 ↑ (*sig* − 1) = *round* ∧ len *frac* = *sig* − 1 in 606
  let *fracpart* = if *sig* > 1 then [*decimal-point-character*] ⌢ *frac* else [] in
  let *sign* = if *value* < 0 then [*minus-character*] else [] in
  let *esign* = if *exp* < 0 then [*minus-character*] else [] in
  let *sexp* : *Char** be st *numeric-value* (*sexp*) = abs *exp* in 606
  let *expart* = if *exp* = 0 then [] else [*exponent-character*] ⌢ *esign* ⌢ *sexp* in
  sign ⌢ *whole* ⌢ *fracpart* ⌢ *expart*

annotations Return the floating-point representation of *value* to *figs* significant figures.

The procedure `RealToEng`

The call `RealToEng(real,sigFigs,str)` shall behave as the call `RealToFloat(real,sigFigs,str)` except that the number shall be scaled with one to three digits in the whole number part and with an exponent that is a multiple of three.

RealToEng : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

RealToEng (*args*)*handler* $\triangleq$
  let [*real*, *sigfigs*, *str*] = *args* in
  let *number* = *engstr* (*real*, *sigfigs*) in 590
  array-variable-assignment(*str*, *number*) *handler* 150

engstr : $\mathbb{R} \times \mathbb{N} \to Char^*$

engstr (*value*, *figs*) $\triangleq$
  let *mantissa* : $\mathbb{R}$, *exp* : $\mathbb{N}$ be st *mantissa* × 10 ↑ *exp* = abs *value* ∧
      mantissa < 1000.0 ∧ *mantissa* ≥ 0.0 ∧ *exp* rem 3 = 0 in
  let *sig* = if *figs* = 0 then *default-figs* else *figs* in
  let *wholeplaces* = if *mantissa* ≥ 100.0 then 3
      elseif *mantissa* ≥ 10.0 then 2 else 1 in
  let *round* : $\mathbb{N}$ be st abs (*round* − *mantissa* × 10 ↑ (*sig* − *wholeplaces*)) ≤ 0.5 in
  let *whole*, *frac* : *Char** be st *fixed-point-value* (*whole*, *frac*) × 10 ↑ (*sig* − *wholeplaces*) = *round* ∧ 606
      len *frac* = if *sig* > *wholeplaces* then *sig* − *wholeplaces* else 0 in
  let *fracpart* = if *sig* > *wholeplaces* then [*decimal-point-character*] ⌢ *frac* else [] in
  let *sign* = if *value* < 0 then [*minus-character*] else [] in
  let *esign* = if *exp* < 0 then [*minus-character*] else [] in
  let *sexp* : *Char** be st *numeric-value* (*sexp*) = abs *exp* in 606
  let *expart* = if *exp* = 0 then [] else [*exponent-character*] ⌢ *esign* ⌢ *sexp* in
  sign ⌢ *whole* ⌢ *fracpart* ⌢ *expart*

annotations Return the engineering floating-point representation of *value* to *figs* significant figures.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The procedure `RealToFixed`

The call `RealToFixed(real,place,str)` shall assign to `str` the possibly truncated string corresponding to the value of `real` in fixed-point form. A sign shall be included only for negative values. At least one digit shall be included in the whole number part. The value shall be rounded to the given value of `place` relative to the decimal point. The decimal point shall be suppressed if `place` is less than 0.

RealToFixed : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

RealToFixed (*args*)*handler* $\triangleq$
  let [*real*, *place*, *str*] = *args* in
  let *number* = *fixedstr* (*real*, *place*) in 591
  array-variable-assignment(*str*, *number*) *handler* 150

fixedstr : $\mathbb{R} \times \mathbb{N} \to$ *Char**

fixedstr (*value*, *place*) $\triangleq$
  let *power* = if *place* < 0 then *place* + 1 else *place* in
  let *round* : $\mathbb{N}$ be st abs (*round* − abs *value* × 10 ↑ *power*) ≤ 0.5 in
  let *whole*, *frac* : *Char** be st *fixed-point-value* (*whole*, *frac*) × 10 ↑ *power* = *round* ∧ 606
    len *frac* = if *power* > 0 then *power* else 0 in
  let *fracpart* = if *place* ≥ 0 then [*decimal-point-character*] ⌢ *frac* else [] in
  let *sign* = if *value* < 0 then [*minus-character*] else [] in
  sign ⌢ *whole* ⌢ *fracpart*

annotations Return the fixed-point representation of *value* rounded to *place*.

The procedure `RealToStr`

If the sign and magnitude of `real` can be shown within the capacity of `str`, the call `RealToStr(real,str)` shall behave as the call `RealToFixed(real,place,str)`, with a value of `place` chosen to fill exactly the remainder of `str`. Otherwise, the call shall behave as the call `RealToFloat(real,sigFigs,str)`, with a value of `sigFigs` of at least one, but otherwise limited to the number of significant digits that can be included together with the sign and exponent part in `str`.

RealToStr : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

RealToStr (*args*)*handler* $\triangleq$
  let [*real*, *str*] = *args* in
  let *capacity* = card dom *str* in
  let *number* = *realstr* (*real*, *capacity*) in 591
  array-variable-assignment(*str*, *number*) *handler* 150

realstr : $\mathbb{R} \times \mathbb{N}_1 \to$ *Char**

realstr (*value*, *width*) $\triangleq$
  let *round* : $\mathbb{N}$ be st abs (*round* − abs *value*) ≤ 0.5 in
  let *whole* : *Char** be st *numeric-value* (*whole*) = *round* in 606
  let *needs* = if *value* < 0 then len *whole* + 1 else len *whole* in
  if *width* ≥ *needs*
  then let *place* = *width-needs*-1 in
    fixedstr (*value*, *place*) 591
  else let *mantissa* : $\mathbb{R}$, *exp* : $\mathbb{N}$ be st *mantissa* × 10 ↑ *exp* = abs *value* ∧
      mantissa < 10.0 ∧ *mantissa* ≥ 0.0 in
    let *l1* = if *value* < 0 then 2 else 1 in
    let *sexp* : *Char** be st *numeric-value* (*sexp*) = abs *exp* in 606
    let *l2* = if *exp* = 0 then 0
      elseif *exp* > 0 then len *sexp* else len *sexp* + 1 in
    let *figs* = if *width* − *l1* − *l2* > 1 then *width* − *l1* − *l2* − 1 else 1 in
    floatstr (*value*, *figs*) 590

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9.5.3 Low-Level String Conversion Modules

Separate low-level string conversion modules are defined for the whole number types (INTEGER and CARDINAL), and for the real number types (REAL and LONGREAL). These all use decimal notation.

Procedures are defined to return the length of the string that is required to represent a given value, to return the format of a given string, to return the value of a string known to be in the correct format for conversion, and to allow control of lexical scanning of character sequences.

NOTE — The types designated by ConvTypes.ScanClass and ConvTypes.ScanState are not given aliases in the low-level conversion modules. This is because clients of a separate finite state interpreter module need to refer only to procedures such as WholeConv.ScanInt, which represent the start state, and not to the types themselves.

9.5.3.1 The Module WholeConv

The module WholeConv provides low-level string conversion procedures for values of the type INTEGER and values of the type CARDINAL.

9.5.3.1.1 The Definition Module of WholeConv

```
DEFINITION MODULE WholeConv;

  (* Low-level whole-number/string conversions *)

IMPORT
  ConvTypes;

TYPE
  ConvResults = ConvTypes.ConvResults; (* strAllRight, strOutOfRange, strWrongFormat, strEmpty *)

PROCEDURE ScanInt (inputCh: CHAR; VAR chClass: ConvTypes.ScanClass;
                   VAR nextState: ConvTypes.ScanState);
  (* Represents the start state of a finite state scanner for signed whole numbers -
     assigns class of inputCh to chClass and a procedure representing the next state to
     nextState.
  *)

PROCEDURE FormatInt (str: ARRAY OF CHAR): ConvResults;
  (* Returns the format of the string value for conversion to INTEGER. *)

PROCEDURE ValueInt (str: ARRAY OF CHAR): INTEGER;
  (* Returns the value corresponding to the signed whole number string value str if str
     is well-formed; otherwise raises the WholeConv exception.
  *)

PROCEDURE LengthInt (int: INTEGER): CARDINAL;
  (* Returns the number of characters in the string representation of int. *)

PROCEDURE ScanCard (inputCh: CHAR; VAR chClass: ConvTypes.ScanClass;
                    VAR nextState: ConvTypes.ScanState);
  (* Represents the start state of a finite state scanner for unsigned whole numbers -
     assigns class of inputCh to chClass and a procedure representing the next state to
     nextState.
  *)

PROCEDURE FormatCard (str: ARRAY OF CHAR): ConvResults;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  (* Returns the format of the string value for conversion to CARDINAL. *)

PROCEDURE ValueCard (str: ARRAY OF CHAR): CARDINAL;
  (* Returns the value corresponding to the unsigned whole number string value str if
     str is well-formed; otherwise raises the WholeConv exception.
   *)

PROCEDURE LengthCard (card: CARDINAL): CARDINAL;
  (* Returns the number of characters in the string representation of card. *)

PROCEDURE IsWholeConvException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional execution state because
     of the raising of the WholeConv exception; otherwise returns FALSE.
  *)

END WholeConv.
```

9.5.3.1.2 The Procedures of WholeConv

The procedure ScanInt

A call of ScanInt(inputCh,chClass,nextState) shall assign values to chClass and nextState depending upon the value of inputCh as shown in the following table.

Procedure	inputCh	chClass	nextState a procedure with behaviour of
ScanInt	space	padding	ScanInt
	sign	valid	*S*
	decimal digit	valid	*W*
	other	invalid	ScanInt
S	decimal digit	valid	*W*
	other	invalid	*S*
W	decimal digit	valid	*W*
	other	terminator	—

NOTE 1 — The procedure ScanInt corresponds to the start state of a finite state machine to scan for a character sequence that forms a signed whole number. Like ScanCard and the corresponding procedures in the other low-level string conversion modules, it may be used to control the actions of a finite state interpreter. As long as the value of chClass is other than terminator or invalid, the interpreter should call the procedure whose value is assigned to nextState by the previous call, supplying the next character from the sequence to be scanned. It may be appropriate for the interpreter to ignore characters classified as invalid, and proceed with the scan. This would be the case, for example, with interactive input, if only valid characters are being echoed in order to give interactive users an immediate indication of badly-formed data.

If the character sequence ends before one is classified as a terminator, the string-terminator character should be supplied as input to the finite state scanner. If the preceding character sequence formed a complete number, the string-terminator will be classified as terminator, otherwise it will be classified as invalid.

EXAMPLE — Use of ScanInt.

The following procedure uses ScanInt to locate the position of the first character in a string that follows any leading spaces and to locate the position of the first character that is not part of an integer. These positions will coincide if the string does not contain an integer after any leading spaces, and will be equal to the string length if no such character is contained in the string.

```
PROCEDURE FindInt(str: ARRAY OF CHAR; VAR first, next: CARDINAL);
VAR
  ch: CHAR;
  len, index: CARDINAL;
  state: ConvTypes.ConvState;
  class: ConvTypes.ConvClass;
BEGIN
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
    len := LENGTH(str);
    index := 0;
    first := len;
    state := WholeConv.ScanInt;
    LOOP
      IF index = len THEN EXIT END;
      state(str[index], class, state);
      CASE class OF
      | ConvTypes.padding:
      | ConvTypes.valid:
         IF index < first THEN first := index END;
      | ConvTypes.invalid, ConvTypes.terminator:
         EXIT
      END;
      INC(index)
    END;
    next := index
  END FindInt;
```

ScanInt : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

ScanInt (*args*)*handler* $\triangleq$
  let [*inputCh*, *chClass*, *nextState*] = *args* in
  (*isa-space* (*inputCh*) → (*assign*(*chClass*, *padding*) *handler*; 450 150
  assign(*nextState*, *ScanInt-pid*) *handler*), 150
  isa-sign (*inputCh*) → (*assign*(*chClass*, *valid*) *handler*; 605 150
  assign(*nextState*, *S-pid*) *handler*), 150
  isa-numeral (*inputCh*) → (*assign*(*chClass*, *valid*) *handler*; 451 150
  assign(*nextState*, *W-pid*) *handler*), 150
  others → (*assign*(*chClass*, *invalid*) *handler*; 150
  assign(*nextState*, *ScanInt-pid*) *handler*)) 150

S : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

S (*args*)*handler* $\triangleq$
  let [*inputCh*, *chClass*, *nextState*] = *args* in
  (*isa-numeral* (*inputCh*) → (*assign*(*chClass*, *valid*) *handler*; 451 150
  assign(*nextState*, *W-pid*) *handler*), 150
  others → (*assign*(*chClass*, *invalid*) *handler*; 150
  assign(*nextState*, *S-pid*) *handler*)) 150

W : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

W (*args*)*handler* $\triangleq$
  let [*inputCh*, *chClass*, *nextState*] = *args* in
  (*isa-numeral* (*inputCh*) → (*assign*(*chClass*, *valid*) *handler*; 451 150
  assign(*nextState*, *W-pid*) *handler*), 150
  others → *assign*(*chClass*, *terminator*) *handler*) 150

values

scanint-map = {*ScanInt-pid* ↦ *make-proper-procden* (*ScanInt*), 339
S-pid ↦ *make-proper-procden* (*S*), 339
W-pid ↦ *make-proper-procden* (*W*)} 339

annotations The mapping *scanint-map*, from elements of type *Procedure-id* to the corresponding procedure denotations, is part of the mapping *dens* used in the construction of the abstract implementation module — see annex B.

The procedure `FormatInt`

The value of the call `FormatInt(str)` shall be

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

`strAllRight`	if `str` has a value representing a complete signed whole number that is in the range of the type `INTEGER`;
`strOutOfRange`	if `str` has a value representing a complete signed whole number that is not in the range of `INTEGER`;
`strWrongFormat`	if `str` has a value with remaining characters that do not form a complete signed whole number;
`strEmpty`	if `str` has a value with no remaining characters;

ignoring any leading space characters in `str`.

FormatInt : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Enumeration-value*

FormatInt (*args*)*handler* $\triangleq$
  let [*str*] = *args* in
  let *string* = *pure-string* (*str*) in 579
  let *remainder* = *no-leading-spaces* (*string*) in 451
  let *mk-* (-, *result*) = *strint* (*remainder*) in 584
  return *convenum* (*result*) 606

The procedure `ValueInt`

If `str` has a value representing a signed whole number, the value of the call `ValueInt(str)` shall be the `INTEGER` value that corresponds to that number. Otherwise, the `WholeConv` exception shall be raised.

ValueInt : *Arguments* → *Exception-handler* $\xrightarrow{o}$ $\mathbb{Z}$

ValueInt (*args*)*handler* $\triangleq$
  let [*str*] = *args* in
  let *string* = *pure-string* (*str*) in 579
  let *remainder* = *no-leading-spaces* (*string*) in 451
  let *mk-* (*value*, -) = *strint* (*remainder*) in 584
  if *value* = nil
  then *wholeconv-exception*(NOT-A-SIGNED-WHOLE-NUMBER) *handler* 597
  else return *value*

The procedure `LengthInt`

The value of the call `LengthInt(int)` shall be the number of characters in the string representation of the value of `int`.

NOTE 2 — This value corresponds to the capacity of an array `str` which is of the minimum capacity needed to avoid truncation of the result in the call `WholeStr.IntToStr(int,str)`.

LengthInt : *Arguments* → *Exception-handler* $\xrightarrow{o}$ $\mathbb{N}$

LengthInt (*args*)*handler* $\triangleq$
  let [*int*] = *args* in
  let *string* = *intstr* (*int*) in 585
  let *length* = len *string* in
  get-whole-result(UNSIGNED-TYPE, *length*) *handler* 199

The procedure `ScanCard`

The call `ScanCard(inputCh,chClass,nextState)` shall assign values to `chClass` and `nextState` depending upon the value of `inputCh` as shown in the following table.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Procedure	inputCh	chClass	nextState a procedure with behaviour of
ScanCard	space	padding	ScanCard
	decimal digit	valid	*W*
	other	invalid	ScanCard
W	decimal digit	valid	*W*
	other	terminator	—

ScanCard : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

ScanCard (*args*)*handler* $\triangleq$
  let [*inputCh*, *chClass*, *nextState*] = *args* in
  (*isa-space* (*inputCh*) → (*assign*(*chClass*, *padding*) *handler*; 450 150
    assign(*nextState*, *ScanCard-pid*) *handler*), 150
   isa-numeral (*inputCh*) → (*assign*(*chClass*, *valid*) *handler*; 451 150
    assign(*nextState*, *W-pid*) *handler*), 150
   others → (*assign*(*chClass*, *invalid*) *handler*; 150
    assign(*nextState*, *ScanCard-pid*) *handler*)) 150

values

scancard-map = {*ScanCard-pid* ↦ *make-proper-procden* (*ScanCard*)} 339

annotations The mapping *scancard-map*, from elements of type *Procedure-id* to the corresponding procedure denotations, is part of the mapping *dens* used in the construction of the abstract implementation module — see annex B.

The procedure FormatCard

The value of the call FormatCard(str) shall be

strAllRight	if str has a value representing a complete unsigned whole number that is in the range of the type CARDINAL;
strOutOfRange	if str has a value representing a complete unsigned whole number that is not in the range of CARDINAL;
strWrongFormat	if str has a value with remaining characters that do not form a complete unsigned whole number;
strEmpty	if str has a value with no remaining characters;

ignoring any leading space characters in str.

FormatCard : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Enumeration-value*

FormatCard (*args*)*handler* $\triangleq$
  let [*str*] = *args* in
  let *string* = *pure-string* (*str*) in 579
  let *remainder* = *no-leading-spaces* (*string*) in 451
  let *mk-* (-, *result*) = *strcard* (*remainder*) in 585
  return *convenum* (*result*) 606

The procedure ValueCard

If str has a value representing an unsigned whole number, the value of the call ValueCard(str) shall be the CARDINAL value that corresponds to that number. Otherwise, the WholeConv exception shall be raised.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

ValueCard : *Arguments* → *Exception-handler* $\xrightarrow{o}$ $\mathbb{Z}$

ValueCard (*args*)*handler* $\triangleq$
  let [*str*] = *args* in
  let *string* = *pure-string* (*str*) in 579
  let *remainder* = *no-leading-spaces* (*string*) in 451
  let *mk-* (*value*, -) = *strcard* (*remainder*) in 585
  if *value* = nil
  then *wholeconv-exception*(NOT-AN-UNSIGNED-WHOLE-NUMBER) *handler* 597
  else return *value*

The procedure LengthCard

The value of the call LengthCard(card) shall be the number of characters in the string representation of the value of card.

NOTE 3 — This value corresponds to the capacity of an array str which is of the minimum capacity needed to avoid truncation of the result in the call WholeStr.CardToStr(card,str).

LengthCard : *Arguments* → *Exception-handler* $\xrightarrow{o}$ **N**

LengthCard (*args*)*handler* $\triangleq$
  let [*card*] = *args* in
  let *string* = *cardstr* (*int*) in 586
  let *length* = len *string* in
  get-whole-result(UNSIGNED-TYPE, *length*) *handler* 199

The procedure IsWholeConvException

The value of the call IsWholeConvException() shall be *true* if the calling coroutine is in the state of exceptional execution because of the raising of the WholeConv exception, and *false* otherwise.

IsWholeConvException : *Arguments* → *Exception-handler* $\xrightarrow{o}$ $\mathbb{B}$

IsWholeConvException (*args*)*handler* $\triangleq$
  def *source* = *wholeconv-source* in
  def *res* = *m-IsCurrentSource*([*source*]) in 386
  return *res*

9.5.3.1.3 Exceptions Raised by WholeConv

Following the detection of one of the exceptions defined for the module WholeConv, an implementation shall raise the WholeConv exception.

The character string associated with each of the exceptions raised by WholeConv shall be implementation-defined.

types

Wholeconvexception = NOT-A-SIGNED-WHOLE-NUMBER
  | NOT-AN-UNSIGNED-WHOLE-NUMBER

wholeconv-exception : *Wholeconvexception* → *Exception-handler* $\xrightarrow{o}$ ()

wholeconv-exception (*wce*)*handler* $\triangleq$
  def *source* = *wholeconv-source* in
  def *message* = *wholeconv-exception-string*(*wce*) in 598
  let *number* : **N** in
  m-RAISE([*source*, *number*, *message*]) *handler* 383

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wholeconv-exception-string : *Wholeconvexception* $\xrightarrow{o}$ *Message*

wholeconv-exception-string (*wce*) $\triangleq$
  implementation-defined

annotations The result is an implementation-defined string value.

9.5.3.1.4 The State for WholeConv

The state for the module `WholeConv` shall include a value of the exception source type identifying exceptions raised by the module.

state *WholeConv* of
  wholeconv-source : *Exception-source*
end

9.5.3.1.5 The Initialization of WholeConv

The initialization of the module `WholeConv` shall allocate a unique value of the exception source type to identify exceptions raised by the module.

wholeconv-initialization : () $\rightarrow$ *Exception-handler* $\xrightarrow{o}$ ()

wholeconv-initialization ()*handler* $\triangleq$
  def *source* = *allocate-source*() in
  if *source* $\neq$ nil
  then *wholeconv-source* := *source*
  else *mandatory-detection*(NO-EXCEPTION-SOURCE) *handler*

342

396

9.5.3.2 The Modules RealConv and LongConv

The modules `RealConv` and `LongConv` provide low-level string conversion procedures for values of the type `REAL` and values of the type `LONGREAL`. In the case of `RealConv`, real number parameters are of the type `REAL`. In the case of `LongConv`, real number parameters are of the type `LONGREAL`.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.5.3.2.1 The Definition Module of RealConv

```
DEFINITION MODULE RealConv;

  (* Low-level REAL/string conversions *)

IMPORT
  ConvTypes;

TYPE
  ConvResults = ConvTypes.ConvResults; (* strAllRight, strOutOfRange, strWrongFormat, strEmpty *)

PROCEDURE ScanReal (inputCh: CHAR; VAR chClass: ConvTypes.ScanClass;
                    VAR nextState: ConvTypes.ScanState);
  (* Represents the start state of a finite state scanner for real numbers - assigns
     class of inputCh to chClass and a procedure representing the next state to
     nextState.
   *)

PROCEDURE FormatReal (str: ARRAY OF CHAR): ConvResults;
  (* Returns the format of the string value for conversion to REAL. *)

PROCEDURE ValueReal (str: ARRAY OF CHAR): REAL;
  (* Returns the value corresponding to the real number string value str if str is
     well-formed; otherwise raises the RealConv exception.
  *)

PROCEDURE LengthFloatReal (real: REAL; sigFigs: CARDINAL): CARDINAL;
  (* Returns the number of characters in the floating-point string representation of
     real with sigFigs significant figures.
  *)

PROCEDURE LengthEngReal (real: REAL; sigFigs: CARDINAL): CARDINAL;
  (* Returns the number of characters in the floating-point engineering string
     representation of real with sigFigs significant figures.
  *)

PROCEDURE LengthFixedReal (real: REAL; place: INTEGER): CARDINAL;
  (* Returns the number of characters in the fixed-point string representation of real
     rounded to the given place relative to the decimal point.
  *)

PROCEDURE IsRConvException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional execution state because
     of the raising of the RealConv exception; otherwise returns FALSE.
  *)

END RealConv.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9.5.3.2.2 The Definition Module of LongConv

```
DEFINITION MODULE LongConv;

  (* Low-level LONGREAL/string conversions *)

IMPORT
  ConvTypes;

TYPE
  ConvResults = ConvTypes.ConvResults; (* strAllRight, strOutOfRange, strWrongFormat, strEmpty *)

PROCEDURE ScanReal (inputCh: CHAR; VAR chClass: ConvTypes.ScanClass;
                    VAR nextState: ConvTypes.ScanState);
  (* Represents the start state of a finite state scanner for real numbers - assigns
     class of inputCh to chClass and a procedure representing the next state to
     nextState.
  *)

PROCEDURE FormatReal (str: ARRAY OF CHAR): ConvResults;
  (* Returns the format of the string value for conversion to LONGREAL. *)

PROCEDURE ValueReal (str: ARRAY OF CHAR): LONGREAL;
  (* Returns the value corresponding to the real number string value str if str is
     well-formed; otherwise raises the LongConv exception.
  *)

PROCEDURE LengthFloatReal (real: LONGREAL; sigFigs: CARDINAL): CARDINAL;
  (* Returns the number of characters in the floating-point string representation of
     real with sigFigs significant figures.
  *)

PROCEDURE LengthEngReal (real: LONGREAL; sigFigs: CARDINAL): CARDINAL;
  (* Returns the number of characters in the floating-point engineering string
     representation of real with sigFigs significant figures.
  *)

PROCEDURE LengthFixedReal (real: LONGREAL; place: INTEGER): CARDINAL;
  (* Returns the number of characters in the fixed-point string representation of real
     rounded to the given place relative to the decimal point.
  *)

PROCEDURE IsRConvException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional execution state because
     of the raising of the LongConv exception; otherwise returns FALSE.
  *)

END LongConv.
```

9.5.3.2.3 The Procedures of RealConv and LongConv

The semantics of the two modules shall be the same, except that when module `RealConv` refers to the pervasive type `REAL`, the corresponding procedure in `LongConv` shall refer to the pervasive type `LONGREAL`.

NOTE 1 — The above statement is merely to avoid needless repetition of the semantics for `LongConv`.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The procedure `ScanReal`

A call of `ScanReal(inputCh,chClass,nextState)` shall assign values to `chClass` and `nextState` depending upon the value of `inputCh` as shown in the following table.

Procedure	`inputCh`	`chClass`	`nextState` a procedure with behaviour of
`ScanReal`	space	`padding`	`ScanReal`
	sign	`valid`	*RS*
	decimal digit	`valid`	*P*
	other	`invalid`	`ScanReal`
RS	decimal digit	`valid`	*P*
	other	`invalid`	*RS*
P	decimal digit	`valid`	*P*
	"."	`valid`	*F*
	"E"	`valid`	*E*
	other	`terminator`	—
F	decimal digit	`valid`	*F*
	"E"	`valid`	*E*
	other	`terminator`	—
E	sign	`valid`	*SE*
	decimal digit	`valid`	*WE*
	other	`invalid`	*E*
SE	decimal digit	`valid`	*WE*
	other	`invalid`	*SE*
WE	decimal digit	`valid`	*WE*
	other	`terminator`	—

ScanReal : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

ScanReal (*args*)*handler* $\triangleq$
  let [*inputCh*, *chClass*, *nextState*] = *args* in
  (*isa-space* (*inputCh*) → (*assign*(*chClass*, *padding*) *handler*; 450 150
    assign(*nextState*, *ScanReal-pid*) *handler*), 150
   isa-sign (*inputCh*) → (*assign*(*chClass*, *valid*) *handler*; 605 150
    assign(*nextState*, *RS-pid*) *handler*), 150
   isa-numeral (*inputCh*) → (*assign*(*chClass*, *valid*) *handler*; 451 150
    assign(*nextState*, *P-pid*) *handler*), 150
   others → (*assign*(*chClass*, *invalid*) *handler*; 150
    assign(*nextState*, *ScanReal-pid*) *handler*)) 150

RS : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

RS (*args*)*handler* $\triangleq$
  let [*inputCh*, *chClass*, *nextState*] = *args* in
  (*isa-numeral* (*inputCh*) → (*assign*(*chClass*, *valid*) *handler*; 451 150
    assign(*nextState*, *P-pid*) *handler*), 150
   others → (*assign*(*chClass*, *invalid*) *handler*; 150
    assign(*nextState*, *RS-pid*) *handler*)) 150

P : *Arguments* → *Exception-handler* $\xrightarrow{o}$ ()

P (*args*)*handler* $\triangleq$
  let [*inputCh*, *chClass*, *nextState*] = *args* in
  (*isa-numeral* (*inputCh*) → (*assign*(*chClass*, *valid*) *handler*; 451 150
    assign(*nextState*, *P-pid*) *handler*), 150
   inputCh = *decimal-point-character* → (*assign*(*chClass*, *valid*) *handler*; 150
    assign(*nextState*, *F-pid*) *handler*), 150

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  inputCh = exponent-character   → (assign(chClass, valid) handler;                150
                                    assign(nextState, E-pid) handler ),            150
  others                         → assign(chClass, terminator) handler; )        150
```

```
F : Arguments → Exception-handler ~o~> ()
F (args)handler ≙
  let [inputCh, chClass, nextState] = args in
  ( isa-numeral (inputCh)          → (assign(chClass, valid) handler;              451 150
                                      assign(nextState, F-pid) handler ),          150
    inputCh = exponent-character   → (assign(chClass, valid) handler;              150
                                      assign(nextState, E-pid) handler ),          150
    others                         → assign(chClass, terminator) handler; )       150
```

```
E : Arguments → Exception-handler ~o~> ()
E (args)handler ≙
  let [inputCh, chClass, nextState] = args in
  ( isa-sign (inputCh)             → (assign(chClass, valid) handler;              605 150
                                      assign(nextState, SE-pid) handler ),         150
    isa-numeral (inputCh)          → (assign(chClass, valid) handler;              451 150
                                      assign(nextState, WE-pid) handler ),         150
    others                         → (assign(chClass, invalid) handler;            150
                                      assign(nextState, E-pid) handler ))          150
```

```
SE : Arguments → Exception-handler ~o~> ()
SE (args)handler ≙
  let [inputCh, chClass, nextState] = args in
  ( isa-numeral (inputCh)          → (assign(chClass, valid) handler;              451 150
                                      assign(nextState, WE-pid) handler ),         150
    others                         → (assign(chClass, invalid) handler;            150
                                      assign(nextState, SE-pid) handler ))         150
```

```
WE : Arguments → Exception-handler ~o~> ()
WE (args)handler ≙
  let [inputCh, chClass, nextState] = args in
  ( isa-numeral (inputCh)          → (assign(chClass, valid) handler;              451 150
                                      assign(nextState, WE-pid) handler ),         150
    others                         → assign(chClass, terminator) handler; )       150
```

values

```
  scanreal-map = {ScanReal-pid ↦ make-proper-procden (ScanReal),                   339
                  RS-pid ↦ make-proper-procden (RS),                               339
                  P-pid ↦ make-proper-procden (P),                                 339
                  F-pid ↦ make-proper-procden (F),                                 339
                  E-pid ↦ make-proper-procden (E),                                 339
                  SE-pid ↦ make-proper-procden (SE),                               339
                  WE-pid ↦ make-proper-procden (WE)}                               339
```

annotations The mapping *scanreal-map*, from elements of type *Procedure-id* to the corresponding procedure denotations, is part of the mapping *dens* used in the construction of the abstract implementation module — see annex B.

The procedure `FormatReal`

The value of the call `FormatReal(str)` shall be

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

`strAllRight`	if `str` has a value representing a complete signed real number that is in the range of the real number type corresponding to the module;
`strOutOfRange`	if `str` has a value representing a complete signed real number that is not in the range of the real number type corresponding to the module;
`strWrongFormat`	if `str` has a value with remaining characters that do not form a complete signed real number;
`strEmpty`	if `str` has a value with no remaining characters;

ignoring any leading space characters in `str`.

FormatReal : *Arguments* → *Exception-handler* $\xrightarrow{o}$ *Enumeration-value*

FormatReal (*args*)*handler* $\triangleq$
  let [*str*] = *args* in
  let *string* = *pure-string* (*str*) in 579
  let *remainder* = *no-leading-spaces* (*string*) in 451
  let *mk-* (-, *result*) = *strreal* (*remainder*) in 589
  return *convenum* (*result*) 606

The procedure `ValueReal`

If `str` has a value representing a real number, the value of the call `ValueReal(str)` shall be the `REAL` (or `LONGREAL`) value that corresponds most closely to that number. Otherwise, an exception shall be raised.

ValueReal : *Arguments* → *Exception-handler* $\xrightarrow{o}$ $\mathbb{R}$

ValueReal (*args*)*handler* $\triangleq$
  let [*str*] = *args* in
  let *string* = *pure-string* (*str*) in 579
  let *remainder* = *no-leading-spaces* (*string*) in 451
  let *mk-* (*value*, -) = *strreal* (*remainder*) in 589
  if *value* = nil
  then *realconv-exception*(NOT-A-REAL-NUMBER) *handler* 605
  else return *value*

The procedure `LengthFloatReal`

The value of the call `LengthFloatReal(real,sigFigs)` shall be the number of characters in the floating-point string representation of the value of `real` when using `sigFigs` significant figures.

NOTE 2 — This value corresponds to the capacity of an array `str` which is of the minimum capacity needed to avoid truncation of the result in the call `RealStr.RealToFloat(real,sigFigs,str)` (in the case of `RealConv.LengthFloatReal`) or the call `LongStr.RealToFloat(real,sigFigs,str)` (in the case of `LongConv.LengthFloatReal`).

LengthFloatReal : *Arguments* → *Exception-handler* $\xrightarrow{o}$ $\mathbb{N}$

LengthFloatReal (*args*)*handler* $\triangleq$
  let [*real*, *sigfigs*] = *args* in
  let *string* = *floatstr* (*real*, *sigfigs*) in 590
  let *length* = len *string* in
  get-whole-result(UNSIGNED-TYPE, *length*) *handler* 199

The procedure `LengthEngReal`

The value of the call `LengthEngReal(real,sigFigs)` shall be the number of characters in the floating-point engineering string representation of the value of `real` when using `sigFigs` significant figures.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NOTE 3 — This value corresponds to the capacity of an array `str` which is of the minimum capacity needed to avoid truncation of the result in the call `RealStr.RealToEng(real,sigFigs,str)` (in the case of `RealConv.LengthEngReal`) or the call `LongStr.RealToEng(real,sigFigs,str)` (in the case of `LongConv.LengthEngReal`).

LengthEngReal : *Arguments* → *Exception-handler* $\xrightarrow{o}$ N

LengthEngReal (*args*)*handler* $\triangleq$
  let [*real*, *sigfigs*] = *args* in
  let *string* = *engstr* (*real*, *sigfigs*) in 590
  let *length* = len *string* in
  get-whole-result(UNSIGNED-TYPE, *length*) *handler* 199

The procedure `LengthFixedReal`

The value of the call `LengthFixedReal(real,place)` shall be the number of characters in the fixed-point string representation of the value of `real` when rounded to the place relative to the decimal point given by the value of `place`.

NOTE 4 — This value corresponds to the capacity of an array `str` which is of the minimum capacity needed to avoid truncation of the result in the call `RealStr.RealToFixed(real,place,str)` (in the case of `RealConv.LengthFixedReal`) or the call `LongStr.RealToFixed(real,place,str)` (in the case of `LongConv.LengthFixedReal`).

LengthFixedReal : *Arguments* → *Exception-handler* $\xrightarrow{o}$ N

LengthFixedReal (*args*)*handler* $\triangleq$
  let [*real*, *sigfigs*] = *args* in
  let *string* = *fixedstr* (*real*, *sigfigs*) in 591
  let *length* = len *string* in
  get-whole-result(UNSIGNED-TYPE, *length*) *handler* 199

The procedure `IsRConvException`

If the calling coroutine is in the state of exceptional execution because of the raising of the `RealConv` exception, the value of the call `RealConv.IsRConvException()` shall be *true*; otherwise the value shall be *false*.

If the calling coroutine is in the state of exceptional execution because of the raising of the `LongConv` exception, the value of the call `LongConv.IsRConvException()` shall be *true*; otherwise the value shall be *false*.

IsRConvException : *Arguments* → *Exception-handler* $\xrightarrow{o}$ $\mathbb{B}$

IsRConvException (*args*)*handler* $\triangleq$
  def *source* = *realconv-source* in
  def *res* = *m-IsCurrentSource*([*source*]) in 386
  return *res*

9.5.3.2.4 Exceptions Raised by RealConv and LongConv

Following the detection of one of the exceptions defined for the modules `RealConv`, the `RealConv` exception shall be raised.

Following the detection of one of the exceptions defined for the modules `LongConv`, the `LongConv` exception shall be raised.

The character string associated with each of the exceptions raised by `RealConv` and `LongConv` shall be implementation-defined.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

types

Realconvexception = NOT-A-REAL-NUMBER

realconv-exception : *Realconvexception* → *Exception-handler* $\xrightarrow{o}$ ()

realconv-exception (*rce*)*handler* $\triangleq$
  def *source* = *realconv-source* in
  def *message* = *realconv-exception-string*(*rce*) in 605
  let *number* : $\mathbb{N}$ in
  m-RAISE([*source*, *number*, *message*]) *handler* 383

realconv-exception-string : *Realconvexception* $\xrightarrow{o}$ *Message*

realconv-exception-string (*rce*) $\triangleq$
  implementation-defined

annotations The result is an implementation-defined string value.

9.5.3.2.5 The State for RealConv and LongConv

The state for each of the modules `RealConv` and `LongConv` shall include a value of the exception source type identifying exceptions raised by the corresponding module.

state *RealConv* of
  realconv-source : *Exception-source*
end

9.5.3.2.6 The Initialization of RealConv and LongConv

The initialization of the modules `RealConv` and `LongConv` shall allocate a unique value of the exception source type to identify exceptions raised by the module.

realconv-initialization : () → *Exception-handler* $\xrightarrow{o}$ ()

realconv-initialization ()*handler* $\triangleq$
  def *source* = *allocate-source*() in 342
  if *source* ≠ nil
  then *realconv-source* := *source*
  else *mandatory-detection*(NO-EXCEPTION-SOURCE) *handler* 396

9.5.4 Auxiliary Definitions

The following functions are used in the conversion operations

is-digits : *Char** → $\mathbb{B}$

is-digits (*s*) $\triangleq$
  $s \neq [] \wedge \forall c \in \text{elems } s \cdot c \in digits$

annotations Return whether *s* is not empty and consists entirely of digits

isa-sign : *Char* → $\mathbb{B}$

isa-sign (*c*) $\triangleq$
  $c = plus\text{-}character \vee c = minus\text{-}character$

annotations Return whether *c* is a sign character

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

is-optminus : $Char^* \to \mathbb{B}$

is-optminus $(s) \triangleq$
  len $s = 1 \wedge$ hd $s =$ *minus-character*

annotations Return whether s consists of a single minus character

digit-value : $Char \to \mathbf{N}$

digit-value $(c) \triangleq$
  scalar-value $(c) -$ *scalar-value* $('0')$ 210 210

annotations Return the value of a numeral.

numeric-value : $Char^+ \to \mathbf{N}$

numeric-value $(s) \triangleq$
  let $c =$ hd s in
  (*digit-value* $(c) \times 10 \uparrow$ len tl $s) +$ if tl $s = [\,]$ then 0 else *numeric-value* (tl s) 606 606

annotations The numeric value of a string of numerals.

fixed-point-value : $Char^+ \times Char^* \to \mathbf{N}$

fixed-point-value $(w, f) \triangleq$
  let $wn =$ *numeric-value* (w) in 606
  let $fn =$ if $f = [\,]$ then 0 else *numeric-value* $(f) \times 10.0 \uparrow -$len f in 606
  $wn + fn$

annotations The numeric value of a string of numerals w giving the whole-number part and f giving the optional fractional part.

The following definitions give the enumeration values used in the formal model.

values

  convresults = [`strAllRight`, `strOutOfRange`, `strWrongFormat`, `strEmpty`];

  strAllRight = *mk-Enumeration-value* (`strAllRight`, *convresults*);

  strOutOfRange = *mk-Enumeration-value* (`strOutOfRange`, *convresults*);

  strWrongFormat = *mk-Enumeration-value* (`strWrongFormat`, *convresults*);

  strEmpty = *mk-Enumeration-value* (`strEmpty`, *convresults*)

convenum : *Conv-result* → *Enumeration-value*

convenum $(cr) \triangleq$
  cases cr :
    ALL-RIGHT → *strAllRight*,
    OUT-OF-RANGE → *strOutOfRange*,
    WRONG-FORMAT → *strWrongFormat*,
    EMPTY → *strEmpty*
  end

values

  scanclass = [`padding`, `valid`, `invalid`, `terminator`];

  padding = *mk-Enumeration-value* (`padding`, *scanclass*);

  valid = *mk-Enumeration-value* (`valid`, *scanclass*);

  invalid = *mk-Enumeration-value* (`invalid`, *scanclass*);

  terminator = *mk-Enumeration-value* (`terminator`, *scanclass*)

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9.6 The Module SysClock

The module SysClock provides facilities for accessing a system clock that records the date and time of day.

9.6.1 The Definition Module of SysClock

```
DEFINITION MODULE SysClock;

(* Facilities for accessing a system clock that records the date and time of day *)

CONST
  maxSecondParts = <implementation-defined integral value>;

TYPE
  Month    = [1 .. 12];
  Day      = [1 .. 31];
  Hour     = [0 .. 23];
  Min      = [0 .. 59];
  Sec      = [0 .. 59];
  Fraction = [0 .. maxSecondParts];
  UTCDiff  = [-780 .. 720];
  DateTime =
    RECORD
      year:       CARDINAL;
      month:      Month;
      day:        Day;
      hour:       Hour;
      minute:     Min;
      second:     Sec;
      fractions:  Fraction;      (* parts of a second *)
      zone:       UTCDiff;       (* Time zone differential factor which is the number
                                    of minutes to add to local time to obtain UTC. *)
      summerTimeFlag: BOOLEAN;   (* Interpretation of flag depends on local usage. *)
    END;

PROCEDURE CanGetClock (): BOOLEAN;
  (* Returns TRUE if a system clock can be read; FALSE otherwise *)

PROCEDURE CanSetClock (): BOOLEAN;
  (* Returns TRUE if a system clock can be set; FALSE otherwise *)

PROCEDURE IsValidDateTime (userData: DateTime): BOOLEAN;
  (* Returns TRUE if the value of userData represents a valid date and time; FALSE otherwise *)

PROCEDURE GetClock (VAR userData: DateTime);
  (* If possible, assigns system date and time of day to userData *)

PROCEDURE SetClock (userData: DateTime);
  (* If possible, sets the system clock to the values of userData *)

END SysClock.
```

NOTES

1 No provision is made for leap seconds.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2 'UTC' is 'Universal Coordinated Time'. This is the correct international designation for what was once called GMT (Greenwich Mean Time).

3 The field **summerTimeFlag** is present for information only. UTC can always be obtained by subtracting the **UTCDiff** value from the time data, regardless of the value of the **summerTimeFlag**. However, its presence does allow a program to know whether or not the date and time data represents standard time for that location, or 'summer time'. A program could therefore be written to change the system clock to summer time automatically on a certain date, provided it had not already been changed.

9.6.2 The Types of SysClock

On a processor where `MAX(CARDINAL)` is larger than 2048, the value of the `year` field of the type denoted by `DateTime` shall directly encode the number of the year. On processors where the range of the type `CARDINAL` is insufficient to achieve this, the interpretation of the value of this field and its mapping to some range of actual years shall be implementation-defined.

9.6.3 The Procedures of SysClock

The procedure `CanGetClock`

The value of the call

```
CanGetClock()
```

shall be an implementation-defined Boolean value. If the value *true* is returned, there shall be a system clock which the program is permitted to read.

$CanGetClock : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} \mathbb{B}$

$CanGetClock\ (args)handler \triangleq$
implementation-defined value

The procedure `CanSetClock`

The value of the call

```
CanSetClock()
```

shall be an implementation-defined Boolean value. If the value *true* is returned, there shall be a system clock which the program is permitted to set.

$CanSetClock : Arguments \rightarrow Exception\text{-}handler \xrightarrow{o} \mathbb{B}$

$CanSetClock\ (args)handler \triangleq$
implementation-defined value

The procedure `IsValidDateTime`

The value of the call

```
IsValidDateTime(userData)
```

shall be *true* if the value of `userData` represents a valid date and time, and shall be *false* otherwise.

NOTE 1 — Only the date components of **userData** need to be validated since all combinations of values of the time components are are known to be valid.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

IsValidDateTime : *Arguments* → *Exception-handler* $\xrightarrow{o}$ $\mathbb{B}$

IsValidDateTime (*args*)*handler* $\triangleq$
  let [*mk-DateTime* (*year*, *month*, *day*, -, -, -, -, -)] = *args* in
  return *is-valid-date* (*year*, *month*, *day*) 610

The procedure `GetClock`

The call

```
GetClock(userData)
```

shall assign values for each field of the variable `userData` for which information is available. Each of the remaining fields of `userData` shall be set to zero, where this is a valid value, and otherwise shall be set to the lower bound of the range of allowed values.

GetClock : *Arguments* × *Exception-handler* $\xrightarrow{o}$ *Value*

GetClock (*args*)*handler* $\triangleq$
  let [*userData*] = *args* in
  assign(*userData*, *clock*) *handler* 150

The procedure `SetClock`

The call

```
SetClock(userData)
```

shall set the system clock to the date and time specified by `userData`, provided that the program may set the system clock, and that the value of `userData` represents a valid date and time. If the program cannot set the system clock, a call of `SetClock` shall have no effect.

NOTE 2 — The effect of a call of `SetClock` is implementation-dependent if it is permitted to set the system clock, but an invalid date and time is given.

SetClock : *Arguments* × *Exception-handler* $\xrightarrow{o}$ ()

SetClock (*args*)*handler* $\triangleq$
  def *canset* = *CanSetClock*([]) *handler* in 608
  if *canset*
  then let [*userData*] = *args* in
    clock := *userData*
  else skip

annotations The invariant on the state for SysClock requires *val* to be a valid time.

9.6.4 The State for SysClock

Domains

values

MAXSECONDPARTS : N = implementation-defined

types

Month = N
inv *m* $\triangleq$ *m* ∈ {1, . . . , 12};

Day = N
inv *d* $\triangleq$ *d* ∈ {1, . . . , 31};

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

$Hour = \mathbf{N}$

inv $h \triangleq h \in \{0, \ldots, 23\}$;

$Min = \mathbf{N}$

inv $m \triangleq m \in \{0, \ldots, 59\}$;

$Sec = \mathbf{N}$

inv $s \triangleq s \in \{0, \ldots, 59\}$;

$Fraction = \mathbf{N}$

inv $f \triangleq f \in \{0, \ldots, MAXSECONDPARTS\}$;

$UTCDiff = \mathbb{Z}$

inv $u \triangleq u \in \{-780, \ldots, 720\}$;

$DateTime :: year : \mathbf{N}$
$month : Month$
$day : Day$
$hour : Hour$
$minute : Min$
$second : Sec$
$fraction : Fraction$
$zone : UTCDiff$
$summertimeflag : \mathbb{B}$

state *SysClock* of
$clock : DateTime$
inv $mk\text{-}SysClock(clock) \triangleq$
let $mk\text{-}DateTime(year, month, day, -, -, -, -, -) = clock$ in
$is\text{-}valid\text{-}date\ (year, month, day)$
end

610

$is\text{-}valid\text{-}date : \mathbf{N} \times Month \times Day \to \mathbb{B}$

$is\text{-}valid\text{-}date\ (y, m, d) \triangleq$
$(m \in \{1, 3, 5, 7, 8, 10, 12\} \wedge d \in \{1, \ldots, 31\}) \vee$
$(m \in \{4, 6, 9, 11\} \wedge d \in \{1, \ldots, 30\}) \vee$
$(m = 2 \wedge \neg\ is\text{-}leap\text{-}year(y) \wedge d \in \{1, \ldots, 28\}) \vee$
$(m = 2 \wedge is\text{-}leap\text{-}year(y) \wedge d \in \{1, \ldots, 29\})$

$is\text{-}leap\text{-}year : \mathbf{N} \to \mathbb{B}$

$is\text{-}leap\text{-}year\ (y) \triangleq$
$(y \bmod 4 = 0) \wedge (y \bmod 100 \neq 0) \vee (y \bmod 400 = 0)$

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Section 10 : Auxiliary Formal Model Definitions

10.1 Extensions to VDM-SL

The formal definition of Modula-2 in this part of ISO/IEC 10514 uses extensions to VDM-SL that are defined in this clause. The intention of introducing these extensions is to simplify the definition by allowing an operational reading of the dynamic semantics.

10.1.1 Replacement Definitions for Combinators

Operation signatures written in the formal definition as follows

$$M : Command \to Environment \xrightarrow{o}$$

are to be interpreted as function signatures written as:

$$M : Command \to Environment \to CmdCont \to CmdCont$$

If t is a transformation on command continuations, then

$$t : CmdCont \to CmdCont$$
$$t() \triangleq \lambda c \cdot \lambda \sigma \cdot t(c)\sigma$$

Operation signatures written in the formal definition as follows

$$E : Command \to Environment \to Value \xrightarrow{o}$$

are to be interpreted as function signatures written as:

$$E : Command \to Environment \to Value \to CmdCont \to CmdCont$$

If t is a transformation on expression continuations, then

$$t : ExprCont \to CmdCont$$
$$t() \triangleq \lambda k \cdot \lambda \sigma \cdot t(k)\sigma$$

The semicolon combinator is defined as:

$$t1; t2 \triangleq t1 \circ t2$$

The definition statement is defined as:

$$t_v : CmdCont \to CmdCont$$
$$e : ExprCont \to CmdCont$$
$$\mathsf{def}\ v = e\ \mathsf{in}\ t_v \triangleq e(\lambda v \cdot t_v)$$

Explanation:

$$f : CmdCont \to CmdCont$$
$$f(c) \triangleq e(\lambda v \cdot t_v(c))$$

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

$e : ExprCont \to CmdCont$
$t_v : CmdCont \to CmdCont$

let $c \in CmdCont$	then	$t_v(c) \in CmdCont$
	and	$\lambda\, v \cdot t_v(c) \in ExprCont$
	therefore	$e(\lambda\, v \cdot t_v(c)) \in CmdCont$

For expression continuations we have:

$t_v : ExprCont \to CmdCont$
$e : ExprCont \to CmdCont$
def $v = e$ in $t_v \triangleq e(\lambda\, v \cdot t_v)$
let $v = e$ in $e_v \triangleq (\lambda\, v \cdot e_v)(e)$

Explanation:

$f : ExprCont \to CmdCont$
$f\,(k) \triangleq e(\lambda\, v \cdot t_v(k))$

$e : ExprCont \to CmdCont$
$t_v : ExprCont \to CmdCont$

let $k \in ExprCont$	then	$t_v(k) \in CmdCont$
	and	$\lambda\, v \cdot t_v(k) \in ExprCont$
	therefore	$e(\lambda\, v \cdot t_v(k)) \in CmdCont$

To convert a value into a continuation

$return : Value \to (ExpCont \to CmdCont)$
$return\,() \triangleq \lambda\, k \cdot k(v)$

$skip : CmdCont \to CmdCont$
$skip\,() \triangleq \lambda\, c \cdot c$

10.1.2 The McCarthy Conditional Expression

The syntax of the McCarthy conditional expression used in this part of ISO/IEC 10514 is:

McCarthy expression = "(" guarded expression list, [",", other expression], ")" ;

guarded expression list = guarded expression, { "," guarded expression } :

guarded expression = expression, "→", expression ;

other expression = "other", expression ;

The semantics of the McCarthy expression are given by translating it into an if expression. The syntax of the if expression is:

if expression = "if", expression, "then", expression
{ elseif expression }, "else" expression ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

elseif expression = "elseif", expression, "then", expression ;

The syntax of the if expression can be rewritten as follows:

if expression = guarded if expression list, else expression ;

guarded if expression list = guarded if expression, guarded elseif expression ;

guarded if expression = "if", expression, "then", expression ;

guarded elseif expressions = "elseif", expression, "then", expression ;

else expression = "else", "error" | "else" expression − ("error") ;

The value error denotes the VDM-SL semantic value ⊥. Evaluation of this value would imply that no meaning can be given to the corresponding Modula-2 program by this part of ISO/IEC 10514.

The McCarthy expression can be translated into an equivalent if expression by the following steps:

Step 1. — The production McCarthy expression is translated into the if expression as follows:

the "(" and ")" terminals are deleted;

the guarded expression list is translated into a guarded if expression list;

if an other expression is present, it is translated into an else expression; otherwise a "else", "error" is appended.

Step 2. — The production guarded expression list is translated into the guarded if expression list as follows:

any "," terminals are deleted;

the first guarded expression is translated into a guarded if expression; and

if any subsequent guarded expression is present, each is translated into a guarded elseif expression.

Step 3. — The first guarded expression is translated into a guarded if expression as follows:

the terminal "if" is added;

the first expression copied;

the "→" terminal is replaced by a "then" terminal; and

the second expression is copied.

Step 4. — Subsequent occurrences of guarded expression are each translated into a guarded elesif expression as follows:

the terminal "elseif" is added;

the first expression copied;

the "→" terminal is replaced by a "then" terminal; and

the second expression is copied.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

10.2 VDM-SL Functions

In this clause 'D', 'R' and 'X' are used to denote arbitrary types

10.2.1 Set functions

$is\text{-}disj : (X\text{-set})\text{-set} \to \mathbb{B}$
$is\text{-}disj\,(ss) \triangleq \forall s, t \in ss \cdot s = t \vee s \cap t = \{\}$

annotations Check if a family of sets is pairwise disjoint

$applys : (D \to R) \times D\text{-set} \to R\text{-set}$
$applys\,(f, s) \triangleq \{f\,(e) \mid e \in s\}$

annotations Apply the function f to every member of the set s

$mins\,(s : X\text{-set})r : X$
pre $s \neq \{\}$
post $r \in s \wedge \forall x \in s \cdot r \leq x$

annotations Return the smallest element of a set that has an order relation defined on it

$maxs\,(s : X\text{-set})r : X$
pre $s \neq \{\}$
post $r \in s \wedge \forall x \in s \cdot r \geq x$

annotations Return the largest element of a set that has an order relation defined on it

10.2.2 Sequence functions

$is\text{-}suffix : X^+ \times X^+ \to \mathbb{B}$
$is\text{-}suffix\,(l, m) \triangleq \exists i \in \mathsf{inds}\ m \cdot l = m(i, \ldots, \mathsf{len}\ m)$

$is\text{-}uniques : X^* \to \mathbb{B}$
$is\text{-}uniques\,(l) \triangleq \forall i, j \in \mathsf{inds}\ l \cdot i = j \vee l\,(i) \neq l\,(j)$

annotations Check if all elements of a sequence are distinct.

$rev : X^* \to X^*$
$rev\,(l) \triangleq \mathsf{if}\ l = [\,]\ \mathsf{then}\ [\,]\ \mathsf{else}\ rev\,(\mathsf{tl}\ l) \curvearrowright [\mathsf{hd}\ l]$ 614

annotations Reverse a sequence.

$front : X^* \times \mathbb{N}_1 \to X^*$
$front\,(l, n) \triangleq l\,(1, \ldots, n)$
pre $n \leq \mathsf{len}\ l$

annotations Return the first n elements of a sequence

$rest : X^* \times \mathbb{N}_1 \to X^*$
$rest\,(l, i) \triangleq l\,(i, \ldots, \mathsf{len}\ l)$
pre $i \leq \mathsf{len}\ l$

annotations Return a sequence consisting of the ith through to the last element of a sequence

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

$firstocc : X \times X^* \to \mathbb{N}$

$firstocc\,(x, l) \triangleq$
  let $els = \text{elems } l$ in
  if $x \notin els$
  then 0
  else $mins\,(\{i \in \text{inds } l \mid x = l(i)\})$ 614

annotations Return the index of the first occurrence of x in l or 0 if x is not an element of the sequence.

$del : X^* \times \mathbb{N} \to X^*$

$del\,(l, i) \triangleq front\,(l, i-1) \frown rest\,(l, i+1)$ 614 614
pre $i \in \text{inds } l$

annotations Remove the ith element of a sequence.

10.2.3 Map functions

$map\text{-}to\text{-}seq : (D \xrightarrow{m} R) \to R^*$

$map\text{-}to\text{-}seq\,(m) \triangleq$
  if $m = \{\mapsto\}$
  then $[\,]$
  else let $n = mins\,(\text{dom } m)$ in 614
    $[m\,(n)] \frown map\text{-}to\text{-}seq\,(m \lhd \{n\})$ 615

annotations Convert a map to a sequence starting with the smallest element in the domain.

$seq\text{-}to\text{-}map : \mathbb{N} \times X^* \to (\mathbb{N} \xrightarrow{m} X)$

$seq\text{-}to\text{-}map\,(n, S) \triangleq \{(x - 1 + n) \mapsto S(x) \mid x \in \text{inds } S\}$

annotations Convert a sequence to a map with n as the smallest element in the domain.

10.2.4 Numerical functions

$sign : \mathbb{R} \to \mathbb{R}$

$sign\,(x) \triangleq$
  $(x > 0 \to +1,$
  $x = 0 \to 0,$
  $x < 0 \to -1)$

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Annex A
(normative)
Minimum Limit Specifications

A.1 The Limit-specification Generators

The following generator-programs are specifications expressed in Modula-2 of programs which specify certain attributes for which minimum limits are required and which specify the minimum limits for those attributes. Thus, if a generator program was to be executed, its output would be a program which must be correctly translated and executed by an implementation.

With imported identifers having the meaning specified by this part of ISO/IEC 10514 in section 9, an implementation shall successfully translate and execute the programs that are generated by execution of the Modula-2 programs in the following subclauses.

NOTES

1 There is no requirement upon a conforming implementation that the generator programs in this annex be executable by that implementation.

2 The generated programs are themselves specifications (of attributes and minimum limits) and should not be regarded as test programs.

3 The generator programs are parameterized by a constant (always called `minLimit`) which define attributes' minimum limits.

A.1.1 Length of Identifiers

The following program generates a program to specify the *length of identifiers* attribute and that the minimum limit for the attribute be 32.

```
MODULE IdentLength;
IMPORT STextIO;

CONST
  minLimit = 32;

PROCEDURE GenName (n: CARDINAL);
VAR
  i : CARDINAL;
BEGIN
  FOR i := 1 TO n DO
    STextIO.WriteString ('w');
  END;
END GenName;

BEGIN
  STextIO.WriteString ("MODULE LimitIdentLength;");
  STextIO.WriteLn;
  STextIO.WriteString ("VAR ");
  STextIO.WriteLn;
  GenName (minLimit-1);
  STextIO.WriteString ("x");
  STextIO.WriteString (": INTEGER;");
  STextIO.WriteLn;
  GenName (minLimit-1);
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  STextIO.WriteString ("y");
  STextIO.WriteString (": INTEGER;");
  STextIO.WriteLn;
  STextIO.WriteString ("BEGIN");
  STextIO.WriteLn;
  STextIO.WriteString ("END LimitIdentLength.");
  STextIO.WriteLn;
END IdentLength.
```

A.1.2 Length of String Literals

The following program generates a program to specify the *length of string literals* attribute and that the minimum limit for the attribute be 32.

```
MODULE StringLength;
IMPORT STextIO;

CONST
  minLimit = 32;

PROCEDURE GenName (n: CARDINAL);
VAR
  i: CARDINAL;
BEGIN
  FOR i := 1 TO n DO
    STextIO.WriteString ('w');
  END;
END GenName;

BEGIN
  STextIO.WriteString ("MODULE LimitStringLength;");
  STextIO.WriteLn;
  STextIO.WriteString ("CONST");
  STextIO.WriteLn;
  STextIO.WriteString ("X = '");
  GenName (minLimit);
  STextIO.WriteString ("';");
  STextIO.WriteLn;
  STextIO.WriteString ("BEGIN");
  STextIO.WriteLn;
  STextIO.WriteString ("END LimitStringLength.");
  STextIO.WriteLn;
END StringLength.
```

A.1.3 Size of Single Dimension Arrays

The following program generates a program to specify the *size of single dimension arrays* attribute and that the minimum limit for the attribute be 4096.

```
MODULE SingleDimArray;
IMPORT SWholeIO;
IMPORT STextIO;

CONST
  minLimit = 4096;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
BEGIN
  STextIO.WriteString ("MODULE LimitSingleDimArray;");
  STextIO.WriteLn;
  STextIO.WriteString ("IMPORT SYSTEM;");
  STextIO.WriteLn;
  STextIO.WriteString ("CONST");
  STextIO.WriteLn;
  STextIO.WriteString ("minLimit =");
  SWholeIO.WriteCard (minLimit, 0);
  STextIO.WriteString (";");
  STextIO.WriteLn;
  STextIO.WriteString ("TYPE");
  STextIO.WriteLn;
  STextIO.WriteString ("A = ARRAY ");
  STextIO.WriteString ("[1..minLimit] OF SYSTEM.WORD;");
  STextIO.WriteLn;
  STextIO.WriteString ("VAR");
  STextIO.WriteLn;
  STextIO.WriteString ("a: A;");
  STextIO.WriteLn;
  STextIO.WriteString ("BEGIN");
  STextIO.WriteLn;
  STextIO.WriteString ("END LimitSingleDimArray.");
  STextIO.WriteLn;
END SingleDimArray.
```

A.1.4 Number of Dimensions of Multidimension Arrays

The following program generates a program to specify the *number of dimensions of multidimension arrays* attribute and that the minimum limit for the attribute be 16.

```
MODULE MultiDimArray;
IMPORT STextIO;

CONST
  minLimit = 16;
VAR
  j: CARDINAL;
BEGIN
  STextIO.WriteString ("MODULE LimitMultiDimArray; ");
  STextIO.WriteLn;
  STextIO.WriteString ("IMPORT SYSTEM;");
  STextIO.WriteLn;
  STextIO.WriteString ("TYPE");
  STextIO.WriteLn;
  STextIO.WriteString ("A = ARRAY ");
  STextIO.WriteLn;
  FOR j := 1 TO minLimit - 1 DO
    STextIO.WriteString ("[1..1]");
    STextIO.WriteLn;
    STextIO.WriteString (",");
  END;
  STextIO.WriteString ("[1..1] OF SYSTEM.WORD;");
  STextIO.WriteLn;
  STextIO.WriteString ("VAR");
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  STextIO.WriteLn;
  STextIO.WriteString ("a: A;");
  STextIO.WriteLn;
  STextIO.WriteString ("BEGIN");
  STextIO.WriteLn;
  STextIO.WriteString ("END LimitMultiDimArray.");
  STextIO.WriteLn;
END MultiDimArray.
```

A.1.5 Depth of Nested Records

The following program generates a program to specify the *depth of nested records* attribute and that the minimum limit for the attribute be 16.

```
MODULE NestedRecords;
IMPORT SWholeIO;
IMPORT STextIO;

CONST
  minLimit = 16;

PROCEDURE GenRec (n: CARDINAL);
BEGIN
  STextIO.WriteString ("RECORD");
  STextIO.WriteLn;
  STextIO.WriteString ("x: SYSTEM.WORD;");
  STextIO.WriteLn;
  IF n > 0 THEN
    STextIO.WriteString ("T");
    SWholeIO.WriteCard (n, 1);
    STextIO.WriteString (":");
    GenRec (n-1);
  END;
  STextIO.WriteString ("END;");
  STextIO.WriteLn;
END GenRec;

BEGIN
  STextIO.WriteString ("MODULE LimitNestedRecords;");
  STextIO.WriteLn;
  STextIO.WriteString ("IMPORT SYSTEM;");
  STextIO.WriteLn;
  STextIO.WriteString ("TYPE");
  STextIO.WriteLn;
  STextIO.WriteString ("R = ");
  STextIO.WriteLn;
  GenRec (minLimit);
  STextIO.WriteString ("VAR");
  STextIO.WriteLn;
  STextIO.WriteString ("r: R;");
  STextIO.WriteLn;
  STextIO.WriteString ("BEGIN");
  STextIO.WriteLn;
  STextIO.WriteString ("END LimitNestedRecords.");
  STextIO.WriteLn;
END NestedRecords.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

A.1.6 Number of Sequential Local Modules

The following program generates a program to specify the *number of sequential local modules* attribute and that the minimum limit for the attribute be 16.

```
MODULE ModuleSeq;
IMPORT SWholeIO;
IMPORT STextIO;

CONST
  minLimit = 16;
VAR
  j: CARDINAL;

BEGIN
  STextIO.WriteString ("MODULE LimitModuleSeq;");
  STextIO.WriteLn;
  FOR j := 1 TO minLimit DO
    STextIO.WriteString ("MODULE M");
    SWholeIO.WriteCard (j, 1);
    STextIO.WriteString (";");
    STextIO.WriteLn;
    STextIO.WriteString ("VAR");
    STextIO.WriteLn;
    STextIO.WriteString ("i");
    SWholeIO.WriteCard (j, 1);
    STextIO.WriteString (": INTEGER;");
    STextIO.WriteLn;
    STextIO.WriteString ("BEGIN");
    STextIO.WriteLn;
    STextIO.WriteString ("EXCEPT");
    STextIO.WriteLn;
    STextIO.WriteString ("FINALLY");
    STextIO.WriteLn;
    STextIO.WriteString ("EXCEPT");
    STextIO.WriteLn;
    STextIO.WriteString ("END M");
    SWholeIO.WriteCard (j, 1);
    STextIO.WriteString (";");
    STextIO.WriteLn;
  END;
  STextIO.WriteString ("BEGIN");
  STextIO.WriteLn;
  STextIO.WriteString ("END LimitModuleSeq.");
  STextIO.WriteLn;
END ModuleSeq.
```

A.1.7 Depth of Nested Local Modules

The following program generates a program to specify the *depth of nested local modules* attribute and that the minimum limit for the attribute be 16.

```
MODULE NestedModules;
IMPORT SWholeIO;
IMPORT STextIO;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
CONST
  minLimit = 16;

PROCEDURE ModGen (n: CARDINAL);
BEGIN
  IF n <> 0 THEN
    STextIO.WriteString ("MODULE M");
    SWholeIO.WriteCard (n, 1);
    STextIO.WriteString (";");
    STextIO.WriteLn;
    STextIO.WriteString ("VAR");
    STextIO.WriteLn;
    STextIO.WriteString ("i");
    SWholeIO.WriteCard (n, 1);
    STextIO.WriteString (": INTEGER;");
    STextIO.WriteLn;
    ModGen (n-1);
    STextIO.WriteString ("BEGIN");
    STextIO.WriteLn;
    STextIO.WriteString ("EXCEPT");
    STextIO.WriteLn;
    STextIO.WriteString ("FINALLY");
    STextIO.WriteLn;
    STextIO.WriteString ("EXCEPT");
    STextIO.WriteLn;
    STextIO.WriteString ("END M");
    SWholeIO.WriteCard (n, 1);
    STextIO.WriteString (";");
    STextIO.WriteLn;
  END;
END ModGen;

BEGIN
  STextIO.WriteString ("MODULE LimitNestedModules;");
  STextIO.WriteLn;
  ModGen (minLimit);
  STextIO.WriteString ("BEGIN");
  STextIO.WriteLn;
  STextIO.WriteString ("END LimitNestedModules.");
  STextIO.WriteLn;
END NestedModules.
```

A.1.8 Number of Enumeration Literals

The following program generates a program to specify the *number of enumeration literals* attribute and that the minimum limit for the attribute be 256.

```
MODULE Enumerations;
IMPORT SWholeIO;
IMPORT STextIO;

CONST
  minLimit = 256;
VAR
  j : CARDINAL;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
BEGIN
  STextIO.WriteString ("MODULE LimitEnumerations;");
  STextIO.WriteString ("TYPE");
  STextIO.WriteLn;
  STextIO.WriteString ("E = (");
  STextIO.WriteLn;
  FOR j := 1 TO minLimit-1 DO
    STextIO.WriteString ("e");
    SWholeIO.WriteCard (j, 1);
    STextIO.WriteString (",");
    STextIO.WriteLn;
  END;
  STextIO.WriteString ("eLast");
  STextIO.WriteLn;
  STextIO.WriteString (");");
  STextIO.WriteLn;
  STextIO.WriteString ("BEGIN");
  STextIO.WriteLn;
  STextIO.WriteString ("END LimitEnumerations.");
  STextIO.WriteLn;
END Enumerations.
```

A.1.9 Number of Sequential Procedure Declarations

The following program generates a program to specify the *number of sequential procedure declarations* attribute and that the minimum limit for the attribute be 16.

```
MODULE ProcedureSeq;
IMPORT SWholeIO;
IMPORT STextIO;

CONST
  minLimit = 16;
VAR
  j: CARDINAL;

BEGIN
  STextIO.WriteString ("MODULE LimitProcedureSeq;");
  STextIO.WriteLn;
  FOR j := 1 TO minLimit DO
    STextIO.WriteString ("PROCEDURE P");
    SWholeIO.WriteCard (j, 1);
    STextIO.WriteString ("; ");
    STextIO.WriteLn;
    STextIO.WriteString ("VAR");
    STextIO.WriteLn;
    STextIO.WriteString ("i");
    SWholeIO.WriteCard (j, 1);
    STextIO.WriteString (": INTEGER;");
    STextIO.WriteLn;
    STextIO.WriteString ("BEGIN");
    STextIO.WriteLn;
    STextIO.WriteString ("EXCEPT");
    STextIO.WriteLn;
    STextIO.WriteString ("END P");
    SWholeIO.WriteCard (j, 1);
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
    STextIO.WriteString (";");
    STextIO.WriteLn;
  END;
  STextIO.WriteString ("BEGIN");
  STextIO.WriteLn;
  STextIO.WriteString ("END LimitProcedureSeq.");
  STextIO.WriteLn;
END ProcedureSeq.
```

A.1.10 Depth of Nested Procedure Declarations

The following program generates a program to specify the *depth of nested procedure declarations* attribute and that the minimum limit for the attribute be 16.

```
MODULE NestedProcedures;
IMPORT SWholeIO;
IMPORT STextIO;

CONST
  minLimit = 16;

PROCEDURE ProcGen (n: CARDINAL);
BEGIN
  IF n <> 0 THEN
    STextIO.WriteString ("PROCEDURE P");
    SWholeIO.WriteCard (n, 1);
    STextIO.WriteString (";");
    STextIO.WriteLn;
    STextIO.WriteString ("VAR");
    STextIO.WriteLn;
    STextIO.WriteString ("i");
    SWholeIO.WriteCard (n, 1);
    STextIO.WriteString (": INTEGER;");
    STextIO.WriteLn;
    ProcGen(n-1);
    STextIO.WriteString ("BEGIN");
    STextIO.WriteLn;
    STextIO.WriteString ("EXCEPT");
    STextIO.WriteLn;
    STextIO.WriteString ("END P");
    SWholeIO.WriteCard (n, 1);
    STextIO.WriteString (";");
    STextIO.WriteLn;
  END;
END ProcGen;

BEGIN
  STextIO.WriteString ("MODULE LimitNestedProcedures;");
  STextIO.WriteLn;
  ProcGen (minLimit);
  STextIO.WriteString ("BEGIN");
  STextIO.WriteLn;
  STextIO.WriteString ("END LimitNestedProcedures.");
  STextIO.WriteLn;
END NestedProcedures.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

A.1.11 Number of Coroutines

The following program generates a program to specify the *number of coroutines* attribute and that the minimum limit for the attribute be 16.

```
MODULE Coroutines;
IMPORT SWholeIO;
IMPORT STextIO;

CONST
  minLimit = 16;
VAR
  i: CARDINAL;

BEGIN
  STextIO.WriteString ("MODULE LimitCoroutines;");
  STextIO.WriteLn;
  STextIO.WriteString ("IMPORT SYSTEM;");
  STextIO.WriteLn;
  STextIO.WriteString ("IMPORT COROUTINES;");
  STextIO.WriteLn;
  STextIO.WriteString ("CONST");
  STextIO.WriteLn;
  STextIO.WriteString ("wspsize = 400;");
  STextIO.WriteLn;
  STextIO.WriteString ("VAR");
  STextIO.WriteLn;
  STextIO.WriteString ("i: CARDINAL;");
  STextIO.WriteLn;
  FOR i := 1 TO minLimit DO
    STextIO.WriteString ("wsp");
    SWholeIO.WriteCard (i, 1);
    STextIO.WriteString (": ARRAY[1..wspsize] OF SYSTEM.WORD;");
    STextIO.WriteLn;
  END;
  STextIO.WriteString ("main: COROUTINES.COROUTINE;");
  STextIO.WriteLn;
  FOR i := 1 TO minLimit DO
    STextIO.WriteString ("p");
    SWholeIO.WriteCard (i, 1);
    STextIO.WriteString (": COROUTINES.COROUTINE;");
    STextIO.WriteLn;
  END;
  STextIO.WriteLn;
  STextIO.WriteString ("PROCEDURE TrBack;");
  STextIO.WriteLn;
  STextIO.WriteString ("VAR");
  STextIO.WriteLn;
  STextIO.WriteString ("self: COROUTINES.COROUTINE;");
  STextIO.WriteLn;
  STextIO.WriteString ("BEGIN");
  STextIO.WriteLn;
  STextIO.WriteString ("i := i+1;");
  STextIO.WriteLn;
  STextIO.WriteString ("COROUTINES.TRANSFER (self, main);");
  STextIO.WriteLn;
  STextIO.WriteString ("END TrBack;");
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  STextIO.WriteLn;
  STextIO.WriteLn;
  STextIO.WriteString ("BEGIN");
  STextIO.WriteLn;
  STextIO.WriteString ("i := 0;");
  STextIO.WriteLn;
  FOR i := 1 TO minLimit DO
    STextIO.WriteString ("COROUTINES.NEWCOROUTINE (TrBack,");
    STextIO.WriteString ("SYSTEM.ADR (wsp");
    SWholeIO.WriteCard (i, 1);
    STextIO.WriteString ("), wspsize, p");
    SWholeIO.WriteCard (i, 1);
    STextIO.WriteString (");");
    STextIO.WriteLn;
  END;
  FOR i := 1 TO minLimit DO
    STextIO.WriteString ("COROUTINES.TRANSFER (main, p");
    SWholeIO.WriteCard (i, 1);
    STextIO.WriteString (");");
    STextIO.WriteLn;
  END;
  STextIO.WriteString ("END LimitCoroutines.");
  STextIO.WriteLn;
END Coroutines.
```

A.1.12 Number of Parameters of a Procedure

The following program generates a program to specify the *number of parameters of a procedure* attribute and that the minimum limit for the attribute be 16.

```
MODULE Parameters;
IMPORT SWholeIO;
IMPORT STextIO;

CONST
  minLimit = 16;
VAR
  i: CARDINAL;

BEGIN
  STextIO.WriteString ("MODULE LimitParameters;");
  STextIO.WriteLn;
  STextIO.WriteString ("PROCEDURE P (");
  FOR i := 1 TO minLimit-1 DO
    STextIO.WriteString ("p");
    SWholeIO.WriteCard (i, 1);
    STextIO.WriteString (",");
        STextIO.WriteLn;
  END;
  STextIO.WriteString ("p");
  SWholeIO.WriteCard (minLimit, 1);
  STextIO.WriteString (": CARDINAL);");
  STextIO.WriteLn;
  STextIO.WriteString ("BEGIN");
  STextIO.WriteLn;
  STextIO.WriteString ("END P;");
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  STextIO.WriteLn;
  STextIO.WriteLn;
  STextIO.WriteString ("BEGIN");
  STextIO.WriteLn;
  STextIO.WriteString ("P (");
  FOR i := 1 TO minLimit-1 DO
    SWholeIO.WriteCard (i, 1);
    STextIO.WriteString (", ");
        STextIO.WriteLn;
  END;
  SWholeIO.WriteCard (minLimit, 1);
  STextIO.WriteString (");");
  STextIO.WriteLn;
  STextIO.WriteString ("END LimitParameters.");
  STextIO.WriteLn;
END Parameters.
```

A.1.13 Depth of Nested IF-THEN-ELSEs in THEN-branch

The following program generates a program to specify the *depth of nested IF-THEN-ELSEs in THEN-branch* attribute and that the minimum limit for the attribute be 16.

```
MODULE NestedIfThen;
IMPORT SWholeIO;
IMPORT STextIO;

CONST
  minLimit = 16;
VAR
  m: CARDINAL;

PROCEDURE GenStat;
BEGIN
  STextIO.WriteString ("i1 := 3;");
  STextIO.WriteLn;
END GenStat;

PROCEDURE GenIf (n: CARDINAL);
BEGIN
  IF n = 0 THEN
    GenStat;
  ELSE
    STextIO.WriteString ("IF b");
    SWholeIO.WriteCard (n, 1);
    STextIO.WriteString  (" THEN");
    STextIO.WriteLn;
    GenIf (n-1);
    STextIO.WriteString ("END;");
    STextIO.WriteLn;
  END;
END GenIf;

BEGIN
  STextIO.WriteString ("MODULE LimitNestedIfThen;");
  STextIO.WriteLn;
  STextIO.WriteString ("VAR");
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  STextIO.WriteLn;
  STextIO.WriteString ("i1: INTEGER;");
  STextIO.WriteLn;
  FOR m := 1 TO minLimit DO
    STextIO.WriteString ("b");
    SWholeIO.WriteCard (m, 1);
    STextIO.WriteString (": BOOLEAN;");
    STextIO.WriteLn;
  END;
  STextIO.WriteString ("BEGIN");
  STextIO.WriteLn;
  FOR m := 1 TO minLimit DO
    STextIO.WriteString ("b");
    SWholeIO.WriteCard (m, 1);
    STextIO.WriteString (" := TRUE;");
    STextIO.WriteLn;
  END;
  GenIf (minLimit);
  STextIO.WriteString ("END LimitNestedIfThen.");
  STextIO.WriteLn;
END NestedIfThen.
```

A.1.14 Depth of Nested IF-THEN-ELSEs in Both Branches

The following program generates a program to specify the *depth of nested IF-THEN-ELSEs in both branches* attribute and that the minimum limit for the attribute be 4.

```
MODULE NestedIfBoth;
IMPORT SWholeIO;
IMPORT STextIO;

CONST
  minLimit = 4;
VAR
  m: CARDINAL;

PROCEDURE GenStat;
BEGIN
  STextIO.WriteString ("i1 := 3;");
  STextIO.WriteLn;
END GenStat;

PROCEDURE GenIf (n: CARDINAL);
BEGIN
  IF n = 0 THEN
    GenStat;
  ELSE
    STextIO.WriteString ("IF b");
    SWholeIO.WriteCard (n, 1);
    STextIO.WriteString  (" THEN");
    STextIO.WriteLn;
    GenIf (n-1);
    STextIO.WriteString ("ELSE");
    STextIO.WriteLn;
    GenIf  (n-1);
    STextIO.WriteString ("END;");
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
    STextIO.WriteLn;
  END;
END GenIf;

BEGIN
  STextIO.WriteString ("MODULE LimitNestedIfBoth;");
  STextIO.WriteLn;
  STextIO.WriteString ("VAR");
  STextIO.WriteLn;
  STextIO.WriteString ("i1: INTEGER;");
  STextIO.WriteLn;
  FOR m := 1 TO minLimit DO
    STextIO.WriteString ("b");
    SWholeIO.WriteCard (m, 1);
    STextIO.WriteString (": BOOLEAN;");
    STextIO.WriteLn;
  END;
  STextIO.WriteString ("BEGIN");
  STextIO.WriteLn;
  FOR m := 1 TO minLimit DO
    STextIO.WriteString ("b");
    SWholeIO.WriteCard (m, 1);
    STextIO.WriteString (" := TRUE;");
    STextIO.WriteLn;
  END;
  GenIf (minLimit);
  STextIO.WriteString ("END LimitNestedIfBoth.");
  STextIO.WriteLn;
END NestedIfBoth.
```

A.1.15 Depth of Nested LOOP Statements

The following program generates a program to specify the *depth of nested LOOP statements* attribute and that the minimum limit for the attribute be 16.

```
MODULE NestedLoops;
IMPORT SWholeIO;
IMPORT STextIO;

CONST
  minLimit = 16;
VAR
  m: CARDINAL;

PROCEDURE GenLoop (n: CARDINAL);
BEGIN
  IF n <> 0 THEN
    STextIO.WriteString ("i");
    SWholeIO.WriteCard (n, 1);
    STextIO.WriteString (" := 1;");
    STextIO.WriteLn;
    STextIO.WriteString ("LOOP");
    STextIO.WriteLn;
    STextIO.WriteString ("INC (i");
    SWholeIO.WriteCard (n, 1);
    STextIO.WriteString (", 1);");
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
    STextIO.WriteLn;
    GenLoop (n-1);
    STextIO.WriteString ("IF i");
    SWholeIO.WriteCard (n, 1);
    STextIO.WriteString ("=2 THEN EXIT; END;");
    STextIO.WriteLn;
    STextIO.WriteString ("END;");
    STextIO.WriteLn;
  END;
END GenLoop;

BEGIN
  STextIO.WriteString ("MODULE LimitNestedLoops;");
  STextIO.WriteLn;
  STextIO.WriteString ("VAR");
  STextIO.WriteLn;
  FOR m := 1 TO  minLimit DO
    STextIO.WriteString ("i");
    SWholeIO.WriteCard (m, 1);
    STextIO.WriteString (": INTEGER;");
    STextIO.WriteLn;
  END;
  STextIO.WriteLn;
  STextIO.WriteString ("BEGIN");
  STextIO.WriteLn;
  GenLoop (minLimit);
  STextIO.WriteString ("END LimitNestedLoops.");
  STextIO.WriteLn;
END NestedLoops.
```

A.1.16 Depth of Nested WHILE Statements

The following program generates a program to specify the *depth of nested WHILE statements* attribute and that the minimum limit for the attribute be 16.

```
MODULE NestedWhiles;
IMPORT STextIO;

CONST
  minLimit = 16;

PROCEDURE GenWhile (n: CARDINAL);
BEGIN
  IF n > 0 THEN
    STextIO.WriteString ("WHILE b>0 DO");
    STextIO.WriteLn;
    GenWhile (n-1);
    STextIO.WriteString ("END;");
    STextIO.WriteLn;
  ELSE
    STextIO.WriteString ("b := -1;");
  END;
END GenWhile;

BEGIN
  STextIO.WriteString ("MODULE LimitNestedWhiles;");
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  STextIO.WriteLn;
  STextIO.WriteString ("VAR");
  STextIO.WriteLn;
  STextIO.WriteString ("b:INTEGER;");
  STextIO.WriteLn;
  STextIO.WriteString ("BEGIN");
  STextIO.WriteLn;
  STextIO.WriteString ("b := 1;");
  STextIO.WriteLn;
  GenWhile (minLimit);
  STextIO.WriteString ("END LimitNestedWhiles.");
  STextIO.WriteLn;
END NestedWhiles.
```

A.1.17 Depth of Nested REPEAT Statements

The following program generates a program to specify the *depth of nested REPEAT statements* attribute and that the minimum limit for the attribute be 16.

```
MODULE NestedRepeats;
IMPORT STextIO;

CONST
  minLimit = 16;

PROCEDURE GenRep (n: CARDINAL);
BEGIN
  IF n > 0 THEN
    STextIO.WriteString ("REPEAT");
    STextIO.WriteLn;
    GenRep( n-1);
    STextIO.WriteString ("UNTIL b>0;");
    STextIO.WriteLn;
  ELSE
    STextIO.WriteString ("b := 1;");
    STextIO.WriteLn;
  END;
END GenRep;

BEGIN
  STextIO.WriteString ("MODULE LimitNestedRepeats;");
  STextIO.WriteLn;
  STextIO.WriteString ("VAR");
  STextIO.WriteLn;
  STextIO.WriteString ("b: INTEGER;");
  STextIO.WriteLn;
  STextIO.WriteString ("BEGIN");
  STextIO.WriteLn;
  STextIO.WriteString ("b := -1;");
  STextIO.WriteLn;
  GenRep(minLimit);
  STextIO.WriteString ("END LimitNestedRepeats.");
  STextIO.WriteLn;
END NestedRepeats.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A.1.18 Depth of Nested FOR Statements

The following program generates a program to specify the *depth of nested FOR statements* attribute and that the minimum limit for the attribute be 16.

```
MODULE NestedFors;
IMPORT SWholeIO;
IMPORT STextIO;

CONST
  minLimit = 16;
VAR
  n: CARDINAL;

PROCEDURE GenDecl (n: CARDINAL);
BEGIN
  IF n>0 THEN
    STextIO.WriteString ("i");
    SWholeIO.WriteCard (n, 1);
    IF n > 1 THEN
      STextIO.WriteString (",");
    END;
    STextIO.WriteLn;
    GenDecl (n-1);
  END;
END GenDecl;

PROCEDURE GenFor (n: CARDINAL);
BEGIN
  IF n > 0 THEN
    STextIO.WriteString ("FOR i");
    SWholeIO.WriteCard (n, 1);
    STextIO.WriteString (" :=1 TO 2 DO");
    STextIO.WriteLn;
    GenFor (n-1);
    STextIO.WriteString ("END;");
    STextIO.WriteLn;
  END;
END GenFor;

BEGIN
  STextIO.WriteString ("MODULE LimitNestedFors;");
  STextIO.WriteLn;
  STextIO.WriteString ("VAR ");
  STextIO.WriteLn;
  GenDecl (minLimit);
  STextIO.WriteString (": CARDINAL;");
  STextIO.WriteLn;
  STextIO.WriteString ("BEGIN");
  STextIO.WriteLn;
  GenFor (minLimit);
  STextIO.WriteString ("END LimitNestedFors.");
  STextIO.WriteLn;
END NestedFors.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

A.1.19 Depth of Nested WITH Statements

The following program generates a program to specify the *depth of nested WITH statements* attribute and that the minimum limit for the attribute be 16.

```
MODULE NestedWiths;
IMPORT SWholeIO;
IMPORT STextIO;

CONST
  minLimit = 16;

PROCEDURE GenB (n: CARDINAL);
BEGIN
  IF n>0 THEN
    STextIO.WriteString ("b");
    SWholeIO.WriteCard (n, 1);
    IF n>1 THEN
      STextIO.WriteString (",");
    END;
    STextIO.WriteLn;
    GenB (n-1);
  END;
END GenB;

PROCEDURE GenWith (n: CARDINAL);
BEGIN
  IF n > 0 THEN
    STextIO.WriteString ("WITH b");
    SWholeIO.WriteCard (n, 1);
    STextIO.WriteString (" DO");
    STextIO.WriteLn;
    GenWith (n-1);
    STextIO.WriteString ("END;");
    STextIO.WriteLn;
  END;
END GenWith;

BEGIN
  STextIO.WriteString ("MODULE LimitNestedWiths;");
  STextIO.WriteLn;
  STextIO.WriteString ("TYPE");
  STextIO.WriteLn;
  STextIO.WriteString ("S = RECORD");
  STextIO.WriteLn;
  STextIO.WriteString ("s: INTEGER;");
  STextIO.WriteLn;
  STextIO.WriteString ("END;");
  STextIO.WriteLn;
  STextIO.WriteString ("VAR ");
  STextIO.WriteLn;
  GenB (minLimit);
  STextIO.WriteString (": S;");
  STextIO.WriteLn;
  STextIO.WriteString ("BEGIN");
  STextIO.WriteLn;
  GenWith(minLimit);
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  STextIO.WriteString ("END LimitNestedWiths.");
  STextIO.WriteLn;
END NestedWiths.
```

A.1.20 Number of CASE Alternatives with ELSE part

The following program generates a program to specify the *number of CASE alternatives* attribute and that the minimum limit for the attribute be 255.

```
MODULE CaseAlternatives;
IMPORT SWholeIO;
IMPORT STextIO;

CONST
  minLimit = 255;
VAR
  j: CARDINAL;

BEGIN
  STextIO.WriteString ("MODULE LimitCaseAlternatives;");
  STextIO.WriteLn;
  STextIO.WriteString ("VAR");
  STextIO.WriteLn;
  STextIO.WriteString ("k, n: CARDINAL;");
  STextIO.WriteLn;
  STextIO.WriteString ("BEGIN");
  STextIO.WriteLn;
  STextIO.WriteString ("k := 1;");
  STextIO.WriteLn;
  STextIO.WriteString ("CASE k OF");
  STextIO.WriteLn;
  FOR j := 1 TO minLimit-1 DO
    SWholeIO.WriteCard (j, 1);
    STextIO.WriteString (":");
    STextIO.WriteLn;
    STextIO.WriteString ("n := k;");
    STextIO.WriteLn;
    STextIO.WriteString ("|");
    STextIO.WriteLn;
  END;
  SWholeIO.WriteCard (minLimit, 1);
  STextIO.WriteString (":");
  STextIO.WriteLn;
  STextIO.WriteString ("n := k;");
  STextIO.WriteLn;
  STextIO.WriteString ("ELSE");
  STextIO.WriteLn;
  STextIO.WriteString ("n := 0;");
  STextIO.WriteLn;
  STextIO.WriteString ("END;");
  STextIO.WriteLn;
  STextIO.WriteString ("END LimitCaseAlternatives.");
  STextIO.WriteLn;
END CaseAlternatives.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

A.1.21 Depth of Nested Parentheses in Expressions

The following program generates a program to specify the *depth of nested parentheses in expressions* attribute and that the minimum limit for the attribute be 16.

```
MODULE NestedParentheses;
IMPORT SWholeIO;
IMPORT STextIO;

CONST
  minLimit = 16;

PROCEDURE GenTerm (m: CARDINAL);
BEGIN
  IF m = 0 THEN
    STextIO.WriteString ("t");
  ELSE
    SWholeIO.WriteCard (m, 1);
    IF m MOD 2 = 0 THEN
      STextIO.WriteString ("*(");
    ELSE
      STextIO.WriteString ("/(");
    END;
    GenTerm (m-1);
    IF m MOD 2 = 0 THEN
      STextIO.WriteString ("-");
    ELSE
      STextIO.WriteString ("+");
    END;
    SWholeIO.WriteCard (m, 1);
    STextIO.WriteString (")");
    STextIO.WriteLn;
  END;
END GenTerm;

BEGIN
  STextIO.WriteString ("MODULE LimitNestedParentheses;");
  STextIO.WriteLn;
  STextIO.WriteString ("VAR");
  STextIO.WriteLn;
  STextIO.WriteString ("t: INTEGER;");
  STextIO.WriteLn;
  STextIO.WriteString ("BEGIN");
  STextIO.WriteLn;
  STextIO.WriteString ("t := 1;");
  STextIO.WriteLn;
  STextIO.WriteString ("t := ");
  STextIO.WriteLn;
  GenTerm (minLimit);
  STextIO.WriteString (";");
  STextIO.WriteLn;
  STextIO.WriteString ("END LimitNestedParentheses.");
  STextIO.WriteLn;
END NestedParentheses.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

A.1.22 Depth of Recursion

The following program generates a program to specify the *depth of recursion* attribute and that the minimum limit for the attribute be 256.

```
MODULE RecursionDepth;
IMPORT SWholeIO;
IMPORT STextIO;

CONST
  minLimit = 256;

BEGIN
  STextIO.WriteString ("MODULE LimitRecursionDepth;");
  STextIO.WriteLn;
  STextIO.WriteString ("PROCEDURE P (n: CARDINAL);");
  STextIO.WriteLn;
  STextIO.WriteString ("BEGIN");
  STextIO.WriteLn;
  STextIO.WriteString ("IF n=0 THEN RETURN END;");
  STextIO.WriteLn;
  STextIO.WriteString ("P (n-1);");
  STextIO.WriteLn;
  STextIO.WriteString ("END P;");
  STextIO.WriteLn;
  STextIO.WriteString ("BEGIN");
  STextIO.WriteLn;
  STextIO.WriteString ("P (");
  SWholeIO.WriteCard (minLimit, 1);
  STextIO.WriteString (");");
  STextIO.WriteLn;
  STextIO.WriteString ("END LimitRecursionDepth.");
  STextIO.WriteLn;
END RecursionDepth.
```

A.1.23 Number of Imported Modules

The following program generates a program to specify the *number of imported modules* attribute and that the minimum limit for the attribute be 16.

The generated compilation modules may be separated if an implementation only accepts input in such a manner.

```
MODULE Imports;
IMPORT SWholeIO;
IMPORT STextIO;

CONST
  minLimit = 16;
VAR
  i: CARDINAL;
  b: ARRAY [1..3] OF CHAR;

BEGIN
  FOR i := 1 TO minLimit DO
    b[1] := 'Q';
    b[2] := CHR (i DIV 10 + ORD ('0'));
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
    b[3] := CHR (i MOD 10 + ORD ('0'));
    STextIO.WriteString ("DEFINITION MODULE ");
    STextIO.WriteString (b);
    STextIO.WriteString (";");
    STextIO.WriteLn;
    STextIO.WriteString ("VAR");
    STextIO.WriteLn;
    STextIO.WriteString ("r");
    SWholeIO.WriteCard (i, 1);
    STextIO.WriteString (": INTEGER;");
    STextIO.WriteLn;
    STextIO.WriteString ("END ");
    STextIO.WriteString (b);
    STextIO.WriteString (".");
    STextIO.WriteLn;
    STextIO.WriteLn;
    STextIO.WriteString ("IMPLEMENTATION MODULE ");
    STextIO.WriteString (b);
    STextIO.WriteString (";");
    STextIO.WriteLn;
    STextIO.WriteString ("BEGIN");
    STextIO.WriteLn;
    STextIO.WriteString ("END ");
    STextIO.WriteString (b);
    STextIO.WriteString (".");
    STextIO.WriteLn;
    STextIO.WriteLn;
  END;
  STextIO.WriteString ("MODULE LimitImports;");
  STextIO.WriteLn;
  FOR i := 1 TO minLimit DO
    STextIO.WriteString ("FROM ");
    b[1] := 'Q';
    b[2] := CHR (i DIV 10 + ORD ('0'));
    b[3] := CHR (i MOD 10 + ORD ('0'));
    STextIO.WriteString (b);
    STextIO.WriteString (" IMPORT r");
    SWholeIO.WriteCard (i, 1);
    STextIO.WriteString (";");
    STextIO.WriteLn;
  END;
  STextIO.WriteString ("BEGIN");
  STextIO.WriteLn;
  STextIO.WriteString ("END LimitImports.");
  STextIO.WriteLn;
END Imports.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Annex B
(normative)
The Specification of Library Modules

This annex gives the rules of interpretation that are to be applied to the specifications of standard library modules as given in section 8 and section 9. The static semantics for use of a standard library module are given by the corresponding definition module. The dynamic semantics are given by an abstract implementation module formed from the remaining components of the specification.

The realization of the specification of a standard library module need not be in the form of a sourced implementation module provided that the realization conforms to the specification.

B.1 Components of a Library Module Specification

The formal specification of a standard library module shall consist of the components listed below.

— The source text of the definition module.

— Value definitions and type definitions to support any constants and types associated with the formal definition of the module.

— Optionally, a state together with its initialization and invariant; the name of the state is < *LibraryName* > which is the name of the library module. The names of the components of the state are prefixed with < *libraryname* > which is the library name spelt in lower case. If the library has access to external objects, the state should contain a component where a copy of a model of the external object can be stored.

— Optionally, a definition of a composite type of a value that models external objects that the library has access to, and may change; together with an operation to install the values of the components of this type in the state and an operation to recover the components of a value of this type from the state of the library. The name of this type is *external-* < *LibraryName* > *-state*. The name of the operation to install a value of the type in the state is *read-external-* < *libraryname* > *-state* and the name of the operation to recover a value of the type from the state is *write-external-* < *libraryname* > *-state*.

— Optionally, a set of operations — one per function procedure or proper procedure defined in the definition module — named after the procedure identifier.

— Optionally, an initialization operation whose name is < *libraryname* > *-initialization*. This operation defines the actions that are taken during module initialization and corresponds to the statement sequence that is executed during the initialization of a sourced module.

— Optionally, a termination operation whose name is < *libraryname* > *-termination*. This operation defines the actions that are taken during module termination and corresponds to the finalization body of a module that is executed during the termination of a sourced module.

B.2 Interpretation of a Library Module Specification

The components defined in the state for the library, if present, shall be added to *State*, the state for the specification of a Modula-2 program — see 6.12.1.

The invariant component of the library state shall be conjoined to the invariant of *State*.

The initialization component of the library state shall be conjoined to the initialization of *State*.

An object of type *Abstract-implementation-module* shall be constructed in the following way:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— Let *name* : *Identifier* be the abstract representation of the library module name.

— Let *init* denote the operation < *libraryname* > *-initialization*, if it is present; otherwise let *init* denote nil .

— Construct a 1–1 mapping *procs* : *Identifier* $\xrightarrow{m}$ *Procedure-id* from the names of the procedures defined in the definition module and from elements of type *Procedure-id* that are not used elsewhere.

— Construct a mapping *dens* : *Procedure-id* $\xrightarrow{m}$ *Program-cont* from the elements of type *Procedure-id* to the definitions of the library function procedures and proper procedures so that the combination of this mapping and the one described above will allow the corresponding procedure denotation to be recovered from the identifier that denotes that procedure.

— Let *final* denote the operation < *libraryname* > *-termination*, if it is present; otherwise let *final* denote nil .

— Construct an object of type Abstract-implementation-module:

mk-Abstract-implementation-module (*name*, *imports*, *init*, *proc*, *dens*, *final*)

and add it to the set of implementation modules that form the program.

The components defined by the *external-* < *libraryname* > *-state* for the library, if present, shall be added to *External-program-state*, for the specification of a Modula-2 program — see 6.12.2.

The invariant component of the *external-* < *libraryname* > *-state* shall be conjoined to the invariant of *External-program-state*.

As part of the call of *read-external-program-state*, the changes to the library state as defined by the operation *read-external-* < *libraryname* > *-state* should be carried out.

As part of the call of *write-external-program-state*, the extraction of changes to the library state as defined by the operation *write-external-* < *libraryname* > *-state* should be carried out.

The definition of a program text together with its libraries is thus given by the following function from 6.1.1:

m-program : *Program* → *External-program-states* → *External-program-states*

m-program (*mk-Program* (*pmod*, *dmods*, *imods*))(*ext-state*) $\triangleq$

...

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Annex C
(informative)
Collected Modula-2 Concrete Syntax

C.1 Programs, Program Modules, and Separate Modules

C.1.1 Programs and Compilation Modules

compilation module = program module | definition module | implementation module ;

C.1.2 Program Modules

program module =
    "MODULE", module identifier, [interrupt protection], semicolon,
    import lists,
    module block, module identifier, period ;

module identifier = identifier ;

C.1.3 Definition Modules

definition module =
    "DEFINITION", "MODULE", module identifier, semicolon,
    import lists, definitions,
    "END", module identifier, period ;

C.1.4 Implementation Modules

C.1.4.1 Sourced Implementation Modules

implementation module =
    "IMPLEMENTATION", "MODULE", module identifier, [interrupt protection], semicolon,
    import lists,
    module block, module identifier, period ;

C.1.5 Import Lists

import lists = { import list } ;

C.1.5.1 Import List

import list = simple import | unqualified import ;

C.1.5.2 Simple Imports

simple import = "IMPORT", identifier list, semicolon ;

C.1.5.3 Unqualified Import

unqualified import = "FROM", module identifier, "IMPORT", identifier list, semicolon ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

C.1.6 Export Lists

export list = unqualified export | qualified export ;

C.1.6.1 Unqualified Exports

unqualified export = "EXPORT", identifier list, semicolon ;

C.1.6.2 Qualified Exports

qualified export = "EXPORT", "QUALIFIED", identifier list, semicolon ;

C.1.7 Protected Modules

interrupt protection = left bracket, protection expression, right bracket ;

protection expression = constant expression ;

C.2 Definitions and Declarations

C.2.1 Qualified Identifiers

qualified identifier = { qualifying identifier, period }, identifier ;

qualifying identifier = module identifier ;

C.2.2 Definitions

definitions = { definition } ;

definition =
    "CONST", { constant declaration, semicolon } |
    "TYPE", { type definition, semicolon } |
    "VAR", { variable declaration, semicolon } |
    procedure heading, semicolon ;

C.2.3 Type Definitions

type definition = type declaration | opaque type definition ;

opaque type definition = identifier ;

C.2.4 Procedure Headings

procedure heading = proper procedure heading | function procedure heading ;

C.2.4.1 Proper Procedure Headings

proper procedure heading = "PROCEDURE", procedure identifier, [formal parameters] ;

formal parameters = left parenthesis, [formal parameter list], right parenthesis ;

formal parameter list = formal parameter, { semicolon, formal parameter } ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

C.2.4.2 Function Procedure Headings

function procedure heading =
"PROCEDURE", procedure identifier, formal parameters,
colon, function result type ;

function result type = type identifier ;

C.2.4.3 Formal Parameters

formal parameter = value parameter specification | variable parameter specification ;

C.2.4.3.1 Value Parameters

value parameter specification = identifier list, colon, formal type ;

C.2.4.3.2 Variable Parameters

variable parameter specification = "VAR", identifier list, colon, formal type ;

C.2.5 Declarations

declarations = { declaration } ;

declaration =
"CONST", { constant declaration, semicolon } |
"TYPE", { type declaration, semicolon } |
"VAR", { variable declaration, semicolon } |
procedure declaration, semicolon |
local module declaration, semicolon ;

C.2.6 Constant Declarations

constant declaration = identifier, equals, constant expression ;

C.2.7 Type Declarations

type declaration = identifier, equals, type denoter ;

C.2.8 Variable Declarations

variable declaration = variable identifier list, colon, type denoter ;

variable identifier list = identifier, [machine address], { comma, identifier, [machine address] } ;

machine address = left bracket, value of address type, right bracket ;

value of address type = constant expression ;

C.2.9 Procedure Declarations

procedure declaration = proper procedure declaration | function procedure declaration ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

C.2.9.1 Proper Procedure Declarations

proper procedure declaration =
    proper procedure heading, semicolon,
    (proper procedure block, procedure identifier | "FORWARD") ;

procedure identifier = identifier ;

C.2.9.2 Function Procedure Declarations

function procedure declaration =
    function procedure heading, semicolon,
    (function procedure block, procedure identifier | "FORWARD") ;

C.2.10 Local Module Declarations

local module declaration =
    "MODULE", module identifier, [interrupt protection], semicolon,
    import lists, [export list],
    module block, module identifier ;

C.3 Data Types

C.3.1 Type Denoters and Ordinal Type Denoters

type denoter = type identifier | new type ;

ordinal type denoter = ordinal type identifier | new ordinal type ;

C.3.2 Type Identifiers

type identifier = qualified identifier ;

ordinal type identifier = type identifier ;

C.3.3 New Types

new type =
    new ordinal type | set type | packedset type | pointer type |
    procedure type | array type | record type ;

new ordinal type = enumeration type | subrange type ;

C.3.4 Enumeration Types

enumeration type = left parenthesis, identifier list, right parenthesis ;

identifier list = identifier, { comma, identifier } ;

C.3.5 Subrange Types

subrange type = [range type], left bracket, constant expression, ellipsis, constant expression, right bracket ;

range type = ordinal type identifier ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

C.3.6 Set Types

set type = "SET", "OF", base type ;

base type = ordinal type denoter ;

C.3.7 Packedset Types

packedset type = "PACKEDSET", "OF", base type ;

C.3.8 Pointer Types

pointer type = "POINTER", "TO", bound type ;

bound type = type denoter ;

C.3.9 Procedure Types

procedure type = proper procedure type | function procedure type ;

C.3.9.1 Proper Procedure Types

proper procedure type = "PROCEDURE", [left parenthesis, [formal parameter type list], right parenthesis] ;

C.3.9.2 Function Procedure Types

function procedure type =
"PROCEDURE", left parenthesis, [formal parameter type list], right parenthesis,
colon, function result type ;

C.3.9.3 Formal Parameter Type Lists

formal parameter type list = formal parameter type, { comma, formal parameter type } ;

formal parameter type = variable formal type | value formal type ;

C.3.9.4 Variable Formal Types

variable formal type = "VAR", formal type ;

C.3.9.5 Value Formal Types

value formal type = formal type ;

C.3.10 Formal Types

formal type = type identifier | open array formal type ;

open array formal type = "ARRAY", "OF", open array component type ;

open array component type = formal type ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

C.3.11 Array Types

array type = "ARRAY", index type, { comma, index type }, "OF", component type ;

index type = ordinal type denoter ;

component type = type denoter ;

C.3.12 Record Types

record type = "RECORD", field list, "END" ;

field list = fields, { semicolon, fields } ;

fields = [fixed fields | variant fields] ;

C.3.12.1 Fixed Fields

fixed fields = identifier list, colon, field type ;

field type = type denoter ;

C.3.12.2 Variant Fields

variant fields = "CASE", tag field, "OF", variant list, "END" ;

tag field = [tag identifier], colon, tag type ;

tag identifier = identifier ;

tag type = ordinal type identifier ;

variant list = variant, { case separator, variant }, [variant else part] ;

variant else part = "ELSE", field list ;

variant = [variant label list, colon, field list] ;

variant label list = variant label, { comma, variant label } ;

variant label = constant expression, [ellipsis, constant expression] ;

C.4 Blocks

C.4.1 Proper Procedure Blocks

proper procedure block = declarations, [procedure body], "END" ;

procedure body = "BEGIN", block body ;

C.4.2 Function Procedure Blocks

function procedure block = declarations, function body, "END" ;

function body = "BEGIN", block body ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

C.4.3 Module Blocks

module block = declarations, [module body], "END" ;

module body = initialization body, [finalization body] ;

initialization body = "BEGIN", block body ;

finalization body = "FINALLY", block body ;

C.4.4 Block Bodies and Exception Handling

block body = normal part, ["EXCEPT", exceptional part] ;

normal part = statement sequence ;

exceptional part = statement sequence ;

C.5 Statements

statement =
empty statement | assignment statement | procedure call | return statement |
retry statement | with statement | if statement | case statement |
while statement | repeat statement | loop statement | exit statement |
for statement ;

C.5.1 Statement Sequences

statement sequence = statement, { semicolon, statement } ;

C.5.2 Empty Statements

empty statement = ;

C.5.3 Assignment Statements

assignment statement = variable designator, assignment operator, expression ;

C.5.4 Procedure Calls

procedure call = procedure designator, [actual parameters] ;

procedure designator = value designator ;

C.5.5 Return Statements

return statement = simple return statement | function return statement ;

C.5.5.1 Simple Return Statements

simple return statement = "RETURN" ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

C.5.5.2 Function Return Statements

function return statement = "RETURN", expression ;

C.5.6 Retry Statements

retry statement = "RETRY" ;

C.5.7 With Statements

with statement = "WITH", record designator, "DO", statement sequence, "END" ;

record designator = variable designator | value designator ;

C.5.8 If Statements

if statement = guarded statements, [if else part], "END" ;

guarded statements =
"IF", boolean expression, "THEN", statement sequence,
{ "ELSIF", boolean expression, "THEN", statement sequence } ;

if else part = "ELSE", statement sequence ;

boolean expression = expression ;

C.5.9 Case Statements

case statement = "CASE", case selector, "OF", case list, "END" ;

case selector = ordinal expression ;

case list = case alternative, { case separator, case alternative }, [case else part] ;

case else part = "ELSE", statement sequence ;

C.5.9.1 Case Alternatives

case alternative = [case label list, colon, statement sequence] ;

case label list = case label, { comma, case label } ;

case label = constant expression, [ellipsis, constant expression] ;

C.5.10 While Statements

while statement = "WHILE", boolean expression, "DO", statement sequence, "END" ;

C.5.11 Repeat Statements

repeat statement = "REPEAT", statement sequence, "UNTIL", boolean expression ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

C.5.12 Loop Statements

loop statement = "LOOP", statement sequence, "END" ;

C.5.13 Exit Statements

exit statement = "EXIT" ;

C.5.14 For Statements

for statement =
"FOR", control variable identifier, assignment operator, initial value, "TO", final value, ["BY", step size], "DO", statement sequence, "END" ;

control variable identifier = identifier ;

initial value = ordinal expression ;

final value = ordinal expression ;

step size = constant expression ;

C.6 Variable Designators

variable designator = entire designator | indexed designator | selected designator | dereferenced designator ;

C.6.1 Entire Designators

entire designator = qualified identifier ;

C.6.2 Indexed Designators

indexed designator = array variable designator, left bracket, index expression, { comma, index expression }, right bracket ;

array variable designator = variable designator ;

index expression = ordinal expression ;

C.6.3 Selected Designators

selected designator = record variable designator, period, field identifier ;

record variable designator = variable designator ;

field identifier = identifier ;

C.6.4 Dereferenced Designators

dereferenced designator = pointer variable designator, dereferencing operator ;

pointer variable designator = variable designator ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

C.7 Expressions

expression = simple expression, [relational operator, simple expression] ;

simple expression = [sign], term, { term operator, term } ;

term = factor, { factor operator, factor } ;

factor =
left parenthesis, expression, right parenthesis | logical negation operator, factor |
value designator | function call |
value constructor | constant literal ;

C.7.1 Ordinal Expressions

ordinal expression = expression ;

C.7.2 Infix Expressions and Operations

relational operator =
equals operator | inequality operator | less than operator |
greater than operator | less than or equal operator | subset operator |
greater than or equal operator | superset operator | set membership operator ;

term operator =
plus operator | set union operator | minus operator |
set difference operator | logical disjunction operator | string catenate symbol ;

factor operator =
multiplication operator | set intersection operator | division operator |
symmetric set difference operator | rem operator | div operator |
mod operator | logical conjunction operator ;

C.7.3 Value Designators

value designator = entire value | indexed value | selected value | dereferenced value ;

C.7.3.1 Entire Values

entire value = qualified identifier ;

C.7.3.2 Indexed Values

indexed value = array value, left bracket, index expression, { comma, index expression }, right bracket ;

array value = value designator ;

C.7.3.3 Selected Values

selected value = record value, period, field identifier ;

record value = value designator ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

C.7.3.4 Dereferenced Values

dereferenced value = pointer value, dereferencing operator ;

pointer value = value designator ;

C.7.4 Function Calls

function call = function designator, actual parameters ;

function designator = value designator ;

C.7.5 Value Constructors

value constructor = array constructor | record constructor | set constructor ;

C.7.5.1 Array Constructors

array constructor = array type identifier, array constructed value ;

array type identifier = type identifier ;

array constructed value =
left brace, repeated structure component, { comma, repeated structure component }, right brace ;

repeated structure component = structure component, ["BY", repetition factor] ;

repetition factor = constant expression ;

structure component = expression | array constructed value | record constructed value | set constructed value ;

C.7.5.2 Record Constructors

record constructor = record type identifier, record constructed value ;

record type identifier = type identifier ;

record constructed value = left brace, [structure component, { comma, structure component }], right brace ;

C.7.5.3 Set Constructors

set constructor = set type identifier, set constructed value ;

set type identifier = type identifier ;

set constructed value = left brace, [member, { comma, member }], right brace ;

member = interval | singleton ;

interval = ordinal expression, ellipsis, ordinal expression ;

singleton = ordinal expression ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

C.7.6 Constant Literals

constant literal = whole number literal | real literal | string literal ;

C.7.7 Constant Expressions

constant expression = expression ;

C.8 Parameter Compatibility and Argument Binding

C.8.1 Actual Parameters

actual parameters = left parenthesis, [actual parameter list], right parenthesis ;

actual parameter list = actual parameter, { comma, actual parameter } ;

actual parameter = variable designator | expression | type parameter ;

C.8.1.1 Type Parameters

type parameter = type identifier ;

C.9 Lexical components

assignment operator = ? as defined in the Lexis, section 5 ? ;

case separator = ? as defined in the Lexis, section 5 ? ;

colon = ? as defined in the Lexis, section 5 ? ;

comma = ? as defined in the Lexis, section 5 ? ;

dereferencing operator = ? as defined in the Lexis, section 5 ? ;

div operator = ? as defined in the Lexis, section 5 ? ;

division operator = ? as defined in the Lexis, section 5 ? ;

ellipsis = ? as defined in the Lexis, section 5 ? ;

equals = ? as defined in the Lexis, section 5 ? ;

equals operator = ? as defined in the Lexis, section 5 ? ;

greater than operator = ? as defined in the Lexis, section 5 ? ;

greater than or equal operator = ? as defined in the Lexis, section 5 ? ;

identifier = ? as defined in the Lexis, section 5 ? ;

inequality operator = ? as defined in the Lexis, section 5 ? ;

left brace = ? as defined in the Lexis, section 5 ? ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

left bracket = ? as defined in the Lexis, section 5 ? ;

left parenthesis = ? as defined in the Lexis, section 5 ? ;

less than operator = ? as defined in the Lexis, section 5 ? ;

less than or equal operator = ? as defined in the Lexis, section 5 ? ;

logical conjunction operator = ? as defined in the Lexis, section 5 ? ;

logical disjunction operator = ? as defined in the Lexis, section 5 ? ;

logical negation operator = ? as defined in the Lexis, section 5 ? ;

minus operator = ? as defined in the Lexis, section 5 ? ;

mod operator = ? as defined in the Lexis, section 5 ? ;

multiplication operator = ? as defined in the Lexis, section 5 ? ;

period = ? as defined in the Lexis, section 5 ? ;

plus operator = ? as defined in the Lexis, section 5 ? ;

real literal = ? as defined in the Lexis, section 5 ? ;

rem operator = ? as defined in the Lexis, section 5 ? ;

right brace = ? as defined in the Lexis, section 5 ? ;

right bracket = ? as defined in the Lexis, section 5 ? ;

right parenthesis = ? as defined in the Lexis, section 5 ? ;

semicolon = ? as defined in the Lexis, section 5 ? ;

set difference operator = ? as defined in the Lexis, section 5 ? ;

set intersection operator = ? as defined in the Lexis, section 5 ? ;

set membership operator = ? as defined in the Lexis, section 5 ? ;

set union operator = ? as defined in the Lexis, section 5 ? ;

sign = ? as defined in the Lexis, section 5 ? ;

string catenate symbol = ? as defined in the Lexis, section 5 ? ;

string literal = ? as defined in the Lexis, section 5 ? ;

subset operator = ? as defined in the Lexis, section 5 ? ;

superset operator = ? as defined in the Lexis, section 5 ? ;

symmetric set difference operator = ? as defined in the Lexis, section 5 ? ;

whole number literal = ? as defined in the Lexis, section 5 ? ;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Annex D
(informative)
Collected Modula-2 Abstract Syntax

The collected abstract syntax is sorted alphabetically

types

Abstract-implementation-module :: *name* : *Identifier*
init : [*Intialization-actions*]
procs : *Procedures*
dens : *Procdens*
final : [*Termination-actions*]

inv *mk-Abstract-implementation-module* (-, -, *procs*, *dens*, -) $\triangleq$
rng *procs* $\subseteq$ dom *dens*;

Actual-parameter = *Typed-variable-designator* | *Typed-expression* | *Type-parameter*;

Actual-parameters = *Actual-parameter**;

Arithmetic-prefix-operation :: *op* : *Arithmetic-prefix-operator*
otype : *Number-type*;

Arithmetic-prefix-operator = PLUS | MINUS;

Array-constructed-value :: *vals* : *Repeated-elements*;

Array-constructor :: *qid* : *Qualident*
def : *Array-constructed-value*;

Array-formal-type :: *type* : *Open-component-type*;

Array-type :: *itype* : *Type*
ctype : *Type*;

Assignment-statement :: *desig* : *Variable-designator*
expr : *Expression*;

Block-body :: *normal* : *Statement-sequence*
except : [*Statement-sequence*];

Boolean-operation :: *op* : *Boolean-operator*;

Boolean-operator = AND | OR;

Boolean-prefix-operation :: *op* : *Boolean-prefix-operator*;

Boolean-prefix-operator = NOT;

COROUTINES-function-designator :: *ident* : *Identifier*

inv *ident* $\triangleq$
ident $\in$ {HANDLER, IsATTACHED, CURRENT, PROT};

COROUTINES-procedure-designator :: *ident* : *Identifier*

inv *ident* $\triangleq$
ident $\in$ {ATTACH, DETACH, IOTRANSFER, LISTEN, NEWCOROUTINE, TRANSFER};

Case :: *labels* : *Set-definition*
body : *Statement-sequence*;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case-statement :: *expr* : *Expression*
cases : *Case*-set
elsep : [*Statement-sequence*];

Character-number-literal :: *val* : N;

Character-string :: *cl* : *Char**;

Compilation-module = *Program-module*
| *Definition-module*
| *Implementation-module*;

Complex-comparison-operation :: *op* : *Complex-comparison-operator*;

Complex-comparison-operator = EQ | NE;

Complex-number-operation :: *op* : *Complex-number-operator*
otype : *Complex-number-type*;

Complex-number-operator = ADD | SUBTRACT | MULTIPLY | DIVIDE;

Constant-declaration :: *id* : *Identifier*
expr : *Expression*;

Constant-literal = *Whole-number-literal* | *Real-literal* | *String-literal*;

Declaration = *Constant-declaration* | *Type-declaration* | *Variable-declaration*
| *Procedure-declaration* | *Module-declaration*;

Declarations = *Declaration**;

Definition = *Constant-declaration* | *Type-definition* | *Variable-declaration* | *Procedure-heading*;

Definition-module :: *name* : *Identifier*
imports : *Import-lists*
defs : *Definitions*;

Definition-modules = *Definition-module*-set;

Definitions = *Definition**;

Dereferenced-designator :: *desig* : *Variable-designator*;

Dereferenced-value :: *type* : *Typed*
desig : *Value-designator*;

EXCEPTIONS-function-designator :: *ident* : *Identifier*
inv *ident* $\triangleq$
ident $\in$ {`CurrentNumber`, `IsCurrentSource`, `IsExceptionalExecution`};

EXCEPTIONS-procedure-designator :: *ident* : *Identifier*
inv *ident* $\triangleq$
ident $\in$ {`AllocateSource`, `RAISE`, `GetMessage`};

Element = *Expression* | *Array-constructed-value* | *Record-constructed-value* | *Set-constructed-value*;

Elements = *Element**;

Empty-statement :: ;

Entire-designator :: *qid* : *Qualident*;

Entire-value :: *qid* : *Qualident*;

Enumeration-type :: *values* : *Identifier*$^+$;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Exit-statement :: EXIT;

Export-list = *Unqualified-export* | *Qualified-export*;

Expression = *Infix-expression*
| *Prefix-expression*
| *Value-designator*
| *Function-call*
| *Value-constructor*
| *Constant-literal*;

Field-list = *Fields**;

Fields = *Fixed-fields* | *Variant-fields*;

Fixed-fields :: *ids* : *Identifier*$^+$
type : *Type*;

For-statement :: *id* : *Identifier*
initial : *Expression*
final : *Expression*
step : *Expression*
body : *Statement-sequence*;

Formal-parameter = *Value-parameter-specification* | *Variable-parameter-specification*;

Formal-parameter-list = *Formal-parameter**;

Formal-parameter-type = *Variable-formal-type* | *Value-formal-type*;

Formal-parameter-type-list = *Formal-parameter-type*$^+$;

Formal-type = *Parameter-formal-type* | *Array-formal-type*;

Function-call :: *desig* : *Function-designator*
aps : *Actual-parameters*;

Function-designator = *Value-designator*
| *Standard-function-designator*
| *System-function-designator*;

Function-procedure-block :: *decls* : *Declarations*
body : *Block-body*;

Function-procedure-declaration :: *head* : *Function-procedure-heading*
block : (*Function-procedure-block* | FORWARD);

Function-procedure-heading :: *name* : *Identifier*
parms : *Formal-parameter-list*
return : *Type-identifier*;

Function-procedure-type :: *parms* : [*Formal-parameter-type-list*]
result : *Type-identifier*;

Function-return-statement :: *type* : *Typed*
expr : *Expression*;

Guarded-statement :: *guard* : *Expression*
body : *Statement-sequence*;

Identifier = $token_{ident}$;

If-statement :: *thens* : *Guarded-statement**
elsep : [*Statement-sequence*];

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Implementation-module = *Sourced-implementation-module* | *Abstract-implementation-module*;

Implementation-modules = *Implementation-module*-set;

Import-list = *Simple-import* | *Unqualified-import*;

Import-lists = *Import-list**;

Indexed-designator :: *desig* : *Variable-designator*
expr : *Expression*;

Indexed-value :: *desig* : *Value-designator*
expr : *Expression*;

Infix-expression :: *left* : *Expression*
op : *Infix-operation*
right : *Expression*;

Infix-operation = *Complex-number-operation*
| *Real-number-operation*
| *Whole-number-operation*
| *Boolean-operation*
| *Set-operation*
| *String-literal-operation*
| *Relational-operation*;

Interrupt-protection :: *expr* : *Expression*;

Interval :: *min* : *Expression*
max : *Expression*;

Loop-statement :: *body* : *Statement-sequence*;

M2EXCEPTION-function-designator :: *ident* : *Identifier*
inv *ident* $\triangleq$
ident $\in$ {`M2Exception`, `IsM2Exception`};

Member = *Interval* | *Singleton*;

Module-block :: *decls* : *Declarations*
initial : [*Block-body*]
final : [*Block-body*];

Module-declaration :: *name* : *Identifier*
imports : *Import-lists*
export : [*Export-list*]
block : *Module-block*
protection : [*Interrupt-protection*];

New-type = *Enumeration-type*
| *Subrange-type*
| *Set-type*
| *Packed-set-type*
| *Pointer-type*
| *Procedure-type*
| *Array-type*
| *Record-type*;

Opaque-type-definition :: *id* : *Identifier*;

Open-component-type = *Type-identifier* | *Array-formal-type*;

Packed-set-type :: *btype* : *Type*;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Parameter-formal-type :: *type* : *Type-identifier*;

Pointer-relational-operation :: *op* : *Pointer-relational-operator*;

Pointer-relational-operator = EQ | NE;

Pointer-type :: *btype* : *Type*;

Prefix-expression :: *op* : *Prefix-operation*
expr : *Expression*;

Prefix-operation = *Arithmetic-prefix-operation* | *Boolean-prefix-operation*;

Procedure-call :: *desig* : *Procedure-designator*
aps : *Actual-parameters*;

Procedure-declaration = *Proper-procedure-declaration* | *Function-procedure-declaration*;

Procedure-designator = *Value-designator*
| *Standard-procedure-designator*
| *System-procedure-designator*;

Procedure-heading = *Proper-procedure-heading* | *Function-procedure-heading*;

Procedure-relational-operation :: *op* : *Procedure-relational-operator*;

Procedure-relational-operator = EQ | NE;

Procedure-type = *Proper-procedure-type* | *Function-procedure-type*;

Program :: *pmod* : *Program-module*
dmods : *Definition-modules*
imods : *Implementation-modules*;

Program-module :: *name* : *Identifier*
imports : *Import-lists*
block : *Module-block*
protection : [*Interrupt-protection*];

Proper-procedure-block :: *decls* : *Declarations*
body : [*Block-body*];

Proper-procedure-declaration :: *head* : *Proper-procedure-heading*
block : (*Proper-procedure-block* | FORWARD);

Proper-procedure-heading :: *name* : *Identifier*
parms : *Formal-parameter-list*;

Proper-procedure-type :: *parms* : [*Formal-parameter-type-list*];

Protection-relational-operation :: *op* : *Protection-relational-operator*;

Protection-relational-operator = LE | GE;

Qualident = *Identifier* | *Qualified*;

Qualified :: *id* : *Identifier*
qualificand : *Qualident*;

Qualified-export :: *ids* : *Identifier*-set;

Real-literal :: *value* : $\mathbb{R}$;

Real-number-operation :: *op* : *Real-number-operator*
otype : *Real-number-type*;

Real-number-operator = ADD | SUBTRACT | MULTIPLY | DIVIDE;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Record-constructed-value :: *vals* : *Elements*;

Record-constructor :: *qid* : *Qualident*
def : *Record-constructed-value*;

Record-designator = *Variable-designator* | *Value-designator*;

Record-type :: *fields* : *Field-list*;

Relational-operation = *Complex-comparison-operation*
| *Scalar-relational-operation*
| *Set-relational-operation*
| *Procedure-relational-operation*
| *Pointer-relational-operation*
| *Protection-relational-operation*
| *System-basic-type-relational-operator*;

Repeat-statement :: *body* : *Statement-sequence*
expr : *Expression*;

Repeated-element :: *val* : *Element*
by : *Expression*;

Repeated-elements = *Repeated-element**;

Retry-statement :: RETRY;

Return-statement = *Simple-return-statement* | *Function-return-statement*;

SYSTEM-function-designator :: *ident* : *Identifier*

inv *ident* $\triangleq$
ident $\in$ {ADDADR, MAKEADR, ADR, CAST, DIFADR, ROTATE, SHIFT, SUBADR, TSIZE};

Scalar-relational-operation :: *op* : *Scalar-relational-operator*;

Scalar-relational-operator = EQ | NE | LT | GT | LE | GE;

Selected-designator :: *desig* : *Variable-designator*
id : *Identifier*;

Selected-value :: *desig* : *Value-designator*
id : *Identifier*;

Set-constructed-value = *Member*-set;

Set-constructor :: *qid* : *Qualident*
def : *Set-constructed-value*;

Set-definition = *Member*-set;

Set-operation :: *op* : *Set-operator*
otype : *Typed*;

Set-operator = UNION | DIFFERENCE | INTERSECTION | SYMMETRIC-DIFFERENCE;

Set-relational-operation :: *op* : *Set-relational-operator*;

Set-relational-operator = EQ | NE | SUBSET | SUPERSET | IN;

Set-type :: *btype* : *Type*;

Simple-import :: *ids* : *Identifier**;

Simple-return-statement :: RETURN;

Singleton :: *expr* : *Expression*;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Sourced-implementation-module :: *name* : *Identifier*
imports : *Import-lists*
block : *Module-block*
protection : [*Interrupt-protection*];

Standard-function-designator :: *ident* : *Identifier*

inv *ident* $\triangleq$
ident $\in$ {ABS, CAP, CHR, CMPLX, FLOAT, HIGH, IM, INT, LENGTH, LFLOAT, MAX, MIN, ODD, ORD, RE, SIZE, TRUNC, VAL};

Standard-procedure-designator :: *ident* : *Identifier*

inv *ident* $\triangleq$
ident $\in$ {DEC, DISPOSE, EXCL, HALT, INC, INCL, NEW};

Statement = *Empty-statement*
| *Assignment-statement*
| *Procedure-call*
| *Return-statement*
| *Retry-statement*
| *With-statement*
| *If-statement*
| *Case-statement*
| *While-statement*
| *Repeat-statement*
| *Loop-statement*
| *Exit-statement*
| *For-statement*;

Statement-sequence :: *actions* : *Statement*$^+$;

String-literal = *Character-string* | *Character-number-literal*;

String-literal-operation :: *otype* : *String-type*;

Subrange-type :: *rtype* : *Qualident*
range : *Interval*;

System-basic-type-relational-operation :: *op* : *System-basic-type-relational-operator*;

System-basic-type-relational-operator = EQ | NE;

System-function-designator = *SYSTEM-function-designator*
| *COROUTINES-function-designator*
| *TERMINATION-function-designator*
| *EXCEPTIONS-function-designator*
| *M2EXCEPTION-function-designator*;

System-procedure-designator = *COROUTINES-procedure-designator*
| *EXCEPTIONS-procedure-designator*;

TERMINATION-function-designator :: *ident* : *Identifier*

inv *ident* $\triangleq$
ident $\in$ {IsTerminating, HasHalted};

Type = *Type-identifier* | *New-type*;

Type-declaration :: *id* : *Identifier*
type : *Type*;

Type-definition = *Type-declaration* | *Opaque-type-definition*;

Type-identifier :: *qid* : *Qualident*;

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Type-parameter :: *type* : *Qualident*;

Typed-expression :: *expr* : *Expression*
type : *Expression-typed*;

Typed-variable-designator :: *desig* : *Variable-designator*
type : *Typed*;

Unqualified-export :: *ids* : *Identifier*-set;

Unqualified-import :: *from* : *Identifier*
ids : *Identifier*-set;

Value-constructor = *Array-constructor* | *Record-constructor* | *Set-constructor*;

Value-designator = *Entire-value*
| *Indexed-value*
| *Selected-value*
| *Dereferenced-value*;

Value-formal-type :: *ftype* : *Formal-type*;

Value-parameter-specification :: *id* : *Identifier*
ftype : *Formal-type*;

Variable-declaration :: *ids* : *Variable-identifier*-set
type : *Type*;

Variable-designator = *Entire-designator*
| *Indexed-designator*
| *Selected-designator*
| *Dereferenced-designator*;

Variable-formal-type :: *ftype* : *Formal-type*;

Variable-identifier :: *id* : *Identifier*
addr : [*Expression*];

Variable-parameter-specification :: *id* : *Identifier*
ftype : *Formal-type*;

Variant :: *labels* : *Set-definition*
fields : *Field-list*;

Variant-fields :: *tag* : [*Identifier*]
tagt : *Qualident*
variants : *Variant**
elsep : [*Field-list*];

While-statement :: *expr* : *Expression*
body : *Statement-sequence*;

Whole-number-literal :: *value* : ℕ;

Whole-number-operation :: *op* : *Whole-number-operator*
otype : *Whole-number-type*;

Whole-number-operator = ADD | SUBTRACT | MULTIPLY | DIVIDE | REM | DIV | MOD;

With-statement :: *desig* : *Record-designator*
body : *Statement-sequence*

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex E
(informative)
Modula-2 Glossary

The following explanations of terms are given as an aid to the reading of the Modula-2 definition.

Some terms included here are given explicit definitions in normative statements of this part of ISO/IEC 10514 or have definitions that follow from their use in those normative statements; the explanations are given for informative purposes only and are not to be used in preference to those definitions.

Many terms are given special meaning in the context of this part of ISO/IEC 10514 and therefore have interpretations that are not as general as those that might be given in a dictionary, and that might not match the reader's own idea of the meaning of these terms. For example, the term *elementary type* has a precise meaning that can be determined from the Modula-2 definition but is one that is not widely used: a *procedure type* is an elementary type in this part of ISO/IEC 10514, but is described as such in very few text books.

Other terms included here are not actually used in the Modula-2 definition but have related meanings arising, for example, from their use in *Programming in Modula-2*.

Abstract implementation module — The specification of the semantics of a separate module using the formal notation VDM-SL as required by annex B. The term is needed to refer to the formal specification of the standard library modules given in sections 8 and 9. See also *Sourced implementation module.*

Abstract Syntax — An abstract version of the underlying structure of the language in terms of a hierarchy of VDM-SL types. The abstract syntax shows the essential structure of Modula-2; it omits lexical details, such as keywords, which are only needed for parsing source code; it includes some structural information that is not treated in the concrete syntax but which is needed for the formal model, such as the distinction between infix and prefix expressions.

Activation — An instance of a procedure in execution during a procedure call. There may be several such instances, possibly active together in the case of recursive or re-entrant calls. Each activation involves the binding of formal parameters to the particular arguments for the call, and the allocation of storage for local variables. During each activation, the identifiers of local variables and value parameters refer to storage that is distinct from the storage for any other variables.

Anonymous type — A type that, unlike an identified type, has no name. No identifier exists for such a type, and none can be introduced in a type declaration. Of the predefined types, the numeric and string literal types and the nil type are anonymous types; no variables or parameters may be declared to be of these types, and no identified type may be declared as identical to these types. New types may be introduced as anonymous types if they are used in variable declarations or as part of a new type that is being declared in a type declaration. In the following there are three anonymous types: `VAR x,y: ARRAY [1..10] OF (no, yes);` The variables `x` and `y` are of the identical anonymous array type of which the index type is anonymous and the component type is also anonymous.

Any order of evaluation — See *Order of evaluation.*

Assignment-compatible — A type T_v is said to be assignment-compatible with the type of an expression T_e if values of the type T_e may be assigned as values of variables of the type T_v. Identical types are always assignment-compatible, but there are special rules for the assignment of values of the predefined anonymous types (the numeric and string literal types and the nil type), of procedure values, for types which have values in common (the signed and unsigned types, subrange types), and for the address type when assigned to or from any other pointer type. In cases where the types T_v and T_e might be different types, a *range check* might be needed when the assignment takes place. See 6.4.2.

Base type — The base type of a set type is the ordinal type whose values are the values that a value of that set type may contain.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Basic type — A predefined type that is not a constructed type; i.e. the types identified by the pervasive identifiers CHAR, BOOLEAN, the whole number types (INTEGER and CARDINAL), the real number types (REAL and LONGREAL), the complex number types (COMPLEX and LONGCOMPLEX), the anonymous basic types $\mathbb{Z}$, $\mathbb{R}$, $\mathbb{C}$, the nil type, and the string literal type. There are also the system basic types which are only available by import from system modules; these include the location type and the coroutine type.

Block — A sequence of declarations followed by a block body; a component of program modules, implementation modules, local modules, and full procedure declarations

Block body — The statements that are to be executed when a module is initialized or finalized, or when a procedure is activated. Block bodies have a normal part and, optionally, an exceptional part to handle exceptions raised during execution of the normal part. The block bodies of function procedures return a result value whereas the block bodies of modules and of proper procedures do not return a result value.

Capacity — The capacity of an array is the number of components of that array.

Cast — To 'transfer' a representation of a value of one type to be a representation of a value of another type. See *type transfer.*

Compatible — The term used in *Programming in Modula-2* for *expression-compatible.*

Compatibility — Compatibility between two types concerns the static semantic rules that define if and how the types may be used in combination. There are some situations in which only the type identical to a particular type may be used; for example, if the formal type of a variable parameter of a procedure is not a system storage formal type and not an open array formal type, then corresponding actual parameters must be variables of the identical type to the formal type. The rules for combining factors of different types in expressions allow for the use in expressions of factors of one of the anonymous basic types (such as the types of numeric literals) and also for the use of procedures as values. In addition to this, the rules for assignment allow for the cases where different types have values in common. As an example, source code is not well-formed if it includes an infix expression in which one factor is of the signed type (INTEGER) and the other factor is of the unsigned type (CARDINAL); however an assignment statement is well-formed even if the assignment is of the value of an expression of one of these types to a variable of the other of these types. Apart from the special rules for system storage formal types and open array formal types, the rules for compatibility of the type of a value formal parameter with the type of an actual expression are those of assignment compatibility. See *Expression compatibility*, *Assignment compatibility*, and *Parameter compatibility.*

Compilation module — The highest syntactic category of the Modula-2 language and thus the start symbol of the concrete syntax. Source code submitted to an implementation is expected to be in the form of one or more compilation modules. The three categories of compilation module are the program module, definition module, and implementation module.

Compilation unit — The term used in *Programming in Modula-2* for a compilation module.

Compiler directive — A form of *Source code directive.*

$\mathbb{C}$ type — A conceptual type whose values are the complex number literals. Although there is no denotation for a complex number literal, a value of the $\mathbb{C}$-type can be formed by applying the standard function CMPLX to two real literals; for example 'CMPLX(0.0, 1.0)' The types COMPLEX and LONGCOMPLEX are both assignment-compatible and expression-compatible with the type $\mathbb{C}$.

Component type — The type of the components of an array structure. For example, in the declaration:

```
TYPE List = ARRAY CHAR OF CARDINAL;
```

The type CARDINAL is the component type.

Computer system — The combination of hardware and, optionally, firmware and software (e.g. operating system) that enables the execution of software.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Conceptual type — A type which is an anonymous basic type and for which there is therefore no denotation. The conceptual types are the types of literal values and of the nil value.

Concrete string constant — A constant of a concrete string type. Such a constant may be denoted either by an array constructor whose array type identifier denotes a concrete string type, and whose structure components are all constant expressions, or by a constant identifier introduced for such a constructed value in a constant definition or constant declaration.

Concrete string type — An array type whose component type is the identical type to the predefined type `CHAR`.

Concrete string variable — A variable of a concrete string type.

Concrete syntax — A grammar for the source code representation of the language, in the form of a set of production rules. The form of the rules in the Standard are as specified by BS 6154 *Method of defining syntactic metalanguage*, 1981. All symbols of the language (i.e. keywords and operators) appear in the concrete syntax.

Concurrent program — A program that is written so that the potential parallelism is explicitly expressed — either using language constructs or library routines. (The language provides concurrency through the interleaving of execution of coroutines — either explicitly using `TRANSFER` or implicitly using `IOTRANSFER`.)

Conforming implementation — A term used to refer to an implementation in contexts where requirements for conformance to this part of ISO/IEC 10514 are stated.

Constructed type — A type constructed as an enumeration, subrange, set, pointer, procedure, opaque, record, or array type. All new types are constructed as one of these types. Of the predefined types, the types denoted by the pervasive identifiers `BITSET` and `PROC`, and the type denoted by `SYSTEM.ADDRESS` are constructed types.

Conversion — A change of representation of a value from one form to another. For example, the input/output library and the strings conversion library provide procedures that convert numbers between character string representations and representations as values of a numeric type. A special case of conversion is *type conversion*, using the standard procedure `VAL` (as distinct from *type transfer* which uses the system procedure `SYSTEM.CAST`).

Coroutines — Parts of an executing program that may transfer execution between themselves; thus a coroutine may transfer execution to another coroutine and may have execution transferred to it by another coroutine. Transfers either occur synchronously on the execution of an explicit call of `COROUTINES.TRANSFER`, or occur asynchronously in response to an interrupt request. Execution transfers to the start of a new coroutine or to a point immediately after the point of the last transfer away from an old coroutine. (A new coroutine is one that has not yet had execution transferred to it. An old coroutine is one that has had execution transferred to it and from which a transfer has been made.) A coroutine's state (the values of its local variables, its state of execution, and its protection) is preserved while execution is transferred away from it.

Declaration — The introduction of an identifier to denote a constant, type, variable, procedure or module, that is made within the declarations of a block. See also *Definitions*.

Declare-before-use — Implementations which insist that all entities except pointer types and procedure types be declared before they are used are described as declare-before-use. (They are often imprecisely called 'single-pass' implementations.) Declare-before-use implementations cannot, for example, process programs that contain local modules which import from each other. Implementations are termed declare-before-use if they satisfy the second predicate in 6.2.7.

Declare-before-use-in-declarations — Implementations which only require that entities used in declarations be declared before they are used are described as declare-before-use-in-declarations. (They are often called 'multi-pass' implementations.) Declare-before-use-in-declarations implementations can, for example, process programs that contain local modules which import from each other. Implementations are termed declare-before-use-in-declarations if they satisfy the first predicate in 6.2.7.

Definition — The introduction of an identifier to denote a constant, type (including opaque type), variable or procedure, that is made within the definitions of a definition module. See also *Declarations*.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Defining occurrence — The occurrence of an identifier in a context in which the meaning of that identifier is given. This will either be in a *definition* or in a *declaration.*

Dimension — The term is applied to an array structure only: it is the number of successive index expressions that are required to access the first non-array component in the structure. That is it is one more than the number of *immediately* nested array structures within an array structure.

Designator — A form of rererence to a variable or a form of reference to a variable value, a constant value, or a procedure value. The simplest form of designator is an identifier, which may be qualified by one or more module names. Designators of arrays may also be indexed to refer to a component, designators of records may be selected to refer to a field, and designators of pointers may be dereferenced. The following are both in the form of designators: (i) `a.b.c.e`, (ii)`List[x+2]^.next`

Dynamic storage — Variable storage that is not allocated automatically as a program executes (see *Static storage*) but which is allocated by a program during its execution. Variables whose storage is allocated dynamically are referenced by dereferenced designators.

Elaboration — The application of the dynamic semantic rules of definitions and of declarations. In the case of variable definitions and declarations, elaboration involves the allocation of storage; in the case of procedure declarations, elaboration involves preparation for the elaboration of the declarations of the procedure block, and the execution of the statements of the procedure block, on calls of the procedure.

Element — A value of a base type that is included in the value of a set type. This term is only used for sets; it is not used in the context of array component values.

Elementary type — A type which has no components that may be denoted as such. The elementary types are the procedure, opaque, pointer, set and scalar types. (See *Type.*)

Empty string literal — A denotation for the empty string and for the string terminator. Two forms are permitted — `""` (adjacent double quotation marks) and `''` (adjacent single quotation marks). Concrete string types are assignment-compatible with the type of an empty string literal (which can thus be used to initialize a concrete string variable to have a string value of length zero). The type `CHAR`, and subranges of the type `CHAR` are also assignment-compatible with the type of an empty string literal (which can thus be used to initialize a character variable to have the value of the string terminator.) An empty string literal is expression-compatible with the type `CHAR`.

Entity — Anything that may be denoted by an identifier: a constant, variable, type, procedure, or module.

Enumeration type — An ordinal type whose values are defined by a list of identifiers that denote the values; the ordering relation for the type is defined by the order in which the values appear in the list.

Environment — An environment associates an identifier with an entity, and with the properties of that entity. The environment which applies at the point of use of an identifier thus determines the visibility and meaning of that identifier at that point of use.

Error — Source code is said to be in error or to contain an error if either the source code is not well-formed (i.e. it does not conform to lexical or concrete syntax rules) or if syntactically correct source code is not well-formed according to static semantic rules. For example, the syntactically correct source code fragment `a1 + a2` is in error if the entity designated by `a1` (or `a2`) is an array.

Exception — A run-time event that is in violation of the dynamic semantic rules of this part of ISO/IEC 10514. For example, an exception occurs if there is an attempt to dereference a pointer variable whose value is `NIL`. An exception should be detected and raised by an implementation, in which case there will be a defined control flow of the program that is different from the normal flow of control.

Exception Raising — The invocation of exception handling, or exceptional termination, following the detection of an exception occurrence. This part of ISO/IEC 10514 requires that the occurrence of certain exceptions be

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

detected and raised by an implementation in standard mode — see 4.7. If an implementation does not detect and raise an exception that has occurred, the effect is *implementation-dependent*; consequently, this part of ISO/IEC 10514 does not give meaning to a program that relies on the detection of an exception if that exception is one for which detection is not required.

Exception handler — The exceptional part of a block body which will be executed if an exception is raised during execution of the normal part of a block body.

Exceptional termination — Program termination that occurs in the state of exceptional execution (as opposed to the state of normal execution).

Export list — The construct that permits identifiers to be used outside a local module in which they have been declared, possibly qualified by the identifier of that module.

Expression-compatible — This term replaces the term 'compatible' which is used in *Programming in Modula-2*.

Two types are said to be expression-compatible if an expression of one type may be used as a factor in an infix expression in which the other factor is an expression of the other type. Not all types may be used in infix expressions, for example there are no infix operators that apply to structured types. Apart from this, identical types are expression-compatible, the identified whole number types and the $\mathbb{Z}$ type are expression-compatible, as are the identified real number types and the $\mathbb{R}$ type, the identified complex number types and the $\mathbb{C}$ type, the character type and the character literal type, and pointer types and the nil type. In addition, a factor of a procedure type may be tested for eqaulity or inequality with an identified procedure value, or procedure constant value, provided that the structure of the procedure corresponds to the structure of the procedure type.

Since the type of expressions is defined in terms of the host type of any subrange type that might otherwise apply, values of subranges of the same host type are therefore expression compatible with each other and with that host type.

Gradual underflow — A term applied to a floating point system that has representation for numbers of too small a magnitude to be represented in normalized form (and hence having a larger relative spacing).

Host type — The host type of a subrange type is that type of which it is directly or indirectly a subrange and which is not itself a subrange type. for numeric subranges, the host type will either be the unsigned type or the signed type; for character subranges it will be the type `CHAR`; for subranges of enumeration types the host type will be the enumeration type. The host type of an ordinal type other than a subrange is that ordinal type itself.

This term replaces the term 'base type' as it is used in *Programming in Modula-2*.

Identical type — Two different type identifiers `t1` and `t2` denote identical types if they have been defined by:

```
TYPE t1 = t2;
```

or by:

```
TYPE t2 = t1;
```

or by:

```
TYPE t1 = t3;
TYPE t2 = t3;
```

or by:

```
TYPE t1 = t4;
TYPE t2 = t5;
```

where `t4` and `t5` denote identical types.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Implementation — The means by which an executable program for a computer system is produced from the components of a Modula-2 program.

A Modula-2 implementation comprises at least the following:

— a translator;

— any necessary run-time support for the predefined entities such as system modules and standard procedures;

— required separate modules;

— any software needed to support or integrate the above.

The term 'translator' is intended to encompass technologies both known (e.g. compilers and interpreters) and those yet to be developed.

Implementation-defined — A term applied to a value that is not defined by this part of ISO/IEC 10514, but that requires definition before the meaning of a program that relies upon that value can be determined. For example, the minimum and maximum values of the `INTEGER` type are implementation-defined, and the value of a call of `SYSTEM.CAST` is implementation-defined. This part of ISO/IEC 10514 requires conforming implementations to be accompanied by documentation which includes definitions of implementation-defined values for each processor.

Implementation-dependent — A term applied to a value or to behaviour that is not defined by this part of ISO/IEC 10514, and that requires no definition since the meaning of a program that relies upon such a definition is not specified by the terms of this part of ISO/IEC 10514. For example, the order of evaluation of operands in an expression is implementation-dependent; consequently, a program that relies on the order of evaluation of operands in an expression is not given meaning by this part of ISO/IEC 10514.

Implementor — The producer of a Modula-2 implementation.

Import list — The construct specifying which of the identifiers that are visible outside the module containing the import list are to be visible within that module, or which identifiers exported by a module are to be visible within the module containing the import list.

Index type — The type of the index of an array structure. For example, in the declaration:

```
TYPE L = ARRAY CHAR OF CARDINAL;
```

`CHAR` is the index type of the type `L`.

Interrupt — A change in state of the system (hardware and/or software), which is caused by an external event, and whose occurrence is communicated asynchronously to a Modula-2 program executing within the system. At the Modula-2 language level, an interrupt results in an asynchronous coroutine transfer.

Interrupt handler — A coroutine that calls `COROUTINES.IOTRANSFER` to transfer to another coroutine while being prepared for an asynchronous transfer back to itself on the occurrence of an interrupt.

Length — The number of characters in a string value. (In the case of a concrete string, the length is equal to the number of characters up to, but not including, any string terminator.)

Level 0 — The outermost declaration level; that is, the declaration level of definitions and the blocks of static modules: program modules, implementation modules and local modules declared in the declarations of static modules. The declaration level of a procedure block is one more than the declaration level of the block in which the procedure is declared. Only procedures declared at level 0 may be assigned as values of procedure variables.

Library — A collection of separate modules supplied for general use.

Library module — A separate module that is part of a library.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Lifetime — The lifetime of a variable or a coroutine is that period of program execution in which the variable or coroutine exists.

The lifetime of a variable declared at level 0 is the entire period of program execution. The lifetime of a local variable of a procedure is the procedure activation for which the variable was allocated. For a variable whose storage is allocated dynamically, the lifetime is the period between allocation and deallocation of that storage.

The lifetime of the main coroutine, being the coroutine by which static modules are initialized, is the entire period of program execution. The lifetime of a coroutine created by a call of `NEWCOROUTINE` extands from that call to the end of program execution.

Local module — A module that is declared in the declarations of a module or procedure block.

Location type — The system basic type that is denoted by the identifier `LOC` exported from `SYSTEM`. Values of this type correspond to the uninterpreted contents of the smallest addressable unit of storage in an implementation. The location type is a *system storage* type.

Member — The means by which the elements of a set constructed value are specified. A member is either a single ordinal expression or an interval comprising two ordinal expressions.

Mutual exclusion — Two or more activities are carried out in mutual exclusion if a subsequent activity may not start until the first activity has completed. Activities that must be carried out in mutual exclusion include those that access common resources or data structures and which (outside of the zone of mutual exclusion) operate concurrently.

The program and the underlying system (eg. peripheral devices) operate in parallel. Where both share access to data, there is a need for mutual exclusion. The language provides an interrupt mechanism which may be used to ensure that a coroutine does not access shared data whilst the device is operating upon it (for the exclusion to be mutual the device should not alter the data after it has delivered an interrupt until its actions are again initiated by the coroutine: but this requirement is outside the scope of this part of ISO/IEC 10514). The wait for the interrupt is programmed by a call to `IOTRANSFER` from the system module `COROUTINES`.

In the presence of interrupts, coroutines operate concurrently, giving rise to the same need for mutual exclusion where data (or any resource) is shared between them. Provided there is no lowering of protection (by a call of `TRANSFER`, `IOTRANSFER` or `LISTEN`), activities within uninterruptible protection domains will exclude coroutines which are waiting for interrupts and which would otherwise execute because an interrupt is pending. For example, a procedure that accesses a data buffer shared with an interrupt handler should execute within an uninterruptible protection domain.

At a more abstract level, the standard library module `Semaphores` provides facilities to express mutual exclusion between processes such as might be provided by the standard library module `Processes`.

Nested module — A term sometimes used to refer to what is called a *local module* in this part of ISO/IEC 10514.

New type — A type that is not predefined. New types are introduced by the use of a new type denoter in the definition or declaration of a type or of a variable. If the new type is the type denoter of a type declaration, that new type will be identified by the identifier of the type declaration. Identified new types are also introduced as opaque types by opaque type definitions.

Non-standard — If a program is such that this part of ISO/IEC 10514 does not ascribe meaning to that program, it may be described as 'non-standard'. Thus, for example, a program that evaluates an uninitialized variable may be described as non-standard.

Normal termination — Program termination that occurs in the state of normal execution (as opposed to the state of exceptional execution).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Object — A value of a VDM-SL type.

Open array — A formal parameter of a procedure that has a formal type which is of an array structure without a specified index type. The component type of an open array formal type may also be an open array formal type. If the component type of an open array is not a *system storage* formal type, and is not an open array type, actual parameters may be of any array type having a component type that is identical to the component type of the open array. In addition, as a special case, a value parameter that is an open array of the character type is parameter-compatible with the string literal type.

Operation completion condition — This is a predicate on the parameters to a procedure in the string library module that changes a string value. If the condition associated with a procedure is true before the procedure is called, the procedure will be able both to complete the change to the string value and to maintain the string abstraction.

Ordinal type — A type for which the permitted values form a discrete set for which there is a complete ordering. Of the basic types, the Boolean, character, unsigned, signed, and $\mathbb{Z}$ types are all ordinal types. Of the constructed types, enumeration types and subrange types are ordinal types. The ordinal number of a value of an ordinal type may be ascertained using the standard function procedure `ORD`.

Order of evaluation — This term is relevant to the *implementation-dependent* manner in which the following may be evaluated:

— operands in expressions;

— actual value parameters;

— index expressions in array designators;

— left and right hand sides of an assignment statement.

An implementation is free to evaluate the factors of an expression in any order; the operators are applied in a defined order. An implementation is free to evaluate actual value parameters in any order on procedure activation. The order of evaluation of index expressions is not defined, nor is whether the left or right hand side of an assignment statement is evaluated first. If it is not the case that the result of an evaluation of the kind described is the same, no matter what order is used, the program is non-standard. (Consequently, programs that include function calls with side-effects may be non-standard.)

Packing — The implementation technique by which two variables of a structured type (array components or record fields) are represented in less than twice the number of storage locations used for one variable. (Packing is only permissible if the semantics of the language are unchanged when it is used. It is unlikely in Modula-2 because a packing implementation would have to allow packed components or fields as actual parameters corresponding to variable formal parameters.)

Parallelism — The execution of more than one part of the program simultaneously. There are two aspects: the use of parallelism as a possible implementation strategy and expression of parallelism by using language constructs or by using the library routines designed for concurrent programming. An implementation may use parallelism if the resultant meaning is one of those allowed by the standard (e.g. in the evaluation of operands of an expression). The constructs in the language only deal with the case of parallelism between the program and interrupt sources and provide a simple method of mutual exclusion between them using `IOTRANSFER`. The library provides routines that enable the expression of parallelism and mutual exclusion between parts of the program known as processes.

Parameter-compatible — For a given kind of argument binding (either as values or as variables), if the formal type of a formal parameter of a procedure is parameter-comatible with an actual type, then actual parameters of that type may be supplied, in a corresponding position to the formal parameter, in calls of the procedure. Apart from special rules that apply in the cases of *open array* formal parameters and *system storage* formal parameters, the type of a value formal parameter is allowed to be assignment-compatible with the type of an actual parameter; the type of a variable formal parameter must be identical to the type of an actual parameter.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Pervasive identifier — An identifier of a predefined procedure, type, or constant that is itself predefined and whose scope covers the entirety of compilation modules, unless overridden by an identifier of the same spelling that is introduced in a definition or a declaration. Thus a pervasive identifier is visible without requiring it to be defined, declared, or imported (into any kind of module). In the scope of an identifier of the same spelling that is introduced in a definition or a declaration, that identifier loses its pervasive nature, and access to the corresponding predefined procedure, type or constant is also lost.

Post-condition — The post-condition of a VDM function asserts what is true of the result of a function if its input parameters satisfy the function's pre-condition.

Pragma — An instruction to an implementation that affects its behaviour. A pragma may instruct the implementation via its translator by appearing in source code, in which case it is called a *source code directive*. A pragma may instruct the implementation via a command issued from the environment that invokes it. (For example, a pragma might be used to direct the linker by the setting of some implementation parameter, or by a parameter to an operating system command.)

Precedence — The rules determining the binding of operators to operands in an expression without parentheses.

Pre-condition — The condition that the input parameters to a VDM function must satisfy for the function to be defined.

Predefined — A term which is applied to an entity (module, procedure, type or constant), or an identifier, that is built-in to Modula-2. Predefined procedures may have parameters that do not follow the usual rules; for example, the actual parameter to a predefined procedure may denote a type. Predefined procedures may not be used as procedure values, hence they may not be specified as the value of procedure constants, procedure variables, or procedure parameters. (See also *Pervasive Identifier*.)

Pre-emptive scheduling — In a situation where an implementation cannot simultaneously execute all active processes it has to adopt a scheduling strategy to share processing time among the competing processes. Pre-emptive scheduling allows the execution of an active process to be forcibly delayed (pre-empted) while other active processes execute. The standard Processes library module requires any scheduling to be pre-emptive to the extent that there may be no executing process of a lower urgency than any process that is ready but has been delayed by the scheduler. The module does not require a pre-emptive scheduling strategy between processes of the same urgency: the implementor is free to provide any policy appropriate for the intended application.

Priority — The term used in *Programming in Modula-2* to express the degree of protection from interrupts. See *Protection.*

Process — A process is an abstraction supported by the concurrent programming library. Writing a program as a set of processes expresses the fact that parts of that program may be executed concurrently (other than in certain sections subject to mutual exclusion).

Processor — The combination of computer system and implementation that permits the translation and execution of Modula-2 programs. A 'processor' may include two computer systems if cross-translation is involved; in this situation an implementation's translator is operated on a **host** computer system and generates code for execution on a **target** computer system. The term 'processor' does not preclude implementations that involve more than one hardware processor.

Program — The combination of a program module and all modules on which it directly or indirectly depends. See 6.1.1.

Protection — The degree to which the execution of a coroutine is protected from interrupts. The initial protection is implementation-defined but may be specified explicitly by the inclusion of a protection expression in the heading of a program module, implementation module, or local module. Such a module is called a protected module. See 6.1.11.

Qualified — An identifier is qualified if it is preceded by the identifier of the module that exports it, the two

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

identifiers being separated by a period character. The concrete syntax uses the term qualified identifier where an identifier may optionally be qualified. See 6.2.2.

ℝ type — A conceptual type whose values are the real number literals. The types `REAL` and `LONGREAL` are both assignment-compatible and expression-compatible with the type ℝ.

Range type — The range type of a subrange is the type that a programmer asserts contains the minimum and maximum values of the subrange. For example:

```
CONST first = 1; last = 10;
TYPE Short = [0..255];
VAR x: Short[first..last];
```

Here `Short` is the range type of the anonymous subrange type denoted by `[first..last]`. Note that a subrange, and its range type share the same *host type*.

The syntactic device of including an identified range type in the denoter of a new subrange type can be used to force the host type of the subrange to be that of the range type. For example, in the following declaration `Sub` is forced to be an `INTEGER` subrange: `TYPE Sub = INTEGER[0..10];` If a range type is not specified, there are rules for determining the default range type. See 6.3.5.

Range check — In contexts where the static semantics allow a type to be assignment-compatible with another type, if the types are not identical, an implementation must carry out appropriate range checks to ensure that the value of the expression falls within the range of values of the other type. Assignment compatibility applies in assignment statements, in value parameter passing, in function procedure return statements, in array indexing, and in value constructors. If the value of the expression is out of range, an exception is raised. The need for range checking applies where the type for which a value is being specified is a subrange type, the signed type, or the unsigned type.

Safe (type) conversion — A term sometimes used for what is called *type conversion* in this part of ISO/IEC 10514.

Scope — The scope of an identifier consists of the parts of the compilation modules of a program in which the identifier is visible. The scope of an identifier declared in a procedure is the procedure block, but this can be restricted by redeclaring it within a nested procedure or by not importing it into a (nested) local module. The scope of an identifier declared in a local module is the module block plus enclosing blocks to which it is exported, but this can be restricted by redeclaring it within a nested procedure or by not importing it into a (nested) local module. The scope of an identifier introduced in a definition includes the definition module in which the definition appears, the module block of the corresponding implementation module, the definitions of definition modules that import the identifier, and the module blocks of other modules that import the identifier.

Separate compilation — The means by which modules may be compiled separately and by which their dependence is controlled.

Separate module — A component of a Modula-2 program, that may be separately developed, translated, and provided, and which comprises an encapsulation of data and/or procedures that are available for use by clients of the module, together with any actions that are executed to initialize the module, and any actions that are executed to finalize the module. The interface to a separate module is provided by a Modula-2 definition module so that other separate modules written in Modula-2 can use its facilities. The executable part of a separate module is specified in Modula-2 by an implementation module; however, an implementation of a separate module need not be derived from Modula-2 source code.

Signed type — The basic whole number type that is identified by the pervasive identifier `INTEGER` and whose permitted values include positive and negative integers.

Signed arithmetic — Integer arithmetic (which permits negative values).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Simple type — A term sometimes used for what is called an *elementary type* in this part of ISO/IEC 10514.

Simple variable — A variable of an elementary type.

Single pass — This term is used to describe a translator that essentially scans a source program only once. (Thus a single pass translator cannot deal with declaration after use nor can it deal with mutually importing modules; a single pass translator can only accept mutually importing procedures if they appear in definition modules or if one is declared in a forward declaration.)

Smallest addressable unit — The unit of storage for an implementation such that for a type whose variables occupy a smallest addressable unit, the `TSIZE` of the type is 1.

Source code — A fragment of text that is potentially part of the text representation of a component of a Modula-2 program. The term 'potentially' is used since source code may contain errors which would mean that the text was not well-formed.

Source code directive — A type of pragma that is an instruction issued in *source code* to an implementation that affects its behaviour. (For example, a source code directive might switch on optimization, or switch to standard mode.) This part of ISO/IEC 10514 defines the syntax for source code directives — see 5. A conforming implementation is required to process source code directives in such a way that they have no effect on the meaning of the program other than delimiting tokens, or by determining an implementation-defined value A conforming implementation is also required to detect (and issue a warning concerning) a source code directive whose body is not recognized.

Sourced implementation module — The implementation of a separate module for which source code is provided and whose semantics is therefore determined by the semantics of that source code as specified by this part of ISO/IEC 10514.

Standard — If a program is such that this part of ISO/IEC 10514 ascribes meaning to that program, it may be described as 'standard'.

Standard library — The library consisting of the separate modules specified in this part of ISO/IEC 10514: the required modules in section 8 and the non-required modules in section 9.

Standard mode — A mode of use of an implementation in which it complies with the requirements of this part of ISO/IEC 10514.

Standard procedure — A predefined proper procedure or function procedure for which the predefined identifier is a pervasive identifier.

Static storage — Variable storage that is allocated and deallocated automatically as a program executes; the code to allocate the storage is generated by a translator as the result of analyzing the static program text. Variables whose storage is allocated statically are declared in variable definitions or variable declarations and are referenced by entire designators.

Step — The number of ordinal values by which a for loop control variable is to be incremented (or decremented, if negative) on each iteration of a for loop.

String constant — An identifier that denotes a string literal. (The term is not used to describe a structured constant of a concrete string type.)

String literal — A denotation of a string value that includes a sequence of printable characters between either a pair of single quotation marks (`''`) or a pair of double quotation marks (`""`).

String literal type — A conceptual string type (like the numeric types $\mathbb{Z}$ and $\mathbb{R}$) which includes strings of all lengths (in particular, a single quoted character is of the string literal type). Its purpose is to provide a conceptual basis for compatibility rules for string literals.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

String terminator — The string terminator is a value of the predefined type CHAR that is used to terminate a string value in a concrete string variable when the length of the string it represents is less than the capacity of the variable. The value is implementation-defined but may always be represented using an *empty string literal.* The formal definition uses the literal END-OF-STRING-CHAR to stand for the string terminator.

String value — A value of an abstract data type which is a sequence of characters. String values may either be represented as string literals or as a sequence of CHAR values in a concrete string (beginning at the array's lowest index and ending either at the array's highest index or at the first string terminator).

Structured (composite) type — Array types and record types are structured types in the sense that values or variables of such types are compositions of other constituent values or variables which may be denoted as such. The term structured type has been adopted from *Programming in Modula-2*. However, constructed types other than array types and record types also have a structure. For example, procedure types have a specified number of parameters of specified types. Where there might be ambiguity, array and record types are also referred to as composite types in this part of ISO/IEC 10514.

Subrange type — An ordinal type whose sequence of values is a subsequence of a denoted ordinal type.

System module — A predefined module that is an encapsulation of predefined entities whose identifiers are only available by import from that system module. Unlike separate modules, system modules cannot be replaced. A translator can deal with access to a system module in a special way (e.g. 'calls' to system module 'procedures' may be translated as inline code).

System storage — The system storage types are the types such as the *location type* whose values correspond to the uninterpreted contents of storage locations, and whose type identifiers are exported from SYSTEM. In addition to the location type (denoted by LOC) there is also the word type (denoted by WORD) and possibly the byte type (denoted by BYTE) if appropriate for the implementation. When used as system storage formal types, these types permit a relaxation of the rules for parameter compatibility.

System storage formal types are either system storage types or zero-based arrays of a system storage type. Corresponding actual parameters may be of any type for which variables occupy the same number of storage locations as variables of the formal type. If a formal type is a one-dimensional open array of a system storage formal type, actual parameters may be of any type for which variables occupy a multiple of the number of storage locations occupied by variables of the system storage formal type. See 6.9.3.3.

Termination — The actions taken by a program as it terminates. These include the finalization of static modules but otherwise depend upon the event that causes termination to start. See *Normal termination*, *Exceptional termination* and 6.1.7.

Threaten — The control variable of a for statement would be threatened if there was a possibility that the value of the control variable could be altered, while the for statement was being executed, by a statement other than the for statement itself. . Thus a control variable would be threatened by directly assigning to it, by attempting to pass it as an actual variable parameter, or by manipulating it via its address obtained using SYSTEM.ADR.

Type — Every value and every variable has a type. The type of a value determines the operations that may be applied to that value. The type of a variable determines the set of values that may be assigned to that variable and determines the other operations that may be applied to that variable.

Type is also an entity that may be denoted by a predefined type identifier or a type identifier introduced in a type definition or type declaration. The identifier of a type is sometimes informally referred to as the type that it denotes; for example, in a reference to the 'type' CHAR, what is intended is a reference to the character type which is denoted by the pervasive identifier CHAR. The identical type may also be denoted by other type identifiers — see *Identical type.*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Types are classified as shown in the table below.

<table>
<tr><td rowspan="10">basic</td><td rowspan="9">elementary</td><td>system</td><td colspan="2">predefined</td><td rowspan="2" colspan="3">identifiers for new types are introduced in a type definition or declaration such as
TYPE T = (a, b, c)</td></tr>
<tr><td></td><td>SYSTEM.LOC etc.</td></tr>
<tr><td></td><td>string literal
nil</td><td></td><td colspan="3" rowspan="2"></td></tr>
<tr><td>complex number</td><td>ℂ</td><td>COMPLEX
LONGCOMPLEX</td></tr>
<tr><td>real number</td><td>ℝ</td><td>REAL
LONGREAL</td><td colspan="2"></td><td rowspan="4">scalar</td></tr>
<tr><td>whole number</td><td>ℤ</td><td>CARDINAL
INTEGER</td><td colspan="2" rowspan="3"></td><td rowspan="3">ordinal</td></tr>
<tr><td></td><td>character literal</td><td>BOOLEAN
CHAR</td></tr>
<tr><td rowspan="3">constructed</td><td>enumeration
subrange</td><td></td><td></td><td>(a, b, c)
[0 .. 9]</td></tr>
<tr><td rowspan="2">elementary</td><td>set
packedset
pointer
procedure</td><td></td><td>BITSET
SYSTEM.ADDRESS
PROC</td><td>SET OF ordinal
PACKEDSET OF ordinal
POINTER TO denoted
PROCEDURE (T ...)</td></tr>
<tr><td>opaque</td><td></td><td></td><td>—</td><td>T = ;</td></tr>
<tr><td>structured (composite)</td><td>record
array</td><td></td><td></td><td>RECORD ... END
ARRAY ordinal
OF denoted</td></tr>
<tr><td colspan="3"></td><td>anonymous</td><td>identified</td><td>anonymous</td><td>identified</td></tr>
<tr><td colspan="3"></td><td colspan="2"></td><td colspan="2">new</td></tr>
<tr><td colspan="3"></td><td>not denoted</td><td colspan="3">denoted</td></tr>
</table>

Type conversion — The translation of a value of one type into a value of another type according to some logical relationship between the types. (Typically, the relationship is numerical and the conversion implies a change in the underlying representation of the original value.) A run-time check is carried out to ensure that the conversion is possible. In this part of ISO/IEC 10514, type conversion is provided by the standard function `VAL` and by standard functions such as `ORD` which are defined in terms of `VAL`. See also *Type transfer.*

Type transfer — The interpretation of a bit pattern representing the value of one type as a value of another type without any run-time check to ensure that the bit pattern is a valid value of the target type. In this part of ISO/IEC 10514, type conversion is provided by `SYSTEM.CAST`. See also *Type conversion.*

Unsafe (type) conversion — A term sometimes used for what is called *type transfer* in this part of ISO/IEC 10514.

Unsigned arithmetic — Whole number arithmetic that excludes negative values (i.e. cardinal arithmetic).

Unsigned type — The basic whole number type that is identified by the pervasive identifier `CARDINAL` and whose permitted values include only non-negative integers.

Urgency — In the standard library module `Processes`, A measure that is used to select processes for execution when there are more processes that are currently using the processor or are eligible to use the processor than can execute concurrently. The scheduler is required to ensure that it cannot be the case that a process eligible to use the processor has an urgency greater than one of the processes currently doing so.

Variable — A location, or set of locations in the case of a structured (composite) type, that can be used to hold values of a specified type. Values to be held are assigned in assignment statements and are referenced with value designators. These are abstract locations which may be mapped onto actual storage locations when the variable or its value is interpreted as a variable or value of a system storage type, or an array of a system storage type.

VDM — Vienna Development Method.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

VDM-SL — Vienna Development Method Specification Language. The dialect used in this part of ISO/IEC 10514 is an extended version of that specified in ISO/IEC 13817.

Visible — If a point where identifiers may be used is included in

that identifier is said to be visible at that point of use, whether or not its meaning is known.

Well-formed — Meets the static semantic rules of the language and, as a prerequisite to these, meets the syntax rules.

Well-ordered — A module is well-ordered if the declarations within it are well-ordered and if its statements are well-ordered. Declarations are well-ordered if they do not use identifiers that are textually declared later. (The requirement of declaration-before-use is relaxed for the declaration of a pointer type and a procedure type.) Depending on the implementation strategy used, statements are either always well-ordered, or are well-ordered if they do not use identifiers that are declared textually later. (A statement that references a pointer variable whose type has not yet been fully declared is well-ordered if the reference is to the entire variable — a similar constraint to that imposed on a client module using a variable of an opaque type.)

Workspace — The storage used by a procedure activation or by a coroutine. This includes storage allocated for variables and storage used by the implementation for linkage and control information, and for the evaluation of expressions.

$\mathbb{Z}$ type — A conceptual type whose values are the whole number literals. The types `INTEGER` and `CARDINAL` are both assignment-compatible and expression-compatible with the type $\mathbb{Z}$.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex F
(informative)
Participating Individuals and Organisations

This part of ISO/IEC 10514 was prepared by Roger Henry, as project editor, with the assistance of Derek Andrews, principal author of the formal definition, on behalf of ISO/IEC JTC1/SC22/WG13. The following individuals participated in the process of preparation as the nominated experts of ISO/IEC JTC1/SC22/WG13:

Name	*Country*	*Organisation*
Derek Andrews	Principal author	University of Leicester, UK
Frank Bender	Germany	Bender Softwareberatung
Günther Blaschek	Austria	Universität Linz
Randy Bush	USA	Pacific Systems Group
Steven Cater	USA	University of Georgia
Barry Cornelius	UK	University of Durham
Günter Dotzel	Germany	ModulaWare GmbH
Eberhard Enger	Germany	Universität München
Jacques Farré	France	Université de Nice
John Gough	Australia	Queensland University of Technology
Elmar Henne	Germany	p1 GmbH
Roger Henry	Project Editor	Informed Computing, UK
Keith Hopper	New Zealand	University of Waikato
R.W. Hutchinson	USA	
K.N. King	USA	Georgia State University
Jens Kolind	Denmark	Dansk Standardiseringsrad
Herbert Klaeren	Germany	Universität Tübingen
John Lancaster	UK	Aspex Microsystems Ltd
David Lightfoot	UK	Oxford Brookes University
Albert Meier	Switzerland	A+L AG
Peter Moylan	Australia	University of Newcastle
Tom Pittman	USA	Itty Bitty Computers
Cornelis Pronk	Netherlands	Delft University of Technology
Wolfgang Redtenbacher	Germany	Redtenbacher Software
Tom Reid	USA	National University of Singapore
Martin Schönhacker	Austria	Technical University of Vienna
Marius Schoorel	Netherlands	ACE Associated Computer Experts bv
Brian Shearing	UK	The Software Factory
Rick Sutcliffe	Canada	Trinity Western University
Masato Takeichi	Japan	University of Tokyo
Richard Thomas	Australia	Queensland University of Technology
Russ Tront	Canada	Simon Fraser University
Don Ward	UK	Careful Computing
Brian Wichmann	UK	National Physical Laboratory
Daryl Wick	Canada	Victoria University
Albert Wiedemann	Germany	Gisela Gymnasium, München
Martin Whitaker	UK	Aspex Microsystems Ltd
Mark Woodman	Convenor	The Open University, UK
Kinoshita Yoshiki	Japan	Electrotechnical Laboratory
Zheng Guo liang	Peoples' Rep. China	Nanjing University

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The following people are past-members of WG13 or observers whose assistance is gratefully acknowledged. (The affiliation given is as it was last known.)

Name	*Country*	*Organisation*
Elmar Baumgart	Germany	ModulaWare GmbH
Chuck Bilbe	USA	Sun Microsystems
Susan Eisenbach	UK	Imperial College of Science and Technology
Boris Lvin	Russia	Voronezh State University
Frode Odegard	Norway	Odegard Labs, USA
Stan Osborne	USA	ANA Systems
Dan Resler	USA	Virginia Commonwealth University
Dmitry Sagatelyan	Russia	Moscow General Physics Institute
Martin Schleußer	Germany	RST GmbH
Willi Steiger	Switzerland	
John Souter	UK	BSI QA
Thomas Tempelmann	Germany	
Pat Terry	South Africa	Rhodes University

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Key to the Index

The index entries include terms from the lexis and concrete syntax (key cs); pervasive identifiers (key pi); the names of system modules (key sm), system procedures (key sp), library modules (key lm), library procedures (key lp); VDM-SL types (key vt), values (key vv), functions (key vf), and operations (key vo); the names of exceptions that may occur (key ex); and terms from the natural language definition (no key).

Certain prefixes to VDM-SL functions and operations should be ignored when searching the index alphabetically. These prefixes are *a-* (storage allocation), *is-* (tests), *m-* (meaning), *t-* (type), and *wf-* (well-formed). Note that function applications appearing in the body of the document as, for example, *is-Address-value*, where the VDM-SL identifier following the prefix is capitalized, do not appear in the index. These are 'is expressions' which test against the specified VDM-SL type.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Index

ℂ-type, 273
ℤ-type, 272
ℝ-type, 273
$\rho_{COROUTINES}$ vv, 369
$\rho_{EXCEPTIONS}$ vv, 381
$\rho_{M2EXCEPTION}$ vv, 393
ρ_{PERV} vv, 270
ρ_{SYSTEM} vv, 351
$\rho_{TERMINATION}$ vv, 389
ρ_{empty} vv, 271
$\rho_{sys\text{-}modules}$ vv, 347

ABS pi, 260, 288, 290, 290, 291
abs vo, 424
abs lp, 424, 424, 427
m-abs-designator vo, 290, 291
t-abs-designator vf, 289, 290
wf-abs-designator vf, 289, 290
abstract data type, 671
abstract implementation module, 5, 5, 8, 30, 33, 36, 482, 495, 509, 519, 526, 580, 594, 596, 602, 637
Abstract-implementation-module vt, 33, 652
m-abstract-implementation-module vo, 31, 33
wf-abstract-implementation-module vf, 30, 33
Abstract-module-finalization vt, 334
wf-abstract-module-pair vf, 34, 36
wf-abstract-modules vf, 25, 25
access-environment vf, 65, 71, 224, 281, 289, 319, 323, 354, 370, 377, 382, 385, 390, 394
Activate vo, 550
Activate lp, 544, 548, 550, 550, 551
activate-entry-protection vo, 51, 127, 128, 130, 131, 133–140
activate-exit-protection vo, 51, 127, 128, 130, 131, 133–140
is-active-component vo, 183, 186, 186
actual parameter, 9, 61, 62, 75, 78, 114, 121, 123, 152, 153, 169, 170, 229, 230, 253, 254, 258, 260–262, 264, 265, 267, 281, 283, 285–287, 290–304, 352, 355–357, 359–361, 363, 364, 371–374, 376–379, 382–387, 394, 434, 548, 560, 566, 577, 661, 667, 668, 671
actual parameter cs, 258, 258, 650, 650
actual parameter list cs, 258, 258, 650, 650
actual parameters cs, 152, 229, 258, 645, 649, 650
Actual-parameter vt, 258, 652
m-actual-parameter vo, 259, 259
Actual-parameters vt, 258, 652
m-actual-parameters vo, 153, 230, 259, 259
wf-actual-parameters vf, 152, 230, 258
wf-actual-parameter vf, 258, 259
actually-detected vf, 397, 397
add-func-parameters vf, 77, 78, 78
add-offset vo, 355, 356, 357, 358
add-proc-parameters vf, 74, 75
add-protection vf, 28, 32, 82, 318
add-to-constants vf, 95, 95
add-to-store vo, 345, 346
ADDADR sp, 350, 353–355
m-ADDADR vo, 354, 355
t-ADDADR vf, 354, 355
wf-ADDADR vf, 354, 355
Address vt, 333
address arithmetic, 348, 350, 353, 354, 358
ADDRESS-ARITHMETIC ex, 355, 357, 358
ADDRESS-ARITHMETIC-ON-NIL ex, 355, 357, 358
ADDRESS-CONSTRUCTION ex, 359
address-difference vo, 358, 358
address-of vo, 360, 360, 453, 454
address-type-structure vv, 353
is-address-type vf, 70, 125, 263, 324
Address-value vt, 333
is-adequate-work-space vo, 375, 376
ADR sp, 170, 259, 350, 353, 359, 360, 671
m-ADR vo, 354, 360
t-ADR vf, 354, 360
wf-ADR vf, 354, 360
alias, 62, 267, 268, 375, 592
alias-machine-address vo, 71, 336
Alias-relation vt, 332
alias-set vf, 336, 336
alignment, 364, 365, 399, 581
all-storage-pools vf, 335, 336
ALLOCATE vo, 400
ALLOCATE lp, 287, 288, 400, 400, 401–403
allocate-a-variable vo, 71, 71
allocate-abstract-storage vo, 337, 438, 451–453
allocate-coroutine-id vf, 345
allocate-coroutine-id vo, 345, 375
allocate-device-id vo, 534, 539
allocate-heap-storage vo, 400, 403
allocate-procedure-id vo, 339, 340
allocate-program-storage vo, 71, 267, 269, 337, 453, 454
allocate-source vo, 342, 383, 403, 421, 429, 470, 510, 555, 563, 598, 605
allocate-tagged-variables vf, 118, 118
allocate-variables vo, 71, 71
allocate-variant-variables vf, 118, 119
AllocateDeviceId vo, 534
AllocateDeviceId lp, 531, 534, 534
AllocateSource sp, 381, 382
m-AllocateSource vo, 382, 382
wf-AllocateSource vf, 382, 382
allRight vv, 449
alreadyOpen vv, 475
alternative-exponent-character vv, 450
and keyword cs, 19
and synonym cs, 19

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

angle vf, 425–427, 429
anonymous type, 68, 70, 91, 99, 231, 271, 303, 660–662, 669
Answer vt, 344
Append vo, 573
Append lp, 9, 572, 573, 573
applys vf, 614
arccos vo, 419
arccos lp, 419, 419, 427, 427
arcsin vo, 419
arcsin lp, 418, 419, 426, 426
arctan vo, 419
arctan lp, 419, 419, 424, 427, 427
arg vo, 424
arg lp, 424, 424, 427
ArgChan vo, 523
ArgChan lp, 523, 523
Argument vt, 258
argument, 61, 75, 78, 153, 230, 258–260, 265–267, 306, 400, 425, 427, 431, 472, 522, 523, 527, 660, 667
Arguments vt, 258
arithmetic negation operator cs, 19
Arithmetic-prefix-operation vt, 220, 652
arithmetic-prefix-operation-result-type vf, 220, 221
m-arithmetic-prefix-operation vo, 220, 221
t-arithmetic-prefix-operation vf, 220, 220
wf-arithmetic-prefix-operation vf, 219, 220
Arithmetic-prefix-operator vt, 221, 652
m-arithmetic-prefix-operator vo, 221, 221
array assignment, 122, 123, 125, 149, 184, 226, 266, 466, 467, 566, 582, 661, 667, 669
array constructed value cs, 232, 233, 235, 649, 649
array constructor cs, 231, 232, 649, 649
array parameter, 104, 105, 122, 123, 231, 262–264, 266, 268, 279, 294, 302, 328, 383, 566, 661, 667, 671
array type, 91, 106, 107, 116, 123, 184, 226, 230–233, 238, 243, 262, 263, 279, 295, 321, 325, 326, 564, 660, 662, 667, 671
array type cs, 93, 106, 642, 644
array type identifier cs, 232, 232, 649, 649
array value cs, 225, 225, 648, 648
array variable designator cs, 184, 184, 647, 647
Array-components vt, 312
a-array-components vf, 116, 116, 117
Array-constructed-value vt, 233, 652
Array-constructor vt, 232, 652
m-array-constructor vo, 231, 232
t-array-constructor vf, 231, 232
wf-array-constructor vf, 231, 232
wf-array-definition vf, 232, 233, 236
Array-formal-type vt, 104, 652
d-array-formal-type vf, 105
wf-array-formal-type vf, 105, 106, 106
array-string-assignment vo, 150, 150
Array-structure vt, 310
a-array-structure vf, 114, 116
Array-type vt, 106, 652
d-array-type vf, 94, 107
is-array-type vf, 124, 184, 226, 233, 263, 264, 297, 321, 321, 328, 383, 384
wf-array-type vf, 94, 107
Array-value vt, 334
array-value-assignment vo, 150, 150
Array-variable vt, 312
array-variable-assignment vo, 150, 150, 568–574, 585, 586, 590, 591
Assign vo, 568
Assign lp, 567, 567, 567
assign vo, 149, 150, 150, 151, 187, 267, 269, 282, 285, 287, 343, 372, 375, 376, 383, 384, 400, 401, 436, 437, 440–442, 445, 453, 454, 477–481, 486–488, 490, 491, 493, 494, 500, 501, 504–508, 513–518, 524–526, 534, 542, 543, 549, 551, 560, 576–578, 584, 585, 589, 594, 596, 601, 602, 609
ASSIGN-RANGE ex, 150
assignment
  of arrays, *see* array assignment
  of records, *see* record assignment
  of strings, *see* string assignment
  of tag field, 149, 151, 240, 367
  of undefined values, 149, 167, 224
assignment operator cs, 18, 148, 165, 645, 647, 650
assignment statement cs, 146, 148, 645, 645
is-assignment-compatible vf, 123, 149, 155, 166, 176, 184, 211, 226, 236, 243, 263, 285, 287
Assignment-statement vt, 148, 652
m-assignment-statement vo, 147, 149
wf-assignment-statement vf, 146, 149
associated-constant-value vf, 314
associated-local-environment vf, 32, 317
associated-module vf, 28, 33, 35, 43, 45, 134, 137, 139, 317
associated-structure vf, 315
associated-variable vf, 183, 315
ATTACH sp, 369–372
Attach vo, 552
Attach lp, 548, 551, 552
attach vo, 343, 371
is-attached vo, 343, 372
m-ATTACH vo, 370, 371
wf-ATTACH vf, 370, 371
is-available vo, 400, 400

bad-dens vv, 541
bad-ops vv, 541
bad-props vv, 541
badchan vf, 464, 477, 478, 486–488, 490, 500, 501, 504, 513–515, 534, 538
badtable vv, 541
base type, 15, 91, 95, 97–99, 210, 243–245, 247, 279, 285, 287, 325, 361–363, 660, 663, 664

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

base type cs, 97, 97, 98, 643, 643
base-type-of vf, 211, 242, 243, 285, 287, 325, 362, 363
Basic-type vt, 308
a-basic-type vf, 114, 115, 120
Basic-value vt, 333
bind-argument vo, 265, 266
bind-argument-to-value-parameter vo, 266, 267
bind-argument-to-variable-parameter vo, 266, 268
bind-arguments vo, 75, 79, 265, 266
bind-open-array-value vo, 267, 267
bind-open-array-variable vo, 268, 268
bind-value vo, 267, 267
bind-variable vo, 268, 268
bit, 3, 11, 15, 92, 98, 99, 279, 350, 351, 360–365, 408, 409, 430, 614, 672
BITSET pi, 15, 92, 97, 98, 242, 279, 279, 362, 363, 662
bitset type, 279
bitset-structure vv, 279
bitsperloc vv, 352
bitsperloc-constant-value vv, 352
block, 23, 27, 28, 31–33, 40, 41, 43, 44, 46, 47, 50, 52, 54, 58, 61–67, 70–81, 122, 126, 128–132, 135–145, 153–156, 168, 177, 229, 230, 258, 266, 267, 285, 307, 334, 374, 661–666, 669
block body cs, 126, 129, 131, 142, 644, 645, 645
Block-body vt, 142, 652
wf-block-body vf, 126, 129, 132, 142
BOOLEAN pi, 9, 92, 274, 274, 661
boolean expression cs, 158, 158, 162, 646, 646, 646
Boolean type, 92, 158, 162, 163, 202, 207, 208, 210, 213, 215–217, 222, 271, 274, 274, 300, 322, 325, 386, 387, 390, 391, 395
m-Boolean-and vo, 202, 203
Boolean-operation vt, 202, 652
m-Boolean-operation vo, 193, 202
t-Boolean-operation vf, 193, 202
wf-Boolean-operation vf, 192, 202
Boolean-operator vt, 202, 652
m-Boolean-operator vo, 202, 202
m-Boolean-or vo, 202, 203
Boolean-prefix-operation vt, 222, 652
m-Boolean-prefix-operation vo, 220, 222
t-Boolean-prefix-operation vf, 220, 222
wf-Boolean-prefix-operation vf, 219, 222
Boolean-prefix-operator vt, 222, 652
bound type, 99, 100, 115, 187, 228, 283, 284, 287, 288, 325, 326
bound type cs, 99, 99, 643, 643
bound-type-of vf, 187, 228, 284, 288, 326
bounds, 96, 98, 264, 272, 361
build-a-shielded-env vf, 89, 89
build-and-raise-exception vo, 396, 397, 398
build-array-type vf, 269, 269
build-fields-list-structure vf, 108, 108, 110, 112
build-shielded-environment vf, 55, 65, 89
build-variant-structure vf, 110, 110
byte-array-structure vv, 353
BYTE-is-implemented vv, 351
byte-range-structure vv, 353

c-exit vo, 127, 128, 130, 131, 134, 136, 138, 140, 144, 145, 154–156, 164, 167, 257, 345
c-tixe vo, 26, 75, 79, 127, 128, 130, 131, 143–145, 164, 167, 345
calculate-address vf, 336, 336
call-function-procedure-value vo, 230, 339
call-proper-procedure-value vo, 153, 284, 288, 339, 375, 452–454, 464–468
can-add vo, 355, 356, 358
can-allocate-storage vf, 337, 346
can-create-sema vo, 560, 560
can-do-lines vo, 514, 514
can-do-single-chars vo, 514, 514
can-make vo, 359, 359
can-make-a-channel vo, 540, 540
can-reread vo, 489, 490
can-rewrite vo, 490, 490
can-subtract vo, 357, 357
can-write-here vo, 503, 503
CanAppendAll vo, 572
CanAppendAll lp, 572, 572
CanAssignAll vo, 567
CanAssignAll lp, 567, 567, 567
CanConcatAll vo, 573
CanConcatAll lp, 573, 573
CanDeleteAll vo, 569
CanDeleteAll lp, 569, 569
CanExtractAll vo, 568
CanExtractAll lp, 568, 568
CanGetClock vo, 608
CanGetClock lp, 608, 608
CanInsertAll vo, 570
CanInsertAll lp, 567, 570, 570
CanReplaceAll vo, 571
CanReplaceAll lp, 571, 571
CanSetClock vo, 608, 609
CanSetClock lp, 608, 608
CAP pi, 288, 291, 574
cap vf, 574, 575
m-cap-designator vo, 290, 291
t-cap-designator vf, 289, 291
wf-cap-designator vf, 289, 291
capacity vf, 436, 437, 448, 453, 567, 568, 570, 573, 574, 579
capitalizations vv, 292
Capitalize vo, 574
Capitalize lp, 574, 574, 575
CARDINAL pi, 92, 271, 271, 301, 304, 305, 350, 361, 545, 548, 582, 592, 608, 661, 665, 672, 673
cardstr vf, 442, 586, 586, 597
CardToStr vo, 586
CardToStr lp, 585, 586
Case vt, 161, 652
case alternative cs, 159, 161, 646, 646

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

case else part cs, 159, 159, 646, 646
case label, 159–161
case label cs, 161, 161, 646, 646
case label list cs, 161, 161, 646, 646
case list cs, 159, 159, 646, 646
case selector cs, 159, 159, 646, 646
case separator cs, 18, 110, 159, 644, 646, 650
case statement cs, 146, 159, 645, 646
CASE-RANGE ex, 160
Case-statement vt, 159, 652
m-case-statement vo, 147, 160
wf-case-statement vf, 146, 160
wf-case vf, 160, 161
CAST sp, 70, 217, 253, 260, 350, 353, 364, 365, 662, 672
cast-value vo, 269, 364, 365
m-CAST vo, 354, 364
t-CAST vf, 354, 364
wf-CAST vf, 354, 364
catenation, *see* string concatenation
cenv vv, 345
chan-exception vo, 470, 479, 491, 505, 516, 524, 529, 530, 534, 535, 539, 542
chan-exception-detected vf, 470, 471, 471
chan-exception-string vo, 470, 471
chan-exception-using vo, 471, 535
Chan-id vt, 536
ChanConsts lm, 431, 472, 474
ChanException vo, 469
ChanException lp, 469, 469
chanexceptions vv, 470
chanflags vv, 474
Change, 16–21, 34, 39, 41, 51, 73, 76, 95, 97, 98, 104, 121, 123, 129, 165, 189, 198, 205, 231, 242, 252, 264, 273, 274, 280, 288, 294, 302, 304, 347, 350, 364, 370, 372, 374, 545
change-value vo, 150, 151, 335
channel, *see* I/O channel
CHAR pi, 14, 92, 149, 274, 274, 292, 383, 412, 575, 661–665, 671
Char vt, 274
character literal, 121, 123, 124, 155, 250, 271, 274, 278, 279, 322, 324, 664
character number literal cs, 21
character type, 274
wf-character-assignment vf, 123, 124
is-character-compatible vf, 121, 122
is-character-constant-type vf, 122, 124, 322, 324
Character-number-literal vt, 252, 653
m-character-number-literal vo, 251, 253
t-character-number-literal vf, 251, 252
wf-character-number-literal vf, 251, 252
Character-string vt, 251, 653
m-character-string vo, 251, 252
t-character-string vf, 251, 251
wf-character-string vf, 251, 251
is-character-type vf, 122, 263, 322, 323
characters vv, 275
CharClass lm, 399, 412
chars-up-to vo, 479, 492, 506, 520, 526, 530
check-changes vf, 335, 335
check-flags vo, 478–481, 491, 492, 494, 504–508, 515–518, 523–525, 529
check-for-termination vf, 340, 340
check-halting vo, 341, 391
check-termination vo, 341
CHR pi, 288, 292, 292
m-chr-designator vo, 290, 292
t-chr-designator vf, 289, 292
wf-chr-designator vf, 289, 292
Claim vo, 561
Claim lp, 560, 561, 561
Clarification, 29, 34, 49, 95, 102, 105, 123, 160, 169, 206, 214, 215, 243, 254, 262, 264, 281, 286, 290, 291, 294, 298, 299, 375, 400
classification of characters, 412, 593
client, 38
Close lp, 478, 478, 490, 490, 504, 504, 514, 514
closure, 48, 49
Cmdcont vt, 344
CMPLX pi, 17, 247, 260, 273, 288, 292, 292, 293, 661
m-cmplx-designator vo, 290, 293
t-cmplx-designator vf, 289, 293
wf-cmplx-designator vf, 289, 293
collating sequence, 292, 575
collation-map vv, 276
colon cs, 17, 650
combine-definition-and-imported-environments vf, 32, 320
comma cs, 17, 650
comment cs, 21
comment body cs, 21
is-comparable-addresses vo, 358, 358
Compare vo, 575
Compare lp, 574, 575, 575
compareresults vv, 580
compatibility
  assignment compatibility, 58, 68, 91, 99, 121–123, 149, 155, 165, 184, 188, 210, 226, 228, 236, 240, 243, 252, 262, 266, 274, 279, 285, 287, 324, 347, 350, 353, 367, 660, 661, 663, 667, 669, 673
  expression compatibility, 68, 91, 96, 99, 111, 121, 122, 149, 155, 159, 160, 165, 184, 207, 208, 226, 236, 240, 245, 252, 262, 274, 279, 285, 287, 324, 350, 352, 355–357, 361, 363, 367, 373, 374, 383, 660, 661, 663, 664, 669, 673
  parameter compatibility, 68, 91, 99, 121, 123, 152, 188, 228, 229, 261–264, 266, 267, 279, 352, 353, 355–357, 361, 363, 373, 374, 383, 661, 667, 669, 671
is-compatible-case-labels vf, 160, 161
is-compatible-variant-labels vf, 111, 114
compilation module, 2, 4, 10, 11, 23–25, 34, 36–38, 40, 347, 635, 668, 669

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

compilation module cs, 23, 639
compilation order, 23, 24, 28, 30, 32, 34, 36–38
Compilation-module vt, 24, 653
m-compilation-module vo, 26, 26
compilation-orders vf, 24, 26, 36, 38
Compilation-unit vt, 36
compiled-definition-modules vf, 37, 37
compiler
  multi–pass, 6, 11, 662
  single–pass, 6, 11, 662, 670
compiler directive, *see* source code directive
is-complete-declarations vf, 65, 65
is-complete-definitions vf, 55, 55
is-complete-order vf, 37, 37
is-complete-variant-labels vf, 111, 114
COMPLEX pi, 17, 92, 273, 273, 415, 423, 661
complex conjugate, 424
complex type, 92, 124, 194, 273, 273, 274, 293, 295, 301
m-complex-add vo, 195, 195
complex-approximation, 194
complex-approximation vf, 194, 194
complex-arccos vo, 427
complex-arcsin vo, 426
complex-arctan vo, 427
wf-complex-assignment vf, 123, 124
Complex-comparison-operation vt, 207, 653
m-complex-comparison-operation vo, 207, 207
t-complex-comparison-operation vf, 206, 207
wf-complex-comparison-operation vf, 206, 207
Complex-comparison-operator vt, 207, 653
m-complex-comparison-operator vo, 207, 208
is-complex-compatible vf, 121, 122
complex-component-type vf, 295, 296, 302
complex-constant-approximation vf, 194, 194
complex-cos vo, 426
m-complex-divide vo, 195, 195
m-complex-eq vo, 208, 208
complex-exp vo, 425
complex-ln vo, 425
m-complex-multiply vo, 195, 195
m-complex-ne vo, 208, 208
Complex-number-operation vt, 193, 653
m-complex-number-operation vo, 193, 194
t-complex-number-operation vf, 193, 193
wf-complex-number-operation vf, 192, 193
Complex-number-operator vt, 194, 653
m-complex-number-operator vo, 194, 195
Complex-number-type vt, 309
COMPLEX-OVERFLOW ex, 194
complex-power vo, 425
complex-result-type vf, 193, 194, 293
complex-sin vo, 426
complex-sqrt vo, 425
m-complex-subtract vo, 195, 195
complex-tan vo, 426
Complex-type, 121, 122, 124, 191–194, 247, 273, 274, 293, 295, 301, 661
COMPLEX-ZERO-DIVISION ex, 195
ComplexMath lm, 415, 423, 428
complexmath-exception vo, 424, 426, 428
complexmath-exception-string vo, 428, 428
complexmath-initialization vo, 429
Complexmathexception vt, 428
compliance, 14
component type cs, 106, 106, 644, 644
component-type-of vf, 124, 184, 226, 233, 234, 263–265, 269, 297, 325, 383, 384
Component-typed vt, 310
computer system, 4, 4, 5, 10, 11, 14, 116, 279, 665, 668
Concat vo, 574
Concat lp, 573, 574, 574
concrete string, 123, 149, 205, 250, 279, 571, 579, 662, 663, 665, 670, 671
CondClaim vo, 561
CondClaim lp, 560, 561, 561
conforming implementation, xv, 1, 5, 10–15, 52, 408, 665, 670
conj vo, 424
conj lp, 424, 424
connect vo, 552, 557
consistent-imports vf, 28, 32, 46, 81
consistent-imports-in-definitions vf, 30, 45
constant declaration cs, 54, 64, 67, 640, 641, 641
constant expression, 7, 9, 23, 50, 67, 68, 70, 96, 111, 112, 159, 161, 165, 235, 238, 240, 253–256, 273, 278, 279, 283, 287, 302, 359, 365, 367, 662
constant expression cs, 67, 69, 95, 112, 161, 165, 234, 253, 641, 642, 644, 646, 647, 649, 650
constant literal cs, 189, 247, 648, 650
is-constant-actual-parameter vf, 255, 255
is-constant-array-constructor vf, 256, 256
is-constant-array-definition vf, 256
Constant-declaration vt, 67, 653
d-constant-declaration vf, 55, 64, 67
m-constant-declaration vo, 56, 66, 68
wf-constant-declaration vf, 56, 65, 68
is-constant-dereferenced-value vf, 254, 255
is-constant-element vf, 256, 256, 256
is-constant-entire-value vf, 254, 254
is-constant-expression vf, 51, 68, 70, 157, 166, 235, 240, 254, 254, 255–257, 284, 288, 365
is-constant-function-call vf, 254, 255
is-constant-function-designator vf, 255, 255
is-constant-indexed-value vf, 254, 254
is-constant-infix-expression vf, 254, 254
is-constant-interval vf, 97, 257, 257
Constant-literal vt, 247, 653
m-constant-literal vo, 190, 248
t-constant-literal vf, 190, 248
wf-constant-literal vf, 190, 248
is-constant-member vf, 256, 256
is-constant-prefix-expression vf, 254, 254
is-constant-record-constructor vf, 256, 256

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

is-constant-record-definition vf, 256, 256
is-constant-repeated-element vf, 256, 256
is-constant-selected-value vf, 254, 255
is-constant-set-constructor vf, 256, 256
is-constant-set-definition vf, 112, 161, 256, 256, 256
is-constant-singleton vf, 257, 257
is-constant-standard-function vf, 255, 255
is-constant-system-function vf, 255, 255
Constant-value vt, 308
is-constant-value-constructor vf, 254, 255
is-constant-value-designator vf, 254, 254, 254, 255
is-constant vf, 224, 254, 313
Constants vt, 308
wf-constants-in-expression vf, 191, 192
construct-array-value vf, 234, 234
construct-dcl-storage vo, 66, 66
construct-def-storage vo, 56, 56
construct-fields-list-value vo, 239, 241, 241
construct-fields-value vo, 241, 241
construct-fixed-fields-value vo, 241, 241
construct-function-procedure-struc vf, 61, 61
construct-local-environment vf, 29, 317
construct-module-environment vf, 29, 317
construct-parameter-typed vf, 59, 59
construct-parameter-typed-list vf, 59, 59, 61
construct-proper-procedure-struc vf, 59, 59
construct-variant-fields-value vo, 241, 241
construct-variant-value vo, 241, 241
constructor, 230–233, 236–238, 240–244, 250, 254, 256, 279, 564, 566, 662, 669
  repetition factor, 231, 233–235, 256
CONSTRUCTOR-RANGE ex, 236
contents vo, 225, 335, 452, 453
Continuation vt, 308
continuation, 307, 611, 612
continue-reading vf, 452, 452
control character, 464
control variable, 80, 142, 165–167, 169, 170, 670, 671
control variable identifier cs, 165, 165, 647, 647
control-variables vf, 81, 170
wf-control-variables vf, 142, 170
controls vv, 413
Conv-result vt, 584
convenum vf, 584, 585, 589, 595, 596, 603, 606
conversion
  of strings, *see* string conversion
  safe conversion, *see* VAL
  unsafe conversion, *see* CAST
CONVERSIONERROR ex, 306
converted-result vo, 282, 292, 294, 297, 298, 301, 304, 306, 306
convresults vv, 606
ConvTypes lm, 581, 592
Coroutine vt, 334
coroutine, xv, 12, 13, 25, 40, 41, 75, 78, 132, 135, 140, 141, 217, 218, 265, 334, 340, 341, 347, 368–372, 374–380, 384, 386, 387, 390, 394, 395, 399–401, 410, 411, 420, 428, 469, 501, 535, 544, 545, 548, 551, 553, 555, 562, 597, 604, 624, 661, 662, 665, 666, 668, 673
  workspace, 13, 75, 78, 265, 374, 375, 399–401, 548, 555
coroutine type, 217, 369, 370, 372, 374, 376–379, 661
coroutine-continuation vo, 375, 375
Coroutine-env vt, 344
coroutine-locs vo, 345, 346
Coroutine-storage vt, 332
Coroutine-store vt, 332
COROUTINES sm, 50, 217, 347, 368–370, 376, 399–401, 662, 665, 666
COROUTINES-function-designator vt, 376, 652
m-COROUTINES-function-designator vo, 348, 377
t-COROUTINES-function-designator vf, 348, 377
wf-COROUTINES-function-designator vf, 348, 377
COROUTINES-procedure-designator vt, 370, 652
m-COROUTINES-procedure-designator vo, 347, 370
wf-COROUTINES-procedure-designator vf, 347, 370
is-correct-order vf, 37, 37
cos vo, 418
cos lp, 417, 418, 418, 424, 426, 426
Create vo, 549
Create lp, 548, 548, 555, 560, 560
createsem vo, 560, 560
CURRENT sp, 369, 370, 376, 379
current-coroutine vo, 339–344, 346, 372, 376, 379
current-exception-handler vf, 33, 51, 136, 144, 145, 149, 153, 155, 160, 167, 183, 185, 186, 188, 193, 219, 224, 226–228, 230, 236, 244, 282, 284, 285, 287, 288, 291, 293, 295, 296, 302, 303, 306, 318, 337, 354, 358, 360, 366, 370, 377, 382, 385, 394
current-level vf, 59, 61, 317
CurrentFlags vo, 469
CurrentFlags lp, 469, 469
m-CURRENT vo, 377, 379
currentMode lp, 410, 410
CurrentNumber sp, 381, 384, 385, 401
m-CurrentNumber vo, 385, 386, 401, 469, 535, 553
t-CurrentNumber vf, 385, 385
wf-CurrentNumber vf, 385, 385
CurrentPos vo, 502
CurrentPos lp, 502, 502
t-CURRENT vf, 377, 379
wf-CURRENT vf, 377, 379

DateTime vt, 610
Day vt, 609
DEALLOCATE vo, 401
DEALLOCATE lp, 283, 284, 400, 400, 400, 401
deallocate-abstract-storage vo, 337, 438, 452, 453
deallocate-declarations vo, 127, 128, 128, 130, 131
deallocate-heap-storage vo, 401, 403
deallocate-locs vf, 337, 338, 338, 403
deallocate-module-env vo, 337, 338

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

deallocate-modules vo, 337, 337, 338
deallocate-parameters vo, 75, 76
deallocate-program-storage vo, 337, 453, 454
deallocate-storage vo, 76, 128, 337
deallocate-storage-of-environment vo, 337, 337, 338
deallocate-variables vo, 337, 338
DEC pi, 170, 259, 280, 281, 281, 281–283
m-dec-designator vo, 281, 282
wf-dec-designator vf, 281, 281
wf-dec-inc-value vf, 282, 282, 286
wf-dec-inc-var vf, 282, 282, 286
decimal number cs, 20
decimal-point-character vv, 450
Declaration vt, 64, 653
declaration cs, 63, 63, 641, 641
wf-declaration-bodies vf, 28, 32, 66, 67, 74, 78, 81
wf-declaration-body vf, 66, 67, 67
d-declaration vf, 64, 64, 65, 72
m-declaration vo, 66, 66
Declarations vt, 64, 653
declarations cs, 63, 126, 129, 131, 641, 644, 645
d-declarations vf, 35, 64, 64, 65–67, 80, 81, 126, 129, 132
m-declarations vo, 66, 127, 128, 130–132
wf-declarations vf, 28, 32, 65, 74, 77, 81
wf-declaration vf, 65, 65
declare-before-use, 6, 6, 11–13, 55, 64, 662
declare-before-use-in-declarations, 6, 6, 11, 13, 55, 64, 662
decrease-count vo, 561, 562, 563
Def-mods vt, 308
def-nameop vo, 542
default-dens vv, 541
default-figs vv, 590
default-ops vv, 541
Definition vt, 54, 653
definition cs, 54, 54, 640, 640
definition module cs, 23, 29, 639, 639
definition-environment vf, 35, 35
Definition-module vt, 29, 653
d-definition-module vf, 26, 29
m-definition-module vo, 26, 30
Definition-modules vt, 24, 653
wf-definition-module vf, 24, 29
d-definition vf, 54, 55, 55
m-definition vo, 56, 56
Definitions vt, 54, 653
definitions cs, 29, 54, 639, 640
d-definitions vf, 29, 54, 54, 55, 56
m-definitions vo, 30, 56
wf-definitions vf, 30, 55
wf-definition vf, 55, 55
del vf, 117, 119, 265, 615
Delete vo, 570
Delete lp, 569, 569
delimited token cs, 16
Denoted-basic-type vt, 308
Denoted-type vt, 308
dependent-modules vf, 37
dereferenced designator cs, 182, 187, 647, 647
dereferenced value cs, 222, 228, 648, 649
Dereferenced-designator vt, 187, 653
m-dereferenced-designator vo, 182, 188
t-dereferenced-designator vf, 182, 187
wf-dereferenced-designator vf, 182, 187
Dereferenced-value vt, 228, 653
m-dereferenced-value vo, 223, 228
t-dereferenced-value vf, 223, 228
wf-dereferenced-value vf, 223, 228
dereferencing operator cs, 20, 187, 228, 647, 649, 650
Destroy vo, 560
Destroy lp, 560, 560
destroysem vo, 560, 560
DETACH sp, 369–372
Detach vo, 552
Detach lp, 548, 552, 552
detach vo, 343, 372
m-DETACH vo, 371, 372
wf-DETACH vf, 370, 371
Detection-not-required vt, 397
Detection-required vt, 396
Dev-id vt, 536
device module, 12, 430, 431, 461, 464, 466–468, 470, 473, 531, 534, 535
device-op vo, 451–454, 464–468, 539
Device-table vt, 536
DeviceError vo, 470
DeviceError lp, 469, 469, 470
DeviceTablePtrValue vo, 534
DeviceTablePtrValue lp, 531, 534, 534
DIFADR sp, 350, 353, 354, 357, 358
m-DIFADR vo, 354, 358
t-DIFADR vf, 354, 357
wf-DIFADR vf, 354, 357
diff vf, 578, 578
digit cs, 20
digit-value vf, 606, 606
digits vv, 275
digits-order vv, 275
is-digits vf, 584, 585, 589, 605
directive body cs, 21
directly-imports vf, 28, 30, 32, 37, 39, 42, 81
disconnect vo, 552, 557
is-disj vf, 112, 161, 275, 307, 319, 555, 614
is-disjoint-case-labels vf, 160, 161
is-disjoint-declarations vf, 65, 65
is-disjoint-definitions vf, 55, 55
is-disjoint-variant-labels vf, 111, 114
DISPOSE pi, 259, 280, 283, 283, 284, 399
m-dispose-designator vo, 281, 284
wf-dispose-designator vf, 281, 283
is-distinct-identifiers vf, 108, 113
div operator cs, 19, 191, 648, 650
division operator cs, 19, 191, 648, 650

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

do-look vo, 436, 438, 451, 452
do-skip vo, 436, 438, 452, 452
do-skip-rest vo, 438, 438
do-skiplook vo, 438, 452
double quote cs, 20
dynamic storage, 401

E vo, 602
echoFlag vv, 474
elaboration, 7, 28, 30, 32, 33, 37, 56, 57, 66, 68, 69, 71, 75, 78, 79, 81, 114, 132, 137, 139, 265, 307, 663
Element vt, 236, 653
elementary type, 71, 91, 92, 97–99, 101, 102, 115, 149, 206, 224, 270, 271, 278, 280, 321, 412, 660, 663, 670
is-elementary-type vf, 321
Elementary-value vt, 333
Elementary-variable vt, 312
elementary-variable-assignment vo, 150, 150
Elements vt, 238, 653
wf-element vf, 235, 236, 240
ellipsis cs, 17, 650
else expression cs, 613
elseif expression cs, 613
a-elsepart vf, 118, 119, 119
emit vf, 479, 480, 491–493, 505–507, 516, 517, 521, 524, 526, 529, 530
empty Module, 126
empty statement cs, 146, 148, 645, 645
Empty-statement vt, 148, 653
m-empty-statement vo, 147, 148
wf-empty-statement vf, 146, 148
end of line, 438
endOfInput vv, 449
endOfLine vv, 449
EndPos vo, 502
EndPos lp, 502, 502, 503
engstr vf, 446, 590, 590, 604
entire designator, 182, 183, 302, 303, 670
entire designator cs, 182, 183, 647, 647
entire value cs, 222, 223, 648, 648
Entire-designator vt, 183, 653
m-entire-designator vo, 167, 182, 183
t-entire-designator vf, 166, 182, 183, 295, 382
wf-entire-designator vf, 182, 183
Entire-value vt, 223, 653
m-entire-value vo, 223, 224
t-entire-value vf, 223, 224
wf-entire-value vf, 223, 224
Entity vt, 319
enumeration type, 28, 41, 43, 44, 46–48, 69, 70, 84, 91, 94, 95, 107, 108, 110, 304, 322, 325, 380, 392, 393, 402, 448, 470, 472, 554, 564, 575, 581, 662, 664, 667
enumeration type cs, 93, 94, 642, 642
is-enumeration-constant vf, 65, 323
Enumeration-structure vt, 309
a-enumeration-structure vf, 114, 115
Enumeration-type vt, 94, 653
d-enumeration-type vf, 94, 94
is-enumeration-type vf, 322, 323, 323
wf-enumeration-type vf, 94, 95
Enumeration-value vt, 333
Environment vt, 307
Equal vo, 576
Equal lp, 576, 576
equal vv, 580
equals cs, 17, 650
equals operator cs, 19, 190, 648, 650
err-ops vv, 540
err-props vv, 540
err-table vv, 540
ErrChan vo, 434
ErrChan lp, 434, 434
errchan vo, 434, 538
error, 6, 12
evaluate-array-definition vo, 232, 234, 236
evaluate-constant-expression vf, 67, 157, 166, 192, 234, 235, 239, 240, 257, 283, 367
evaluate-constant-interval vf, 96, 97, 257
evaluate-constant-set-definition vf, 112, 114, 161, 257
evaluate-element vo, 235, 236, 241
evaluate-interrupt-protection vo, 28, 32, 51, 81
evaluate-record-definition vo, 236, 237, 239
evaluate-repeated-element vo, 234, 235
evaluate-repeated-elements vo, 234, 234
evaluate-set-definition vo, 236, 242, 244
evaluate-to-ordinal vo, 247, 253
EVENT-occurs vo, 557
Event-Source vt, 555
Example, 49, 69, 98, 106, 111, 142, 160, 184, 205, 212, 213, 225, 232, 235, 237, 242, 262, 263, 266, 268, 295, 299, 300, 305, 388, 568–579, 582, 594
exception, 2, 6, 6, 6, 12, 13, 25, 40, 51, 75, 78, 126–131, 133–141, 143–145, 147, 149, 155, 156, 160, 183, 185, 186, 188, 194–196, 198–200, 212, 220, 224, 226–228, 236, 244, 265, 282, 285–287, 291, 295, 303, 305, 334, 337, 355, 356, 358, 359, 364, 372, 374, 375, 378, 380–387, 392–397, 400, 402, 403, 409–411, 417–421, 424, 426, 428, 429, 431, 464–467, 469, 470, 478, 485, 489, 490, 502–504, 509, 510, 514, 523, 531, 534, 535, 541, 549, 551, 553–555, 560, 562, 563, 595–598, 603–605, 661, 663, 664, 669
exception handling context, 141, 144, 145, 334, 342, 381, 396–398, 411
exception raising, 6
exception-actions vo, 26, 341, 375
Exception-context vt, 334
Exception-context-stack vt, 334
Exception-handler vt, 344

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Exception-source vt, 334
Exception-sources vt, 334
Exception-value vt, 334
exception-value vv, 398
exceptional part cs, 142, 142, 645, 645
exceptional termination, 6, 13, 40, 127, 129, 133–136, 141, 155, 375, 383, 395, 548, 663
is-exceptional vf, 128, 131, 134, 136, 138, 140, 342
EXCEPTIONS sm, 141, 347, 380, 381, 384, 387, 397, 536
EXCEPTIONS-function-designator vt, 384, 653
m-EXCEPTIONS-function-designator vo, 348, 385
t-EXCEPTIONS-function-designator vf, 348, 385
wf-EXCEPTIONS-function-designator vf, 348, 385
EXCEPTIONS-procedure-designator vt, 381, 653
m-EXCEPTIONS-procedure-designator vo, 347, 382
wf-EXCEPTIONS-procedure-designator vf, 347, 382
EXCL pi, 259, 280, 284, 284, 285
m-excl-designator vo, 281, 285
wf-excl-designator vf, 281, 285
EXCLNOTBASE ex, 285
execute vo, 75, 79, 342, 345
execute-dynamic-module-final vo, 139, 139
execute-dynamic-module-final-h vo, 139, 140
execute-dynamic-module-final-hp vo, 139, 140
execute-dynamic-module-final-p vo, 139, 139
execute-dynamic-module-finalization vo, 139, 139
execute-dynamic-module-init vo, 137, 137
execute-dynamic-module-init-h vo, 137, 138
execute-dynamic-module-init-hp vo, 137, 138
execute-dynamic-module-init-p vo, 137, 137
execute-dynamic-module-initialization vo, 137, 137
execute-else-statements vo, 158, 159
execute-except vo, 143, 144
execute-for-statement vo, 167, 167
execute-function-except vo, 145, 145
execute-function-normal vo, 144, 144
execute-function-normal-except vo, 145, 145
execute-non-result-body vo, 127, 133, 135, 137, 139, 143
execute-non-result-body-h vo, 128, 134, 136, 138, 140, 143
execute-normal vo, 143, 143
execute-normal-except vo, 138, 140, 143, 143, 144
execute-program vo, 25, 26
execute-result-body vo, 130, 144
execute-result-body-h vo, 130, 131, 145
execute-sourced-module-final vo, 135, 135
execute-sourced-module-final-h vo, 135, 136
execute-sourced-module-final-hp vo, 135, 136
execute-sourced-module-final-p vo, 135, 135
execute-static-module-init vo, 133, 133
execute-static-module-init-h vo, 133, 134
execute-static-module-init-hp vo, 133, 134
execute-static-module-init-p vo, 133, 133
execute-static-module-initialization vo, 28, 33, 132, 134
Execution-order vt, 39
execution-order vf, 26, 39
is-existing-file vf, 477, 487, 489, 500, 501, 529
exists-loc vo, 188, 228, 335
Exit vt, 313
exit statement cs, 146, 164, 645, 647
Exit-statement vt, 164, 653
m-exit-statement vo, 147, 164
wf-exit-statements vf, 142, 177
wf-exit-statement vf, 146, 164
exp vo, 417
exp lp, 417, 417, 420, 425, 425
exponent lp, 409, 409, 419, 425, 445–447, 589–591
exponent-character vv, 450
export list, 23–25, 29, 46, 48, 49, 79, 80
export list cs, 46, 79, 640, 642
Export-list vt, 46, 654
d-export-list vf, 46, 80, 82
wf-export-list vf, 46, 81
exported-from-protected vf, 51, 80, 318
exported-identifiers vf, 48, 80
is-exported vf, 45, 45
Expression vt, 189, 654
expression cs, 148, 154, 158, 189, 189, 190, 235, 253, 258, 645, 646, 648, 648–650
is-expression-compatible vf, 97, 114, 121, 161, 166, 193, 196, 199, 207, 208, 243, 246
Expression-structure vt, 311
Expression-typed vt, 311
m-expression vo, 51, 68, 71, 149, 155, 158, 160, 162, 163, 167, 185, 190, 191, 203, 219, 226, 235, 236, 241, 246, 253, 257, 260
t-expression vf, 51, 67, 68, 70, 149, 155, 157, 158, 160, 162, 163, 166, 176, 184, 190, 191, 192, 219, 226, 235, 236, 246, 247, 260, 283, 285, 287, 290–298, 300–303, 305, 355–357, 362–364, 367, 371–374, 376–378, 383, 385, 386
wf-expression vf, 68, 70, 149, 155, 158, 160, 162, 163, 166, 184, 190, 191, 219, 226, 235, 236, 246, 247, 260, 284, 288
External-IOLib-state vt, 536
External-program-state vt, 331
External-ProgramArgs-state vt, 527
Extract vo, 569
Extract lp, 568, 569

F vo, 602
f vf, 611, 612
factor cs, 189, 189, 648, 648
factor operator cs, 189, 191, 648, 648
FALSE pi, 274, 274
field identifier cs, 185, 185, 227, 647, 647, 648
field list cs, 107, 107, 110, 112, 644, 644
field type cs, 109, 109, 644, 644
Field-list vt, 108, 654
a-field-list vf, 117, 117, 119
wf-field-list vf, 108, 108, 111, 112
field-types-of-record vf, 157, 173, 186, 227, 326
Fields vt, 108, 654
fields cs, 107, 107, 644, 644

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Fields-list-structure vt, 310
fields-of-fields vf, 327, 327
fields-of-fields-list vf, 326, 327, 327
fields-of-fixed-fields vf, 327, 327
fields-of-variant vf, 327, 327
fields-of-variant-fields vf, 327, 327
fields-of-variants vf, 327, 327
Fields-structure vt, 310
a-fields-structure vf, 117, 117
d-fields vf, 108, 108
wf-fields vf, 108, 108
File vt, 536
file
opening, 431, 483, 485–490, 499–504, 531
random access, 431, 472, 497, 500–504
sequential access, 431, 472, 483, 485, 489, 490, 495
File-id vt, 536
fileExists vv, 475
Filepos vt, 499
filepos-to-pos vf, 503, 504, 504
FileRnd vt, 510
FileSeq vt, 495
FileStream vt, 482
final value cs, 165, 165, 647, 647
finalization body cs, 131, 131, 645, 645
finalize-a-dynamic-module vo, 139, 139
finalize-a-sourced-module vo, 135, 135
finalize-a-static-module vo, 41, 135
finalize-an-abstract-module vo, 135, 136
finalize-dynamic-modules vo, 127, 128, 130, 138, 139, 140
finalize-modules vo, 41, 41, 41
find vf, 576, 577, 577, 577
FindDiff vo, 578
`FindDiff` lp, 574, 578, 578
FindNext vo, 576
`FindNext` lp, 576, 576
FindPrev vo, 577
`FindPrev` lp, 574, 577, 577
firstocc vf, 168, 210, 615
fixed fields cs, 107, 109, 644, 644
Fixed-fields vt, 109, 654
Fixed-fields-structure vt, 310
a-fixed-fields-structure vf, 117, 118
d-fixed-fields vf, 108, 109
wf-fixed-fields vf, 109, 109
fixed-point-value vf, 589–591, 606
fixedstr vf, 447, 591, 591, 604
`FLOAT` pi, 288, 293, 293, 294
m-float-designator vo, 290, 294
t-float-designator vf, 289, 294
wf-float-designator vf, 289, 294
floatstr vf, 446, 590, 590, 591, 603
Flush vo, 468
`Flush` lp, 468, 468, 482, 495, 508, 518, 541
for statement cs, 146, 165, 645, 647
For-statement vt, 165, 654
m-for-statement vo, 147, 167
wf-for-statement vf, 146, 165
formal parameter, 9, 34, 58–63, 68, 72–78, 91, 101, 103–105, 114, 121–123, 152, 153, 168, 170, 182, 183, 213, 222–224, 229–231, 258–268, 283, 287, 294, 307, 326, 352, 660, 661, 667
formal parameter cs, 58, 61, 640, 641
formal parameter list cs, 58, 58, 640, 640
formal parameter type cs, 102, 102, 643, 643
formal parameter type list cs, 101, 102, 643, 643
formal parameters cs, 58, 58, 60, 640, 640, 641
formal type, 34, 59–63, 72, 74, 77, 88, 101–105, 122, 213, 261–268, 294, 328, 661, 667, 671
formal type cs, 62, 103, 104, 104, 641, 643, 643
Formal-parameter vt, 61, 654
Formal-parameter-list vt, 58, 654
Formal-parameter-type vt, 102, 654
Formal-parameter-type-list vt, 102, 654
d-formal-parameter-type-list vf, 101–103, 103
Formal-parameter-typed vt, 310
d-formal-parameter-type vf, 103, 103
wf-formal-parameter-type vf, 101, 102, 103
d-formal-parameter vf, 61, 74, 77
is-formal-parameters-distinct vf, 59, 60, 60
Formal-parameters-typed vt, 310
wf-formal-parameter vf, 59, 60, 62
Formal-type vt, 104, 654
d-formal-type vf, 62, 63, 103, 104, 105, 267, 268
wf-formal-type vf, 62, 63, 103, 104, 105
format effector cs, 22
FormatCard vo, 596
`FormatCard` lp, 596, 596
FormatInt vo, 595
`FormatInt` lp, 594, 594
FormatReal vo, 603
`FormatReal` lp, 602, 602
wf-forward-declarations vf, 65, 72
is-forward-procedure-declaration vf, 65, 72, 72
Fraction vt, 610
`fraction` lp, 409, 409
`fractpart` lp, 410, 410
free-exit-statements-of-case-statement vf, 178, 178
free-exit-statements-of-for-statement vf, 178, 179
free-exit-statements-of-if-statement vf, 178, 178
free-exit-statements-of-repeat-statement vf, 178, 179
free-exit-statements-of-statement vf, 177, 177
free-exit-statements-of-statement-seq vf, 177, 177–179
free-exit-statements-of-while-statement vf, 178, 179
free-exit-statements-of-with-statement vf, 178, 178
front vf, 55, 65, 67, 234, 529, 614, 615
full identifier cs, 16
function body cs, 129, 129, 644, 644
function call cs, 189, 229, 648, 649
function designator cs, 229, 229, 649, 649
function parameter, *see* actual parameter, formal parameter

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

function procedure, 57, 60, 61, 68, 70, 71, 76–79, 91, 100, 101, 122, 124, 126, 129–131, 140, 144, 153–155, 169, 213, 229, 230, 254, 258, 260, 261, 265–267, 270, 353, 354, 358, 359, 389, 393, 531, 566–573, 637, 638, 661, 667, 669, 670
function procedure block cs, 76, 129, 642, 644
function procedure declaration cs, 72, 76, 641, 642
function procedure heading cs, 58, 60, 76, 640, 641, 642
function procedure type cs, 100, 101, 643, 643
function result type cs, 60, 60, 101, 641, 641, 643
function return statement cs, 153, 154, 645, 646
Function-call vt, 229, 654
m-function-call vo, 190, 230
t-function-call vf, 190, 230
wf-function-call vf, 190, 229
is-function-declarations-distinct vf, 77, 79
Function-designator vt, 229, 654
Function-procedure-block vt, 129, 654
m-function-procedure-block-h vo, 79, 130
m-function-procedure-block-hp vo, 79, 131
m-function-procedure-block-p vo, 79, 130
m-function-procedure-block vo, 79, 130
wf-function-procedure-body vf, 78, 129
Function-procedure-cont vt, 344
Function-procedure-declaration vt, 77, 654
d-function-procedure-declaration vf, 72, 77
m-function-procedure-declaration vo, 73, 78
wf-function-procedure-declaration vf, 72, 77
Function-procedure-heading vt, 60, 654
d-function-procedure-heading-declaration vf, 58, 60, 77
d-function-procedure-heading vf, 77, 78
wf-function-procedure-heading vf, 58, 60, 77
Function-procedure-structure vt, 310
a-function-procedure-structure vf, 116, 116
Function-procedure-type vt, 102, 654
Function-procedure-typed vt, 311
d-function-procedure-type vf, 100, 102
is-function-procedure-type vf, 321, 322
wf-function-procedure-type vf, 100, 102
wf-function-procedure vf, 73, 78
Function-return-statement vt, 154, 654
m-function-return-statement vo, 154, 155
wf-function-return-statements vf, 144, 176
wf-function-return-statement vf, 153, 155

generate-function-procedure vf, 78, 78
generate-loc vf, 115, 116, 337, 338
generate-proper-procedure vf, 75, 75
generate-type-name vf, 57, 95–97, 99–102, 107, 108, 269, 320
get-complex-result vo, 194, 194, 221, 293, 424–428
get-coroutine-storage vo, 337, 338, 339
get-device-id vo, 477, 489, 501, 514, 534, 535, 540
get-error vo, 470, 539
get-heap-storage vo, 403, 403
get-locs vf, 338, 338
get-module-environment vf, 317, 337
get-props vo, 469, 482, 495, 503, 508, 518–520, 539
get-protection vo, 344, 344, 373, 375, 379
get-real-result vo, 196, 197, 221, 291, 296, 302, 306, 417–419, 424
get-whole-result vo, 199, 199, 221, 291, 295, 303, 306, 358, 360, 366, 420, 595, 597, 603, 604
getchars vo, 453, 453
GetClock vo, 609
GetClock lp, 609, 609
getlocs vo, 453, 453, 454
GetMessage sp, 381, 383, 384, 397
m-GetMessage vo, 382, 384
wf-GetMessage vf, 382, 384
GetName lp, 467, 467, 481, 494, 508, 518, 526, 541
Getname vo, 468
GMT, *see* Greenwich Mean Time
greater vv, 580
greater than operator cs, 19, 190, 648, 650
greater than or equal operator cs, 20, 190, 648, 650
Greenwich Mean Time, 608
guarded elseif expressions cs, 613
guarded expression cs, 612
guarded expression list cs, 612
guarded if expression cs, 613
guarded if expression list cs, 613
guarded statement, 158, 159
guarded statements cs, 157, 158, 646, 646
Guarded-statement vt, 158, 654

HALT pi, 40, 135, 280, 285, 285, 391
m-halt-designator vo, 281, 285
wf-halt-designator vf, 281, 285
is-handled vf, 75, 76, 79, 133, 135, 137, 139
HANDLER sp, 369, 376–378
Handler vo, 552
Handler vt, 335
Handler lp, 548, 552, 552
m-HANDLER vo, 377, 378
Handlers vt, 335
t-HANDLER vf, 377, 377
wf-HANDLER vf, 377, 377
hardDeviceError vv, 470
HasHalted sp, 389, 391
m-HasHalted vo, 390, 391
t-HasHalted vf, 390, 391
wf-HasHalted vf, 390, 391
HEAP-STORAGE-POOL vv, 402
hex digit cs, 20
hexadecimal number cs, 20
HIGH pi, 266, 267, 272, 288, 294, 294, 294, 295
wf-high-argument vf, 294, 295, 295
m-high-designator vo, 290, 295
t-high-designator vf, 289, 294
wf-high-designator vf, 289, 294
host type, 95, 96, 121, 165, 224, 226–229, 272, 281, 285–287, 290, 298, 299, 304, 305, 325, 361,

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

664, 669
host-type-of vf, 123, 124, 166, 211, 224, 226–228, 230, 282, 283, 285, 287, 300, 305, 306, 325, 325, 328, 362–364, 367
Hour vt, 610

I/O Channel, 12, 430, 431, 433, 434, 436–438, 440, 441, 444, 448–451, 455–457, 459–461, 464, 466–470, 472, 473, 475, 476, 478, 483, 485–490, 497, 499, 500, 502–504, 512–515, 522, 523, 527, 531, 535, 541, 542
Identifier vt, 52, 654
identifier cs, 16, 27, 53, 56, 67–69, 73, 94, 110, 165, 185, 639–641, 644, 647, 650
identifier list cs, 43, 44, 47, 48, 62, 94, 94, 109, 639–642, 642, 644
identifier-of-a-formal-parameter vf, 76
identifiers-declared vf, 82, 83
identifiers-declared-in vf, 46, 76, 79, 82, 128, 171
identifiers-defined vf, 82, 82
identifiers-defined-in vf, 29, 30, 45, 82
identifiers-defined-in-a-variant vf, 84, 84
identifiers-defined-in-fields vf, 83, 84
identifiers-defined-in-fields-list vf, 83, 83, 84
identifiers-defined-in-type vf, 82, 83, 83, 83, 84
identifiers-defined-in-variants vf, 84, 84
identifiers-explicitly-declared vf, 83, 83
identifiers-explicitly-defined vf, 82, 82
identifiers-implicitly-declared vf, 83, 83
identifiers-implicitly-defined vf, 82, 82
identifiers-of-a-variant vf, 113, 113
identifiers-of-field-list vf, 113, 113, 113
identifiers-of-fields vf, 113, 113
identifiers-of-fixed-fields vf, 113, 113
identifiers-of-formal-parameter-list vf, 60, 76, 76, 79, 171
identifiers-of-variant-fields vf, 113, 113
identifiers-of-variants vf, 113, 113, 113
identifiers-used vf, 84, 84
identifiers-used-in vf, 84
identifiers-used-in-a-parameter vf, 88, 88
identifiers-used-in-a-variant vf, 88, 88
identifiers-used-in-actual-parameters vf, 85, 85
identifiers-used-in-an-actual-parameter vf, 85, 85
identifiers-used-in-array-constructor vf, 86, 86
identifiers-used-in-array-formal-type vf, 88, 88
identifiers-used-in-element vf, 86, 86
identifiers-used-in-expression vf, 68, 84, 84, 85–88
identifiers-used-in-fields vf, 87, 87
identifiers-used-in-fields-list vf, 87, 87, 88
identifiers-used-in-formal-type vf, 88
identifiers-used-in-function-call vf, 84, 85
identifiers-used-in-infix-expression vf, 84, 85
identifiers-used-in-member vf, 87, 87
identifiers-used-in-parameters vf, 60, 88, 89
identifiers-used-in-prefix-expression vf, 84, 85
identifiers-used-in-record-constructor vf, 86, 86
identifiers-used-in-set-constructor vf, 86, 86
identifiers-used-in-type vf, 69, 70, 84, 87, 87, 88, 89
identifiers-used-in-value-constructor vf, 84, 86
identifiers-used-in-value-designator vf, 84, 85, 85
identifiers-used-in-variable-designator vf, 85, 86, 86
identifiers-used-in-variants vf, 87, 88
identity operator cs, 19
if else part cs, 157, 158, 646, 646
if expression cs, 612, 613
if statement cs, 146, 157, 645, 646
If-statement vt, 158, 654
m-if-statement vo, 147, 158, 158
wf-if-statement vf, 146, 158
IM pi, 17, 260, 288, 295, 295, 296
m-im-designator vo, 290, 296
t-im-designator vf, 289, 295
wf-im-designator vf, 289, 295
im-max-cc-value vv, 278
im-max-complex-value vv, 278
im-max-long-complex-value vv, 278
im-min-cc-value vv, 278
im-min-complex-value vv, 278
im-min-long-complex-value vv, 278
imp-nameop vo, 542
implementation, 4
implementation module cs, 23, 31, 639, 639
implementation-capitalizations vv, 292
implementation-characters vv, 275
implementation-defined, 5, 6, 9–13, 15, 21, 22, 25, 27, 34, 50, 92, 165, 194, 197, 216, 217, 249, 250, 271–275, 280, 291, 293, 296, 302, 305, 333, 355, 356, 358, 359, 361, 364, 368–371, 375, 383, 387, 395, 397, 402, 407, 408, 411–413, 417, 419–421, 423, 425, 427, 428, 431–433, 445, 464, 470, 471, 499, 503, 509, 510, 523, 527, 545, 548, 554, 562, 575, 589, 597, 598, 604, 605, 608, 663, 665, 667, 668, 670, 671
implementation-defined approximation, 250, 293, 296, 302, 305, 417, 419, 423, 425, 427
implementation-defined graphic cs, 22
implementation-defined-exception-string vo, 396, 397, 397
Implementation-defined-type vt, 309
implementation-dependent, 2, 5, 5, 5, 6, 9, 13, 14, 149, 153, 166, 185, 226, 230, 233, 238, 244, 259, 265, 331, 341, 364, 365, 375, 397, 401, 403, 409, 548, 549, 551, 552, 560, 609, 664, 665, 667
implementation-lower-cases vv, 275
implementation-lower-cases-order vv, 275
Implementation-module vt, 30, 655
m-implementation-module vo, 26, 31
Implementation-modules vt, 24, 655
wf-implementation-module vf, 24, 30
implementation-other-characters vv, 275
implementation-upper-cases vv, 275
implementation-upper-cases-order vv, 275

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

implementation-whitespace vv, 414
implementor, xv, 5, 5, 9, 11, 13, 15, 97, 272, 668
import
  circular dependency, 38
import list, 23, 25, 27–29, 31, 34, 36, 38, 41–43, 45, 46, 49, 79, 80, 132, 135, 283, 287, 665
import list cs, 41, 43, 639, 639
import lists cs, 27, 29, 31, 41, 79, 639, 639, 642
Import-list vt, 43, 655
d-import-list vf, 42, 43
Import-lists vt, 42, 655
d-import-lists vf, 28–30, 32, 35, 42, 80, 81
wf-import-lists vf, 28, 30, 32, 42, 81
wf-import-list vf, 42, 43
import-order vf, 39, 39
import-orders vf, 39, 39
imported-by-definition-module vf, 39, 39
imported-by-implementation-module vf, 39, 39
imported-by-sourced-implementation-module vf, 39, 39
imported-environment vf, 35, 35
is-imported-into-module vf, 170, 176
imported-module-name vf, 42, 42
imported-names vf, 42, 43
imports-of vf, 42, 45, 46, 176
in-heap vf, 401, 403
in-ops vv, 540
in-props vv, 540
in-range vf, 192, 192, 194, 197, 199
in-table vv, 540
INACTIVE-VARIANT ex, 183, 186, 227
INC pi, 170, 259, 280, 282, 283, 286, 286
m-inc-dec vo, 282, 282, 286
m-inc-designator vo, 281, 286
wf-inc-designator vf, 281, 286
InChan vo, 433
InChan lp, 433, 433
inchan vo, 433, 538
INCL pi, 259, 280, 286, 286, 286, 287
m-incl-designator vo, 281, 287
wf-incl-designator vf, 281, 287
INCLNOTBASE ex, 287
increase-count vo, 561, 563
index expression cs, 184, 184, 225, 647, 647, 648
index type, 91, 106, 107, 116, 123, 149, 184, 185, 226, 233, 264–267, 294, 295, 325, 326, 564, 566, 660, 665, 667
index type cs, 106, 106, 644, 644
INDEX-RANGE ex, 185, 226
index-type-of vf, 184, 226, 234, 326, 328
indexed designator cs, 182, 184, 647, 647
indexed value cs, 222, 225, 648, 648
Indexed-designator vt, 184, 655
m-indexed-designator vo, 182, 185
t-indexed-designator vf, 182, 184
wf-indexed-designator vf, 182, 184
Indexed-value vt, 225, 655
m-indexed-value vo, 223, 226
t-indexed-value vf, 223, 226
wf-indexed-value vf, 223, 226
inequality operator cs, 19, 190, 648, 650
Infix-expression vt, 191, 655
m-infix-expression vo, 190, 191
t-infix-expression vf, 190, 191
wf-infix-expression vf, 190, 191
Infix-operation vt, 192, 655
m-infix-operation vo, 191, 193
t-infix-operation vf, 191, 193
wf-infix-operation vf, 191, 192
init vo, 33, 33
initial value cs, 165, 165, 647, 647
initialization body cs, 131, 131, 645, 645
Initialization-actions vt, 334
initialize vf, 345
initialize vo, 345, 375
initialize-a-dynamic-module vo, 137, 137
initialize-a-sourced-implementation-module vo, 31, 33
initialize-a-static-module vo, 134, 134
initialize-an-abstract-implementation-module vo, 31, 33
initialize-an-implementation-module vo, 26, 31
initialize-dynamic-modules vo, 127, 128, 131, 136, 137, 138
initialize-modules vo, 26, 26
initialize-program-module vo, 26, 28
initialize-static-modules vo, 133, 134, 134, 134
initprot vv, 333
m-in vo, 211, 212
Insert vo, 571
Insert lp, 567, 570, 570
INT pi, 17, 288, 296, 296, 296
m-int-designator vo, 290, 297
t-int-designator vf, 289, 296
wf-int-designator vf, 289, 296
Int-type, 121, 122, 124, 191, 192, 198, 199, 245, 248, 249, 254, 271, 272, 281, 283, 286, 302, 303, 365
INTEGER pi, 5, 92, 272, 272, 296, 305, 350, 420, 544, 548, 582, 592, 661, 665, 669, 673
interactive, 511, 513, 593
interactiveFlag vv, 474
interrupt, 25, 27, 31, 32, 50, 80, 126, 217, 271, 280, 333, 368–379, 662, 665–668
interrupt protection cs, 27, 31, 50, 79, 639, 640, 642
interrupt-occurs vo, 373
Interrupt-protection vt, 50, 655
wf-interrupt-protection vf, 28, 32, 51, 81
Interrupt-source vt, 335
INTERRUPTIBLE pi, 17, 50, 280, 280
Interval vt, 245, 655
interval cs, 244, 245, 649, 649
m-interval vo, 245, 246, 257
t-interval vf, 97, 245, 246
wf-interval vf, 97, 245, 245
intpart lp, 410, 410
intstr vf, 441, 585, 585, 595

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

IntToStr vo, 584
IntToStr lp, 584, 584
invalid vv, 606
INVALID-ALIGNMENT ex, 365
INVALID-VALUE ex, 365
InvalidChan vo, 464
InvalidChan lp, 464, 464
IOChan lm, 431, 460, 461, 470, 478, 490, 504, 515, 523, 535, 536
iochan-initialization vo, 470
IOChan-WriteLn vo, 465
IOChannel-ReadResult vo, 469
IOConsts lm, 431, 448, 449, 460
IOException vo, 535
IOException lp, 535, 535, 536
IOLink lm, 431, 531
IOResult lm, 430, 431, 434, 449, 459, 460
IOResult-ReadResult vo, 450
IOTRANSFER sp, 369, 370, 372, 376, 662, 665–667
m-IOTRANSFER vo, 371, 372
wf-IOTRANSFER vf, 370, 372
isa-Char vf, 529, 529
isa-number vf, 440
isa-numeral vf, 440, 444, 451, 594, 596, 601, 602
isa-prefix vf, 530, 530
isa-real-number vf, 444
isa-sign vf, 451, 594, 601, 602, 605
isa-signed-number vf, 440
isa-space vf, 437, 450, 594, 596, 601
isa-token vf, 437, 437
IsArgPresent vo, 523
IsArgPresent lp, 523, 523, 523
IsATTACHED sp, 369, 376, 378
IsAttached vo, 552
IsAttached lp, 548, 552, 552
m-IsATTACHED vo, 377, 378
t-IsATTACHED vf, 377, 378
wf-IsATTACHED vf, 377, 378
IsChanException vo, 469
IsChanException lp, 469, 469
IsCMathException vo, 428
IsCMathException lp, 428, 428
IsControl vf, 413
IsControl lp, 413, 413
IsCurrentSource sp, 381, 384, 386
m-IsCurrentSource vo, 385, 386, 401, 420, 428, 469, 502, 535, 553, 562, 597, 604
t-IsCurrentSource vf, 385, 386
wf-IsCurrentSource vf, 385, 386
IsDevice vo, 535
IsDevice lp, 531, 535, 535
IsExceptionalExecution sp, 381, 384, 387
m-IsExceptionalExecution vo, 385, 387
t-IsExceptionalExecution vf, 385, 387
wf-IsExceptionalExecution vf, 385, 387
IsIOException vo, 535
IsIOException lp, 535, 535, 536
IsLetter vf, 413
IsLetter lp, 412, 413
IsLower vf, 413
IsLower lp, 413, 413
IsLowException lp, 411, 411
m-IsM2Exception vo, 394, 395
t-IsM2Exception vf, 394, 395
wf-IsM2Exception vf, 394, 395
IsNumeric vf, 412
IsNumeric lp, 412, 412
IsProcessesException vo, 553
IsProcessesException lp, 553, 553
IsRConvException vo, 604
IsRConvException lp, 604, 604
IsRMathException vo, 420
IsRMathException lp, 420, 420
IsRndFile vo, 501
IsRndFile lp, 501, 501
IsRndFileException vo, 501
IsRndFileException lp, 501, 501
IsSemaphoresException vo, 562
IsSemaphoresException lp, 562, 562
IsSeqFile vo, 489
IsSeqFile lp, 489, 489
IsStorageException vo, 401
IsStorageException lp, 401, 401
IsStreamFile vo, 477
IsStreamFile lp, 477, 477
IsTermFile vo, 514
IsTermFile lp, 514, 514
IsTerminating sp, 389, 390
m-IsTerminating vo, 390, 390
t-IsTerminating vf, 390, 390
wf-IsTerminating vf, 390, 390
IsUpper vf, 413
IsUpper lp, 413, 413
IsValidDateTime vo, 609
IsValidDateTime lp, 608, 608
IsWhiteSpace vf, 413
IsWhiteSpace lp, 413, 413
IsWholeConvException vo, 597
IsWholeConvException lp, 597, 597
iteration-down-sequence vf, 167, 167, 168
iteration-up-sequence vf, 167, 167
iteration-values vf, 167, 167

keyword cs, 17
kill vo, 549, 556

Language-exception vt, 334
Language-exceptions vt, 396
leap seconds, 607
is-leap-year vf, 610
left brace cs, 18, 650
left bracket cs, 18, 651
left parenthesis cs, 17, 651
is-legal-filename vf, 477, 487–489, 500, 501, 529, 529

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

LENGTH pi, 17, 288, 297, 297, 567
Length vo, 567
Length lp, 567, 567
m-length-designator vo, 290, 297
t-length-designator vf, 289, 297
wf-length-designator vf, 289, 297
LengthCard vo, 597
LengthCard lp, 582, 597, 597
LengthEngReal vo, 604
LengthEngReal lp, 603, 603, 604
LengthFixedReal vo, 604
LengthFixedReal lp, 604, 604
LengthFloatReal vo, 603
LengthFloatReal lp, 603, 603
LengthInt vo, 595
LengthInt lp, 595, 595
less vv, 580
less than operator cs, 19, 190, 648, 651
less than or equal operator cs, 19, 190, 648, 651
is-less-restrictive-protection vo, 51, 344
is-less-than vf, 575, 575
lexical comparison, 564, 578
LFLOAT pi, 17, 288, 297, 297, 298
m-lfloat-designator vo, 290, 298
t-lfloat-designator vf, 289, 298
wf-lfloat-designator vf, 289, 298
library, 4
library module, xv, 1, 2, 4, 4–6, 9, 11–14, 30, 382, 392, 637, 638, 660, 666–668, 672, 676
LISTEN sp, 369, 370, 373, 666
m-LISTEN vo, 371, 373
wf-LISTEN vf, 370, 373
ln vo, 418
ln lp, 418, 418, 420, 425, 425
LOC, 11, 264, 279, 283, 284, 287, 288, 324, 350–352, 361, 400–403, 666, 671
Loc vt, 334
Loc-pair vt, 332
is-loc-type vf, 324, 324
local module, 6, 23–25, 27, 31, 34, 40–44, 46–50, 52, 63–67, 72, 79–82, 126, 127, 129, 131–133, 136–140, 168, 620, 661, 662, 664–666, 668, 669
local module declaration cs, 64, 79, 641, 642
local variable, 62, 75, 78, 133, 168, 266, 375, 660, 666
locs-of vf, 151, 336, 338, 403, 404
locsperbyte vv, 352
locsperbyte-constant-value vv, 352
locsperword vv, 352
locsperword-constant-value vv, 352
logical conjunction operator cs, 19, 191, 648, 651
logical disjunction operator cs, 19, 191, 648, 651
logical negation operator cs, 19, 189, 648, 651
long complex type, 92, 194, 273, 273, 274, 293, 295, 301
long real type, 273
long-complex-approximation vf, 194, 194
long-real-approximation vf, 197, 197
LONGCOMPLEX pi, 17, 92, 273, 273, 415, 423, 661
LongComplexMath lm, 415, 423, 424, 428, 429
LongConv lm, 581, 598, 600, 603–605
LongIO lm, 430, 434, 442, 444, 457, 459
LongMath lm, 415, 417, 420, 421
LONGREAL pi, 17, 92, 273, 273, 298, 299, 405, 408, 415, 417, 442, 444, 582, 586, 588, 592, 598, 600, 603, 661, 669
LongStr lm, 581, 586, 588
Look vo, 464
Look lp, 436, 438, 452, 464, 464, 478, 491, 504, 515, 523
loop statement cs, 146, 163, 645, 647
Loop-statement vt, 163, 655
m-loop-statement vo, 147, 164, 164
wf-loop-statement vf, 146, 164
low line cs, 16
lower case, 413, 637
lower-bound-of vf, 124, 328, 328
lowline, 16
LowLong lm, 300, 399, 405, 407, 408, 411
LowReal lm, 300, 399, 405, 407, 408, 411

M2EXCEPTION sm, 347, 392, 393
M2EXCEPTION-function-designator vt, 393, 655
m-M2EXCEPTION-function-designator vo, 348, 394
t-M2EXCEPTION-function-designator vf, 348, 394
wf-M2EXCEPTION-function-designator vf, 348, 394
m-M2Exception vo, 394, 394
m2exceptions vv, 393
t-M2Exception vf, 394, 394
wf-M2Exception vf, 394, 394
machine address, 25, 70, 71, 331, 336
machine address cs, 69, 69, 641, 641
main program, *see* program module
make-a-channel vo, 477, 486–488, 500, 501, 514, 527, 534, 540
make-component-active vo, 183, 186, 187, 187
make-current vo, 551, 556
make-function-procden vf, 339
make-me-wait-for vo, 561, 563
make-one-process-active-in vo, 561, 563
make-passive vo, 550, 551, 556
make-proper-procden vf, 339, 482, 495, 509, 519, 526, 541, 542, 594, 596, 602
make-ready vo, 549, 550, 556
make-subarray vo, 479–481, 492–494, 506–508, 516–518, 525, 529
make-waiting vo, 551, 557
MAKEADR sp, 70, 253, 350, 353, 359
m-MAKEADR vo, 354, 359
t-MAKEADR vf, 354, 359
wf-MAKEADR vf, 354, 359
MakeChan vo, 534
MakeChan lp, 531, 534, 534
mandatory-detection vo, 51, 144, 145, 150, 155, 160, 185, 188, 194, 195, 197, 198, 201, 226, 229, 236, 244, 285, 287, 306, 337, 355, 357, 358,

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

373, 375, 383, 386, 395, 396, 403, 421, 429, 470, 510, 555, 563, 598, 605
map-indices vf, 267, 268, 269
map-to-seq vf, 448, 480, 481, 493, 494, 504–508, 517, 518, 579, 615, 615
mark-exported-from-protected vf, 32, 80, 318
match-fields vf, 240, 240
match-fields-list vf, 238, 240, 240, 240
match-fixed-fields vf, 240, 240
match-variant vf, 240, 240
match-variant-fields vf, 240, 240
matching-pointer-type vf, 188, 188, 228
MAX pi, 260, 288, 298, 298, 299, 408
m-max-designator vo, 290, 299
t-max-designator vf, 289, 298
wf-max-designator vf, 289, 298
max-long-real-value vv, 278
max-real-value vv, 278
max-rr-value vv, 278
max-signed-value vv, 278
max-unsigned-value vv, 278
max-zz-value vv, 278
maximum vf, 249, 250, 283, 299, 328, 328, 362, 363
maximum-of-basic-type vf, 192, 276, 277, 329
maximum-of-enumeration vf, 329, 329
maximum-of-subrange vf, 329, 329
maxs vf, 276, 329, 509, 614
MAXSECONDPARTS vv, 609
McCarthy expression cs, 612
Me vo, 552
Me lp, 552, 552
meaning of a program, 5, 24, 30, 665
Member vt, 244, 655
member cs, 243, 244, 649, 649
m-member vo, 244, 245
wf-membership-expression vf, 211, 211
t-member vf, 243, 245
wf-member vf, 243, 245
mere textual scan, 253
merge-compilation-modules vo, 26, 26
merge-environment vf, 33, 54, 55, 64, 65, 82, 319, 319, 320, 454
merge-environments vf, 26, 29, 30, 32, 35, 42, 80, 89, 319, 319
merge-modules vo, 26, 26
merged-value-store vf, 335, 335
Message vt, 334
MIN pi, 260, 288, 299, 299, 300
Min vt, 610
m-min-designator vo, 290, 300
t-min-designator vf, 289, 299
wf-min-designator vf, 289, 299
min-long-real-value vv, 278
min-real-value vv, 278
min-rr-value vv, 278
min-signed-value vv, 278
min-zz-value vv, 278
minimum vf, 300, 328, 329, 362, 363
minimum-of-basic-type vf, 192, 276, 277, 329
minimum-of-enumeration vf, 329, 329
minimum-of-subrange vf, 329, 329
mins vf, 38, 277, 297, 329, 614, 615
minus operator cs, 19, 191, 648, 651
minus-character vv, 275
m-minus vo, 221, 221
mod operator cs, 19, 191, 648, 651
Modula-2 implementation, xv, 4, 5, 10–13, 30, 50, 66, 665
modula2 source code cs, 16
module block cs, 27, 31, 79, 131, 639, 642, 645
module body cs, 131, 131, 645, 645
module dependency, 5, 23, 37, 38, 81, 430, 431, 460, 461, 470, 475, 668, 669, 671
module identifier, 24, 27, 29, 31, 38, 44, 45, 52, 53, 79–81
module identifier cs, 27, 27, 29, 31, 44, 53, 79, 639, 639, 639, 640, 642
module initialization, 4, 23, 25, 27, 28, 31, 33, 34, 38, 40, 41, 50, 82, 126, 127, 129–134, 136–140, 142, 375, 382, 403, 421, 429, 470, 510, 527, 531, 555, 563, 581, 582, 586, 598, 605, 637, 661, 666, 669
module priority, *see* protection
Module-block vt, 132, 655
d-module-block vf, 80, 80
m-module-block vo, 28, 32, 81, 132
wf-module-body vf, 28, 32, 81, 132
Module-declaration vt, 80, 655
d-module-declaration vf, 64, 80
m-module-declaration vo, 66, 81
wf-module-declaration vf, 65, 80
Module-environment vt, 312
module-environment vf, 35, 35
Module-finalization vt, 334
Module-order vt, 36
wf-module-pair vf, 25, 34
module-protection-domain vf, 51, 318
is-module vf, 45, 317
Modules vt, 308
wf-modules vf, 24, 25
wf-module vf, 67, 81
modulus vf, 424, 425, 429
Month vt, 609
most-negative-step-value vv, 165
most-positive-step-value vv, 165
multiplication operator cs, 19, 191, 648, 651
MyParam vo, 553
MyParam lp, 553, 553

national alphanumeric cs, 22
national character cs, 21
national graphic character cs, 21
national letter cs, 22
national numeric cs, 22

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

nested procedure, 63, 623, 669
NEW pi, 259, 280, 283, 287, 287, 399
new line cs, 21
new ordinal type cs, 91, 93, 93, 642, 642
new type cs, 91, 93, 642, 642
New-chan-id vt, 536
m-new-designator vo, 281, 288
wf-new-designator vf, 281, 287
New-dev-id vt, 536
new-level vf, 74, 78, 318
new-passive-process vo, 549, 556
New-type vt, 93, 655
d-new-type vf, 92, 93
wf-new-type vf, 92, 94
NEWCOROUTINE sp, 369, 370, 373–375, 400, 401, 666
m-NEWCOROUTINE vo, 371, 375
wf-NEWCOROUTINE vf, 370, 374
NewPos vo, 503
NewPos lp, 502, 502
next-module vo, 41, 341
NextArg vo, 523
NextArg lp, 523, 523
NIL pi, 6, 99, 188, 215, 279, 279, 400, 551, 555, 663
nil type, 279
NIL-DEREFERENCE ex, 188, 229
nilDeallocation vv, 402
NO-CURRENT-EXCEPTION ex, 386, 395
NO-EXCEPTION-SOURCE ex, 383, 403, 421, 429, 470, 510, 555, 563, 598, 605
no-illegal-redeclarations vf, 35, 35
no-imports-from-definition-module vf, 35, 35
no-leading-char vf, 444, 451, 589
no-leading-digits vf, 444, 451, 589
no-leading-exponent vf, 444, 451, 589
no-leading-sign vf, 440, 444, 451, 584, 589
no-leading-spaces vf, 437, 440, 441, 444, 445, 451, 584, 585, 589, 595–597, 603
no-retry-statements vf, 142, 177
noMixedOperations vv, 475
non-mandatory-detection vo, 183, 186, 188, 199, 224, 227, 228, 335, 355, 357–359, 365, 378, 397, 480, 481, 492–494, 506–508, 517, 518, 525
wf-non-result-body vf, 126, 132, 142
NONEXISTENT ex, 188, 228
noRawOperations vv, 475
normal part cs, 142, 645
noRoomOnDevice vv, 475
noSuchFile vv, 475
not equals operator cs, 19
not equals synonym cs, 19
not keyword cs, 19
not synonym cs, 19
NOT-ATTACHED ex, 373
NOT-CURRENT-SOURCE ex, 386, 395
notAChannel vv, 470
notavail-proc vo, 542
notAvailable vv, 470
Note, 4–17, 21, 23–25, 29, 32, 34, 41, 42, 44, 47–49, 51, 52, 54, 55, 57, 61, 63–66, 68–70, 72–75, 77, 78, 81, 82, 91, 98–100, 107, 111, 116, 117, 121, 123, 127, 129, 133–137, 139, 144, 145, 148, 149, 152, 154, 155, 158–160, 162–165, 167–170, 185, 188, 189, 191, 193, 194, 196, 199–203, 205, 206, 208, 209, 212, 213, 215, 216, 219–222, 224, 226, 228, 229, 231, 234, 235, 238, 239, 243, 244, 247–253, 264–267, 272–274, 276, 279–281, 283, 286, 287, 290–294, 297, 298, 300–305, 324, 350, 352, 353, 355–357, 359, 361–365, 370–372, 374–376, 381, 397, 400–402, 408–411, 417, 423–425, 431, 433, 444–447, 460, 464–468, 470, 473, 477, 478, 486–488, 490, 499–501, 503, 504, 513, 515, 523, 536, 545, 548–553, 560, 561, 563, 567–577, 582, 588, 592, 593, 595, 597, 600, 603, 604, 608, 609, 616
note-HALT vo, 285, 341, 549
noTextOperations vv, 475
notKnown vv, 449
m-not vo, 222, 222
null character, *see* string terminator
null device, 431, 433, 541
null statement, *see* empty statement
null-dens vv, 542
null-look vo, 542
null-name vo, 543
null-ops vv, 542
null-proc vo, 543
null-props vv, 542
null-readraw vo, 543
null-readtext vo, 542
null-skip vo, 542
null-skiplook vo, 542
null-table vv, 542
NullChan vo, 433
NullChan lp, 433, 433
nullchan vf, 433, 538
Number vt, 333
number
  decimal, *see* decimal number
  hexadecimal, *see* hexadecimal number
  octal, *see* octal number
  ordinal, *see* ordinal number
number-of-repetitions vf, 234, 234
number-of-tags-in-field-list vf, 367, 367
number-of-tags-in-fields vf, 366, 367, 367
number-of-tags-in-variant vf, 367, 367
number-of-tags-in-variant-fields vf, 367, 367
Number-type vt, 308
is-number-type vf, 294, 298, 305, 324
numeric-value vf, 584–586, 589–591, 606, 606

obtain-string vo, 570–572, 574, 579
octal digit cs, 20
octal number cs, 20

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ODD pi, 288, 300, 300, 301
m-odd-designator vo, 290, 301
t-odd-designator vf, 289, 300
wf-odd-designator vf, 289, 300
oldFlag vv, 474
opaque export, 34
opaque type, 28, 31, 34, 56, 57, 91, 214, 321, 328, 347, 662, 663, 666, 673
  redeclaration, 34
opaque type definition cs, 56, 56, 640, 640
Opaque-type-definition vt, 56, 655
d-opaque-type-definition vf, 57, 57
wf-opaque-type-definition vf, 57, 57
is-opaque-type vf, 36, 215, 321, 322, 328
opaque-types-declared vf, 35, 36
opaque-types-of vf, 35, 36, 320, 328
Open lp, 476, 476, 513, 513
open array component type cs, 104, 104, 643, 643
open array formal type cs, 104, 104, 643, 643
open array parameter, 104, 105, 122, 123, 231, 262–268, 279, 294, 302, 328, 383, 566, 661, 667, 671
is-open-array-component-compatible vf, 263, 263
Open-array-typed vt, 310
is-open-array-value-parameter-compatible vf, 263, 263
is-open-array-variable-parameter-compatible vf, 263, 264
is-open-array vf, 123, 124, 184, 295, 303, 321, 328
Open-component-type vt, 104, 655
d-open-component-type vf, 105, 105
wf-open-component-type vf, 106, 106
OpenAppend vo, 487
OpenAppend lp, 487, 487
OpenClean vo, 501
OpenClean lp, 500, 500
opened vv, 475
OpenOld vo, 500
OpenOld lp, 499, 499
OpenRead vo, 488
OpenRead lp, 488, 488
openresults vv, 474
OpenWrite vo, 486
OpenWrite lp, 486, 486
Operation vt, 536
operator cs, 18
operator precedence, 189
is-optdigits vf, 451
is-optminus vf, 589, 606
is-optsign vf, 451
is-optspaces vf, 450
ORD pi, 91, 288, 301, 301, 667, 672
m-ord-designator vo, 290, 301
t-ord-designator vf, 289, 301
wf-ord-designator vf, 289, 301
order of compilation, *see* compilation order
order of Evaluation, 5, 10, 13, 149, 153, 158, 166, 167, 185, 191, 226, 230, 233, 238, 244, 246, 259, 265, 660, 665, 667
ordered-values-of vf, 234, 306, 330
ordinal expression cs, 159, 165, 184, 190, 245, 246, 646, 647, 648, 649
ordinal type denoter cs, 91, 97, 106, 642, 643, 644
ordinal type identifier cs, 91, 92, 95, 110, 642, 642, 642, 644
is-ordinal-type vf, 97–99, 107, 111, 160, 166, 211, 246, 247, 282, 296, 301, 322, 322
Ordinal-value vt, 333
other expression cs, 612
Other-protection vt, 333
otherProblem vv, 475
out-ops vv, 540
out-props vv, 540
out-table vv, 540
OutChan vo, 433
OutChan lp, 433, 433
outchan vo, 433, 538
outer-defs-environment vf, 24, 26
outOfChans vv, 475
outOfRange vv, 449
overflow, 194, 196, 199, 265, 375, 424
overwrite-const-environment vf, 95, 157, 314
overwrite-environment vf, 29, 53, 55, 56, 65–67, 75, 78, 81, 126, 129, 132, 319
overwrite-mod-environment vf, 28, 32, 33, 317
overwrite-proc-environment vf, 316
overwrite-struc-environment vf, 95–97, 99–102, 107, 108, 269, 315, 453, 454
overwrite-type-environment vf, 314
overwrite-var-environment vf, 157, 266–268, 316

P vo, 601
pa-arg-left vo, 525, 526, 527
pa-chan-id vo, 523, 528
pa-init-props vo, 527, 527
pa-inspect-arg vo, 523, 524, 526
pa-is-present vo, 523, 527, 527
pa-read-chars-up-to vo, 525, 526
pa-reset-args vo, 526, 528
pa-skip-between vo, 523, 528
pa-skip-within vo, 524–526, 528
Packed-set-structure vt, 309
a-packed-set-structure vf, 114, 115
Packed-set-type vt, 98, 655
d-packed-set-type vf, 94, 98
is-packed-set-type vf, 322
wf-packed-set-type vf, 94, 99
packedset type cs, 93, 98, 642, 643
pad vf, 505, 507, 508, 509
padding vv, 606
padding-value vv, 509
is-parameter-compatible vf, 261, 262
Parameter-formal-type vt, 104, 655
d-parameter-formal-type vf, 105, 105
wf-parameter-formal-type vf, 105, 105
parameter-types-of vf, 326

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

is-parameters-match vf, 152, 230, 261
passiveProgram vv, 554
PDP-11, 280
period cs, 17, 651
period-character vv, 275
permission, xiv, 489
pervasive identifier, 4, 4, 9, 16, 17, 23, 27, 28, 32, 44, 50, 52, 53, 92, 97–99, 188, 215, 280, 288, 412, 661, 662, 668, 670, 671, 676
pervasive identifier cs, 16
Pid vt, 555
plus operator cs, 18, 191, 648, 651
plus-character vv, 275
m-plus vo, 221, 221
pointer type, 6, 34, 55, 57, 64, 69, 91, 99, 100, 115, 122–125, 187, 188, 214, 215, 228, 263, 279, 283, 284, 287, 288, 321, 325, 326, 353–357, 374, 400, 401, 531, 660, 662–664, 673
pointer type cs, 93, 99, 642, 643
pointer value, 6, 99, 187, 188, 214–216, 227, 228, 326, 355, 356, 400, 401, 663
pointer value cs, 228, 228, 649, 649
pointer variable designator cs, 187, 187, 647, 647
wf-pointer-and-address-assignment vf, 123, 125
wf-pointer-assignment vf, 123, 124
m-pointer-eq vo, 215, 216
m-pointer-ne vo, 215, 216
Pointer-relational-operation vt, 214, 656
m-pointer-relational-operation vo, 207, 215
t-pointer-relational-operation vf, 206, 215
wf-pointer-relational-operation vf, 206, 215
Pointer-relational-operator vt, 215, 656
m-pointer-relational-operator vo, 215, 215
Pointer-structure vt, 309
a-pointer-structure vf, 114, 115
Pointer-type vt, 99, 656
d-pointer-type vf, 94, 99
is-pointer-type vf, 36, 124, 125, 187, 215, 228, 263, 284, 288, 321, 322, 355–357, 374
wf-pointer-type vf, 94, 100
Pointer-value vt, 333
Pointer-variable vt, 312
pointerToUnallocatedStorage vv, 402
polarToComplex vo, 427
`polarToComplex` lp, 427, 427
pool-locs vf, 332, 338, 338, 404
pop-exception-context vo, 128, 130, 131, 136, 138, 140, 144, 145, 342, 342
portable identifier cs, 16
pos-to-filepos vo, 502, 503, 504
position
  in random access file, *see* GetPos, SetPos
  of substring, *see* Pos
power vo, 419
`power` lp, 97, 249, 417, 419, 419, 419, 425, 425
pragma, *see* source code directive
precondition, 545
pred vf, 168, 168
`pred` lp, 409, 409
predecessor, 282
predefined, 4, 4, 5, 8, 9, 11, 12, 17, 23, 52, 70, 91–93, 96–100, 102, 105, 107, 109, 111, 152, 153, 170, 189, 217, 229, 230, 258–261, 265–267, 270–273, 280, 347, 370, 399, 408, 660–662, 666, 668, 670, 671
Predefined-function-procedure-typed vt, 312
Predefined-procedure-typed vt, 311
Predefined-proper-procedure-typed vt, 311
preferred case separator cs, 18
preferred dereferencing operator cs, 20
preferred left brace cs, 18
preferred left bracket cs, 18
preferred right brace cs, 18
preferred right bracket cs, 18
Prefix-expression vt, 219, 656
m-prefix-expression vo, 190, 219
t-prefix-expression vf, 190, 219
wf-prefix-expression vf, 190, 219
Prefix-operation vt, 219, 656
m-prefix-operation vo, 219, 220
t-prefix-operation vf, 219, 220
wf-prefix-operation vf, 219, 219
principal value, 425, 426
priority, *see* protection
`PROC` pi, 92, 280, 280, 662
proc type, 280
m-proc-eq vo, 214, 214
m-proc-ne vo, 214, 214
proc-structure vv, 280
Procdata vt, 555
Procdens vt, 334
procedure body cs, 126, 126, 644, 644
procedure call cs, 146, 152, 645, 645
procedure declaration cs, 64, 72, 641, 641
procedure designator cs, 152, 152, 645, 645
procedure heading cs, 54, 58, 640, 640
procedure identifier cs, 58, 60, 73, 73, 76, 640–642, 642
procedure parameter, *see* actual parameter, formal parameter
procedure type, 6, 55, 64, 69, 91, 92, 100–103, 105, 122, 124, 152, 212, 213, 229, 254, 271, 279, 280, 321, 347, 660, 662, 664, 671, 673
procedure type cs, 93, 100, 642, 643
wf-procedure-assignment vf, 123, 124
Procedure-call vt, 152, 656
m-procedure-call vo, 147, 153
wf-procedure-call vf, 146, 152
Procedure-declaration vt, 72, 656
d-procedure-declaration vf, 64, 72
m-procedure-declaration vo, 66, 73
is-procedure-declarations-distinct vf, 74, 76
wf-procedure-declaration vf, 65, 72
Procedure-designator vt, 152, 656
Procedure-heading vt, 58, 656

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

d-procedure-heading-declaration vf, 55, 58
wf-procedure-heading vf, 56, 58
Procedure-id vt, 311
procedure-level vf, 124, 316
Procedure-relational-operation vt, 213, 656
m-procedure-relational-operation vo, 207, 213
t-procedure-relational-operation vf, 206, 213
wf-procedure-relational-operation vf, 206, 213
Procedure-relational-operator vt, 214, 656
m-procedure-relational-operator vo, 213, 214
Procedure-structure vt, 309
procedure-structure vf, 124, 213, 315, 316
a-procedure-structure vf, 114, 116
Procedure-type vt, 100, 656
Procedure-typed vt, 311
d-procedure-type vf, 94, 100
is-procedure-type vf, 124, 213, 321, 321
wf-procedure-type vf, 94, 100
Procedure-value vt, 311
is-procedure vf, 224, 254, 316
Procedures vt, 308
procedures-declared vf, 35, 36, 72
procedures-of vf, 35, 36, 316, 320
Procedures-typed vt, 308
wf-procedure vf, 67, 73
process, 6, 10, 11, 53, 96, 112, 152, 158, 161, 165, 235, 544, 545, 548–551, 555, 560, 561, 662, 668, 670, 672, 674
processError vv, 554
`Processes` lm, 544, 545, 553–555, 560, 563, 666, 668, 672
processes-exception vo, 549–551, 554
processes-exception-detected vf, 554, 554
processes-exception-string vo, 554, 554
processes-initialization vo, 555
ProcessesException vo, 553
`ProcessesException` lp, 553, 553
processesexceptions vv, 554
processor, 3, 5, 5, 10, 15, 544, 545, 548, 608, 665, 668, 672
Program vt, 24, 656
program module cs, 23, 27, 639, 639
Program-argument vt, 527
Program-cont vt, 344
program-handlers vo, 343, 378
Program-module vt, 27, 656
m-program-module vo, 26, 28
wf-program-module vf, 24, 27
is-program-terminating vo, 132, 340, 390
`ProgramArgs` lm, 431, 522–528
programargs-dens vv, 526
programargs-getname vo, 526
programargs-initialization vo, 527
programargs-look vo, 523, 524
programargs-ops vv, 526
programargs-rawread vo, 525
programargs-reset vo, 526
programargs-skip vo, 524, 524
programargs-skiplook vo, 524
programargs-textread vo, 524
m-program vf, 25, 638
Programming in Modula-2, xv, 29, 34, 38, 41, 50, 73, 76, 91, 95, 97, 98, 121, 129, 160, 165, 169, 189, 198, 205, 206, 214, 215, 231, 242, 243, 252, 264, 273, 274, 280, 281, 286, 288, 290, 291, 294, 302, 304, 347, 364, 369, 370, 372, 374, 375, 400, 545, 567, 660, 661, 664, 668, 671
wf-program vf, 24
proper procedure, 57–59, 71, 73–75, 77, 100, 101, 122, 124, 126–128, 140, 142, 152–154, 169, 254, 258, 261, 265, 270, 279, 280, 283, 287, 566, 637, 638, 661, 670
proper procedure block cs, 73, 126, 642, 644
proper procedure declaration cs, 72, 73, 641, 642
proper procedure heading cs, 58, 58, 73, 640, 640, 642
proper procedure type cs, 100, 101, 643, 643
Proper-procedure-block vt, 126, 656
m-proper-procedure-block-h vo, 75, 128
m-proper-procedure-block-hp vo, 75, 128
m-proper-procedure-block-p vo, 75, 127
m-proper-procedure-block vo, 75, 127
wf-proper-procedure-body vf, 74, 126
Proper-procedure-cont vt, 344
Proper-procedure-declaration vt, 73, 656
d-proper-procedure-declaration vf, 72, 73
m-proper-procedure-declaration vo, 73, 75
wf-proper-procedure-declaration vf, 72, 74
Proper-procedure-heading vt, 58, 656
d-proper-procedure-heading-declaration vf, 58, 59, 73
d-proper-procedure-heading vf, 74, 75
wf-proper-procedure-heading vf, 58, 59, 74
Proper-procedure-structure vt, 309
a-proper-procedure-structure vf, 116, 116
Proper-procedure-type vt, 101, 656
Proper-procedure-typed vt, 311
d-proper-procedure-type vf, 100, 101
is-proper-procedure-type vf, 321, 321
wf-proper-procedure-type vf, 100, 101
wf-proper-procedure vf, 73, 74
`PROT` sp, 369, 376, 379
m-prot-ge vo, 217, 217
m-prot-le vo, 217, 217
is-protected-domain vf, 318
is-protected-module vf, 51, 133, 135, 137, 139
is-protected-procedure vf, 51, 75, 79
`PROTECTION` pi, 17, 92, 280, 280
Protection vt, 333
protection
  protected Module, 50, 51, 79, 81
  protection expression, 27, 28, 31, 32, 50, 51, 79, 668
protection expression cs, 50, 50, 640, 640
protection type, 280

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Protection-domain vt, 313
is-protection-less-than-or-equal vo, 217, 217, 344
Protection-relational-operation vt, 216, 656
m-protection-relational-operation vo, 207, 216
t-protection-relational-operation vf, 206, 216
wf-protection-relational-operation vf, 206, 216
Protection-relational-operator vt, 216, 656
m-protection-relational-operator vo, 216, 217
m-PROT vo, 377, 379
t-PROT vf, 377, 379
wf-PROT vf, 377, 379
punctuation character cs, 22
pure-string vf, 439, 477, 486–488, 500, 501, 535, 567, 569, 571–578, 579, 579, 584, 585, 589, 595–597, 603
push-exception-context vo, 128, 130, 131, 136, 138, 140, 342, 342
put-coroutine-storage vo, 337, 338, 339
put-heap-storage vo, 403, 403

Qualident vt, 53, 656
wf-qualident vf, 53, 93, 97, 111, 183, 224, 232, 237, 242, 261
Qualified vt, 53, 656
qualified export cs, 46, 47, 640, 640
qualified identifier cs, 53, 92, 183, 223, 640, 642, 647, 648
Qualified-environment vt, 312
Qualified-export vt, 48, 656
d-qualified-export vf, 46, 48
wf-qualified-export vf, 46, 48
wf-qualified vf, 53, 53, 53
qualifying identifier cs, 53, 53, 640, 640
queue-termination vo, 33, 133, 134, 341
quotable character cs, 20
quoted string cs, 20

radix, 248, 408
RAISE sp, 381–383
raise-chan-exception vo, 470, 471, 471
raise-exception vo, 342, 383, 398
raise-processes-exception vo, 554, 554
raise-storage-exception vo, 402, 402
Raised-exception vt, 334
RAISEdevException vo, 535
RAISEdevException lp, 531, 535, 535
m-RAISE vo, 382, 383, 402, 421, 428, 471, 510, 554, 562, 597, 605
wf-RAISE vf, 382, 383
range type cs, 95, 95, 642, 642
range-type-of vf, 325, 325
rawFlag vv, 474
RawIO lm, 430, 434, 447, 459
RawRead vo, 467
RawRead lp, 466, 466, 467, 480, 493, 507, 517, 525
RawWrite vo, 467
RawWrite lp, 467, 467, 481, 494, 508, 518
rdenum vf, 440, 442, 445, 449
rdresult vo, 440, 441, 445, 450, 460, 469, 539
RE pi, 17, 260, 288, 301, 301, 301, 302
m-re-designator vo, 290, 302
t-re-designator vf, 289, 302
wf-re-designator vf, 289, 301
re-max-cc-value vv, 278
re-max-complex-value vv, 278
re-max-long-complex-value vv, 278
re-min-cc-value vv, 278
re-min-complex-value vv, 278
re-min-long-complex-value vv, 278
Read vo, 448
Read lp, 448, 448
read result, 434, 436–438, 440, 441, 444, 448, 452, 464–468, 531
read-chars vo, 437, 453
read-external-IOLib-state vo, 537
read-external-program-state vo, 25, 331
read-external-ProgramArgs-state vo, 528
read-file vo, 477, 487, 488, 500, 537
read-locs vo, 448, 453
read-until-false vo, 452, 452
read-while vo, 436, 437, 440, 441, 445, 452
ReadCard vo, 441
ReadCard lp, 441, 441
ReadChar vo, 436
ReadChar lp, 436, 436
readFlag vv, 474
ReadInt vo, 440
ReadInt lp, 440, 440
ReadReal vo, 445
ReadReal lp, 444, 444
ReadRestLine vo, 436
ReadRestLine lp, 436, 436
ReadResult lp, 450, 450, 460, 460, 469, 469
readresults vv, 449
ReadString vo, 437
ReadString lp, 436, 437
ReadToken vo, 437
ReadToken lp, 12, 437, 437, 582
REAL pi, 92, 272, 272, 294, 298, 299, 405, 408, 415, 417, 442, 444, 582, 586, 588, 592, 598, 600, 661, 669
real literal cs, 20, 247, 650, 651
real type, 272
m-real-add vo, 197, 198
real-approximation vf, 197, 197
wf-real-assignment vf, 123, 124
is-real-compatible vf, 121, 122
real-constant-approximation vf, 197, 197, 250
m-real-divide vo, 197, 198
Real-literal vt, 249, 656
m-real-literal vo, 248, 250
t-real-literal vf, 248, 250
wf-real-literal vf, 248, 250
m-real-multiply vo, 197, 198
Real-number-operation vt, 196, 656

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

m-real-number-operation vo, 193, 196
t-real-number-operation vf, 193, 196
wf-real-number-operation vf, 192, 196
Real-number-operator vt, 197, 656
m-real-number-operator vo, 196, 197
Real-number-type vt, 308
REAL-OVERFLOW ex, 197
real-result-type vf, 196, 197
m-real-subtract vo, 197, 198
Real-type, 121, 122, 124, 191, 192, 196, 197, 249, 250, 254, 272, 273, 293, 295, 301
REAL-ZERO-DIVISION ex, 198
`RealConv` lm, 581, 598, 600, 603–605
realconv-exception vo, 603, 605
realconv-exception-string vo, 605, 605
realconv-initialization vo, 605
Realconvexception vt, 605
`RealIO` lm, 430, 434, 442, 444, 457, 459, 581
`RealMath` lm, 415, 417, 420, 421
realmath-exception vo, 417–420, 420
realmath-exception-string vo, 420, 421
realmath-initialization vo, 421
Realmathexception vt, 420
`RealStr` lm, 581, 588
realstr vf, 447, 591, 591
RealToEng vo, 590
`RealToEng` lp, 590, 590
RealToFixed vo, 591
`RealToFixed` lp, 591, 591
RealToFloat vo, 589
`RealToFloat` lp, 589, 589
RealToStr vo, 591
`RealToStr` lp, 591, 591
record assignment, 149, 240
record constructed value cs, 235, 236, 238, 649, 649
record constructor cs, 231, 236, 649, 649
record designator cs, 156, 156, 646, 646
record type, 49, 91, 107–109, 113, 227, 230, 233, 236–238, 240, 243, 283, 287, 303, 321, 327, 365, 531, 671
record type cs, 93, 107, 642, 644
record type identifier cs, 236, 236, 649, 649
record value cs, 227, 227, 648, 648
record variable designator cs, 185, 185, 647, 647
a-record-component vf, 118, 118, 118
Record-components vt, 313
Record-constructed-value vt, 238, 656
Record-constructor vt, 237, 657
m-record-constructor vo, 231, 237
t-record-constructor vf, 231, 237
wf-record-constructor vf, 231, 237
wf-record-definition vf, 236, 237, 238
Record-designator vt, 156, 657
Record-structure vt, 310
a-record-structure vf, 114, 117
Record-type vt, 108, 657
d-record-type vf, 94, 108
is-record-type vf, 157, 186, 227, 238, 321, 321
wf-record-type vf, 94, 108
Record-value vt, 334
Record-variable vt, 313
record-variable-assignment vo, 150, 151
relational operator cs, 189, 190, 648, 648
Relational-operation vt, 206, 657
m-relational-operation vo, 193, 207
t-relational-operation vf, 193, 206
wf-relational-operation vf, 192, 206
Release vo, 561
`Release` lp, 560, 561, 561
release-a-channel vo, 478, 490, 504, 515, 534, 540
rem operator cs, 19, 191, 648, 651
remove vf, 343, 343
remove-aliases vo, 336, 338
remove-from-environment vf, 65, 314–317, 320, 320
repeat statement cs, 146, 162, 645, 646
Repeat-statement vt, 163, 657
m-repeat-statement vo, 147, 163, 163
wf-repeat-statement vf, 146, 163
repeated structure component cs, 233, 234, 649, 649
Repeated-element vt, 235, 657
Repeated-elements vt, 233, 657
wf-repeated-element vf, 233, 235
repetition factor cs, 234, 234, 649, 649
Replace vo, 572
`Replace` lp, 571, 572, 572
representation, 4, 5, 7, 8, 10, 97–99, 112, 152, 159, 161, 165, 184, 189, 225, 235, 268, 274, 279, 348, 351, 352, 361, 364, 365, 408, 412, 464, 503, 564, 581, 590, 591, 595, 597, 603, 604, 638, 661, 664, 670, 672
  external representation (of data), 464, 503
  internal representation (of data), 252, 276, 464
required case separator cs, 18
required dereferencing operator cs, 20
required left brace cs, 18
required left bracket cs, 18
required right brace cs, 18
required right bracket cs, 18
required symbol cs, 17
required-capitalizations vv, 292
required-characters vv, 275
required-lower-cases vv, 275
required-lower-cases-order vv, 275
required-other-characters vv, 275
required-upper-cases vv, 275
required-upper-cases-order vv, 275
Reread vo, 489
`Reread` lp, 486, 487, 489, 489
reserve-coroutine-workspace vo, 345, 375
reserved word, 17
Reset vo, 468
`Reset` lp, 468, 468, 481, 494, 508, 518, 526, 541
residual-environment vf, 35, 35

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

rest vf, 67, 234, 239–241, 366, 367, 491–493, 504–507, 614, 615
restrict-environment vf, 29, 30, 35, 44, 45, 47, 48, 320, 337
wf-result-body vf, 129, 144
resume vf, 345
resume vo, 343, 345, 372, 376
retry statement cs, 146, 155, 645, 646
Retry-statement vt, 155, 657
m-retry-statement vo, 147, 156
retry-statements-of vf, 177, 177
wf-retry-statement vf, 146, 156
return vf, 612
return statement cs, 146, 153, 645, 645
RETURN-RANGE ex, 155
Return-statement vt, 153, 657
m-return-statement vo, 147, 153
return-statements-of vf, 144, 176, 177
wf-return-statement vf, 146, 153
return-types-of vf, 326
rev vf, 614, 614
revstring vf, 577, 577, 577, 578
Rewrite vo, 489
`Rewrite` lp, 488, 489, 489
right brace cs, 18, 651
right bracket cs, 18, 651
right parenthesis cs, 17, 651
`RndFile` lm, 431, 497, 499, 501, 502, 504–507, 509, 510
rndfile-check-clean vo, 501, 501
rndfile-check-file vo, 499, 500, 501
rndfile-check-old vo, 500, 500
rndfile-Close vo, 504
rndfile-dens vv, 509
rndfile-exception vo, 503, 504, 509
rndfile-exception-string vo, 510, 510
rndfile-flags vv, 499
rndfile-flush vo, 508, 508, 509
rndfile-free vo, 504, 509
rndfile-get-file vo, 502–509, 511
rndfile-get-name vo, 508, 509, 511
rndfile-get-position vo, 502, 504–508, 511
rndfile-get-realprops vo, 503, 508, 511
rndfile-getname vo, 508
rndfile-initialization vo, 510
rndfile-look vo, 504, 505
rndfile-make-a-buffer vo, 500, 501, 510
rndfile-merge-flags vf, 499, 500, 501
rndfile-ops vv, 509
rndfile-rawread vo, 507
rndfile-rawwrite vo, 508
rndfile-release-a-buffer vo, 509, 510
rndfile-reset vo, 508
rndfile-set-file vo, 505, 507, 508, 511
rndfile-set-name vo, 511
rndfile-set-position vo, 503, 506–508, 511
rndfile-skip vo, 505, 505
rndfile-skiplook vo, 505
rndfile-textread vo, 506
rndfile-textwrite vo, 506
rndfile-writeln vo, 505
ROTATE sp, 15, 253, 260, 350, 353, 360–362
rotate vf, 362, 362
m-ROTATE vo, 354, 362
t-ROTATE vf, 354, 362
wf-ROTATE vf, 354, 362
round vo, 420
`round` lp, 305, 410, 410, 420, 420
RS vo, 601

S vo, 594
scalar type, 34, 191, 206, 208, 271, 298, 299, 304, 321, 322, 663
m-scalar-eq vo, 209, 209
m-scalar-ge vo, 209, 210
m-scalar-gt vo, 209, 210
m-scalar-le vo, 209, 210
m-scalar-lt vo, 209, 210
m-scalar-ne vo, 209, 209
Scalar-relational-operation vt, 208, 657
m-scalar-relational-operation vo, 207, 209
t-scalar-relational-operation vf, 206, 208
wf-scalar-relational-operation vf, 206, 208
Scalar-relational-operator vt, 209, 657
m-scalar-relational-operator vo, 209, 209
scalar-set vf, 246, 246
is-scalar-type vf, 208, 298, 299, 305, 321, 322, 329, 330
scalar-value vf, 209, 210, 210, 246, 306, 362, 363, 606
scalarMult vo, 427
`scalarMult` lp, 427, 427
`scale` lp, 20, 249, 410, 410
scale factor, 20, 249
scale factor cs, 20
ScanCard vo, 596
`ScanCard` lp, 593, 595, 595
scancard-map vv, 596
scanclass vv, 606
ScanInt vo, 594
`ScanInt` lp, 592, 593, 593
scanint-map vv, 594
ScanReal vo, 601
`ScanReal` lp, 600, 601
scanreal-map vv, 602
scope of an identifier, 4, 7, 9, 17, 41, 43, 44, 52, 54, 55, 61, 63–65, 80, 94, 267, 668, 669
SE vo, 602
Sec vt, 610
select vo, 549–551, 557
selected designator cs, 182, 185, 647, 647
selected value cs, 222, 227, 648, 648
Selected-designator vt, 185, 657
m-selected-designator vo, 182, 186
t-selected-designator vf, 182, 186
wf-selected-designator vf, 182, 185
Selected-value vt, 227, 657

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

m-selected-value vo, 223, 227
t-selected-value vf, 223, 227
wf-selected-value vf, 223, 227
Semaphores lm, 12, 544, 545, 559, 560, 562, 563, 666
Semaphores-Create vo, 560
semaphores-exception vo, 560, 562
semaphores-exception-string vo, 562, 562
semaphores-initialization vo, 563
semicolon cs, 17, 651
separate module, 4, 4, 5, 8, 11–14, 23–25, 27–34, 36, 38, 42–44, 48–50, 54, 72, 133, 168, 283, 287, 347, 380, 382, 387, 399, 660, 669–671
separator cs, 21
seq-to-map vf, 505, 507, 508, 615
SeqFile lm, 431, 483, 490–496
seqfile-check-append vo, 487, 487
seqfile-check-file vo, 485, 487–489
seqfile-check-read vo, 488, 488
seqfile-check-write vo, 486, 486
seqfile-Close vo, 490
seqfile-dens vv, 495
seqfile-flags vv, 485
seqfile-flush vo, 494, 495, 495
seqfile-free vo, 490, 495
seqfile-get-file vo, 491–495, 496
seqfile-get-name vo, 494, 495, 496
seqfile-get-position vo, 491–493, 496
seqfile-get-realprops vo, 489, 490, 494, 497
seqfile-getname vo, 494
seqfile-look vo, 491, 491
seqfile-make-a-buffer vo, 486–488, 496
seqfile-merge-flags vf, 485, 486–488
seqfile-ops vv, 495
seqfile-rawread vo, 493
seqfile-rawwrite vo, 494
seqfile-release-a-buffer vo, 495, 496
seqfile-reset vo, 494
seqfile-set-file vo, 490, 492–494, 496
seqfile-set-name vo, 496
seqfile-set-position vo, 489, 491–494, 496
seqfile-skip vo, 491, 491
seqfile-skiplook vo, 491
seqfile-textread vo, 492
seqfile-textwrite vo, 493
seqfile-writeln vo, 492
set constructed value cs, 235, 242, 243, 649, 649
set constructor cs, 231, 242, 649, 649
set difference operator cs, 19, 191, 648, 651
set intersection operator cs, 19, 191, 648, 651
set membership, 18, 20, 206, 210, 212
set membership operator cs, 20, 190, 648, 651
set type, 1, 15, 91, 92, 95, 97–99, 191, 203, 206, 210, 216, 230, 241–244, 271, 279, 280, 284–287, 321, 325, 360–363, 660, 663
set type cs, 93, 97, 642, 643
set type identifier cs, 242, 242, 649, 649
set union operator cs, 19, 191, 648, 651
set-aliases vo, 268, 336
Set-constructed-value vt, 243, 657
Set-constructor vt, 242, 657
m-set-constructor vo, 231, 242
t-set-constructor vf, 231, 242
wf-set-constructor vf, 231, 242
Set-definition vt, 112, 657
m-set-definition vo, 160, 236, 242, 244, 257
t-set-definition vf, 114, 161, 243, 243
wf-set-definition vf, 112, 161, 236, 242, 243
m-set-difference vo, 204, 204
m-set-eq vo, 211, 212
set-initializing vo, 26, 340
m-set-intersection vo, 204, 204
m-set-ne vo, 211, 212
Set-operation vt, 203, 657
m-set-operation vo, 193, 203
t-set-operation vf, 193, 203
wf-set-operation vf, 192, 203
Set-operator vt, 204, 657
m-set-operator vo, 203, 204
set-props vo, 489, 490, 494, 503, 508, 528, 539
set-protection vo, 51, 344, 344, 373
SET-RANGE ex, 244
set-rdresult vo, 436, 437, 440, 442, 445, 448, 468, 478–481, 491, 493, 504–507, 515–518, 524, 525, 539, 542, 543
Set-relational-operation vt, 210, 657
m-set-relational-operation vo, 207, 211
t-set-relational-operation vf, 206, 211
wf-set-relational-operation vf, 206, 210
Set-relational-operator vt, 211, 657
m-set-relational-operator vo, 211, 211
Set-structure vt, 309
a-set-structure vf, 114, 115
m-set-symmetric-difference vo, 204, 204
set-terminating vo, 41, 340
set-to-undefined vo, 151, 151
Set-type vt, 97, 657
d-set-type vf, 94, 97
is-set-type vf, 203, 211, 243, 285, 287, 321, 322
wf-set-type vf, 94, 98
m-set-union vo, 204, 204
Set-value vt, 333
SetClock vo, 609
SetClock lp, 609, 609, 609
SetErrChan vo, 434
SetErrChan lp, 434, 434
seterrchan vo, 434, 538
SetInChan vo, 434
SetInChan lp, 434, 434
setinchan vo, 434, 538
setMode lp, 410, 410, 411
SetOutChan vo, 434
SetOutChan lp, 434, 434
setoutchan vo, 434, 538
SetPos vo, 503

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

SetPos lp, 503, 503
SetReadResult vo, 468
SetReadResult lp, 468, 468
shielded vf, 55, 65, 89, 89
shielded-in-a-variant vf, 90, 90
shielded-in-fields vf, 89, 89
shielded-in-fields-list vf, 89, 89, 90
shielded-in-type vf, 89, 89
shielded-in-variants vf, 89, 90
SHIFT sp, 15, 253, 260, 350, 353, 360–363
shift vf, 363, 363
m-SHIFT vo, 354, 363
t-SHIFT vf, 354, 363
wf-SHIFT vf, 354, 363
Sid vt, 562
side effect, 149, 158, 160, 162, 163, 166, 185, 226, 233, 238, 244, 265, 667
side-effect-of-change vf, 335, 336
sign vf, 201, 615
sign cs, 19, 189, 648, 651
sign lp, 18–20, 189, 200, 409, 409, 441, 445–447, 584, 589, 591
signed type, 92, 95, 98, 122, 124, 220, 221, 245, 271, 272, 272, 275, 279, 281, 283, 286, 287, 290, 294–297, 301, 303, 304, 323, 325, 353, 355–358, 361, 363, 374, 383, 385, 660, 661, 664, 669
wf-signed-assignment vf, 123, 124
is-signed-type vf, 323, 323
simple alphanumeric cs, 21
simple expression cs, 189, 189, 648, 648
simple import cs, 43, 43, 639, 639
simple letter cs, 22
simple return statement cs, 153, 154, 645, 645
wf-simple-assignment vf, 123, 123
Simple-import vt, 44, 657
d-simple-import vf, 43, 44
wf-simple-import vf, 43, 44
Simple-return-statement vt, 154, 657
m-simple-return-statement vo, 153, 154
wf-simple-return-statements vf, 143, 176
wf-simple-return-statement vf, 153, 154
is-simple-variable-identifier vf, 166, 315
is-simple-variable vf, 170, 170
sin vo, 418
sin lp, 417, 418, 418, 424, 426, 426
single quote cs, 20
Singleton vt, 247, 657
singleton cs, 244, 246, 649, 649
m-singleton vo, 245, 247
t-singleton vf, 245, 247
wf-singleton vf, 245, 247
SIOResult lm, 430, 431, 455, 459, 460
SIOResult-ReadResult vo, 460
SIZE pi, 254, 259, 260, 264, 272, 284, 288, 302, 302, 303, 365
m-size-designator vo, 290, 303
t-size-designator vf, 289, 303
wf-size-designator vf, 289, 302
size-of-array vf, 233, 328
size-of-array-definition vf, 233, 234
size-of-fields vf, 239, 239, 240
size-of-fields-list vf, 239, 239, 239, 240
size-of-fixed-fields vf, 239, 239
size-of-record vf, 238, 239
size-of-record-definition vf, 238, 239
size-of-variant vf, 239, 240
size-of-variant-fields vf, 239, 239
Skip vo, 465
Skip lp, 436, 438, 452, 464, 464, 465, 478, 491, 505, 515, 524
skip vf, 612
skipAtEnd vv, 470
SkipLine vo, 438
SkipLine lp, 437, 437
SkipLook vo, 465
SkipLook lp, 438, 452, 465, 465, 479, 491, 505, 516, 524
SLongIO lm, 430, 431, 455, 457, 459
SMALL-WORKSPACE ex, 375
smallest addressable unit, *see* LOC
softDeviceError vv, 470
source code, 1, 4, 4, 5, 6, 10, 11, 15, 21, 31, 68, 92, 111, 114, 159, 235, 660–662, 668–670
source code directive, 1, 10, 11, 16, 21, 668, 670
source code directive cs, 21
source program, 670
sourced implementation module, 5, 5, 30, 637, 660
Sourced-implementation-module vt, 31, 657
m-sourced-implementation-module vo, 31, 32
wf-sourced-implementation-module vf, 30, 32
Sourced-module-finalization vt, 334
wf-sourced-module-pair vf, 34, 35
space cs, 21
space-character vv, 275
spaces vf, 441, 442, 446, 447, 451
split vf, 450, 451
sqrt vo, 417
sqrt lp, 417, 417, 425, 425
SRawIO lm, 430, 455, 459
SRealIO lm, 430, 431, 455, 457, 459
stack, 334
standard function, 91, 247, 253, 254, 273, 288, 365, 574, 661, 667, 672
standard library, xv, 1, 2, 4, 4, 5, 9–14, 30, 548, 660, 666, 672
standard library implementation, 10, 12
standard library module implementation, 5, 5, 11, 13
standard mode, 5, 5, 6, 12, 13, 664, 670
standard procedure, 4, 135, 280, 662, 665
Standard-function-designator vt, 288, 658
m-standard-function-designator vo, 230, 290
t-standard-function-designator vf, 230, 289
wf-standard-function-designator vf, 230, 289
Standard-function-procedure-typed vt, 312

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Standard-procedure-designator vt, 280, 658
m-standard-procedure-designator vo, 153, 281
wf-standard-procedure-designator vf, 152, 281
Standard-proper-procedure-typed vt, 312
Start vo, 549
Start lp, 548, 549, 549, 555
StartPos vo, 502
StartPos lp, 502, 502
Statecont vt, 344
Statement vt, 146, 658
statement cs, 146, 147, 645, 645
statement sequence cs, 142, 147, 156, 158, 159, 161–163, 165, 645, 645, 646, 647
Statement-sequence vt, 147, 658
m-statement-sequence vo, 143–145, 148, 148, 157–160, 162–164, 167
wf-statement-sequence vf, 142, 147, 157, 158, 160–164, 166
m-statement vo, 147, 148
wf-statement vf, 146, 147
Std-chan-id vt, 536
Std-dev-id vt, 536
StdChans lm, 430, 431
StdErrChan vo, 433
StdErrChan lp, 433, 433
stderrchan vf, 433, 538
StdInChan vo, 432
StdInChan lp, 432, 432
stdinchan vf, 433, 537
StdOutChan vo, 433
StdOutChan lp, 433, 433
stdoutchan vf, 433, 538
step size cs, 165, 165, 647, 647
wf-step-range vf, 166, 166
STextIO lm, 430, 431, 455, 456, 582
stop vv, 345
StopMe vo, 549
StopMe lp, 548, 549, 549
Storable-value vt, 332
Storage lm, 283, 287, 399, 401, 403
storage unit, 364, 448
storage-as-loc vo, 267, 268, 269
storage-call vv, 284
storage-exception vo, 401, 402
storage-exception-detected vf, 402, 402
storage-exception-string vo, 402, 402
storage-initialization vo, 403
Storage-pool vt, 332
StorageException vo, 401
StorageException lp, 401, 401
storageexceptions vv, 402
Store vt, 332
store-denotation vo, 75, 78, 339
store-procedure-dens vo, 33, 340
strAllRight vv, 606
strcard vf, 441, 585, 585, 596, 597
stream, 11, 12, 430, 431, 436–442, 444–446, 448, 464–467, 472, 475–483, 499, 512, 582
StreamFile lm, 431, 475, 478–483
streamfile-check vo, 477, 477
streamfile-check-file vo, 476, 477
streamfile-Close vo, 478
streamfile-dens vv, 482
streamfile-flags vv, 476
streamfile-flush vo, 481, 482, 482
streamfile-free vo, 478, 482
streamfile-get-ifile vo, 478–480, 483
streamfile-get-name vo, 481, 482, 483
streamfile-get-ofile vo, 479–482, 483
streamfile-getname vo, 481
streamfile-look vo, 478, 479
streamfile-make-a-buffer vo, 477, 483
streamfile-merge-flags vf, 476, 477
streamfile-Open vo, 477
streamfile-ops vv, 482
streamfile-rawread vo, 480
streamfile-rawwrite vo, 481
streamfile-release-a-buffer vo, 482, 483
streamfile-reset vo, 481
streamfile-set-ifile vo, 479–481, 483
streamfile-set-name vo, 483
streamfile-set-ofile vo, 479–481, 483
streamfile-skip vo, 478, 479
streamfile-skiplook vo, 479
streamfile-textread vo, 479
streamfile-textwrite vo, 480
streamfile-writeln vo, 479
strEmpty vv, 606
string assignment, 18, 123, 149, 252, 274, 279, 384, 564, 567, 570, 576–578, 582, 660, 663
string capacity, 9, 279, 384, 568–571, 574, 582, 671
string catenate symbol cs, 19, 191, 648, 651
string concatenation, 205, 206, 250, 278, 279, 570, 573
string conversion, 581, 582, 592, 593, 598
string length, 571, 593
string literal, 16, 20, 121, 123, 125, 149, 191, 205, 206, 250–253, 262, 271, 274, 278, 297, 304, 324, 617, 660, 661, 663, 667, 670, 671
string literal cs, 20, 247, 650, 651
string literal type, 121, 123, 191, 205, 250–253, 262, 271, 274, 278, 278, 279, 297, 304, 324, 660, 661, 667, 670
string module, 571
string parameter, 567
string terminator, 9, 123, 149, 206, 252, 274, 568–571, 573, 574, 579, 582, 663, 665, 671
string type, 123, 205, 252, 279, 324, 564, 662, 663, 670
string value, 9, 123, 206, 297, 384, 402, 421, 428, 436, 437, 467, 471, 510, 535, 554, 562, 564, 567, 569–575, 577–579, 582, 598, 605, 663, 665, 667, 670, 671
string variable, 123, 149, 567–573, 579, 663, 671
wf-string-constant-assignment vf, 123, 124

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

String-literal vt, 250, 658
String-literal-operation vt, 205, 658
m-string-literal-operation vo, 193, 206
t-string-literal-operation vf, 193, 205
wf-string-literal-operation vf, 192, 205
m-string-literal vo, 248, 251
t-string-literal vf, 248, 251
wf-string-literal vf, 248, 251
String-type vt, 310
String-value vt, 333
Strings lm, 9, 12, 564, 579
strint vf, 440, 584, 584, 595
strOutOfRange vv, 606
strreal vf, 445, 589, 589, 603
StrToCard vo, 585
StrToCard lp, 585, 585
StrToInt vo, 584
StrToInt lp, 583, 583
StrToReal vo, 588
StrToReal lp, 588, 588
structural identity, 104, 105, 283, 287, 307, 662
Structure vt, 309
structure component cs, 234, 235, 238, 649, 649
structure-of vf, 49, 114, 124, 152, 213, 230, 238, 239, 284, 287, 315, 315, 321–323, 325, 326, 329, 330, 365
a-structure vf, 114, 114
structured type, 7, 14, 54, 56, 57, 59, 60, 63, 71, 73, 77, 91–93, 95–97, 99–101, 104–108, 116, 121, 122, 213, 233, 235, 236, 238, 240, 243, 254, 266, 267, 280, 283, 287, 307, 321, 326, 328, 660, 664, 667, 670–672
is-structured-type vf, 321
Structures vt, 308
strWrongFormat vv, 606
SUBADR sp, 350, 353, 356
m-SUBADR vo, 354, 356
t-SUBADR vf, 354, 356
wf-SUBADR vf, 354, 356
subrange type, 91, 95, 96, 98, 107, 121, 122, 155, 224, 226–229, 266, 267, 272, 281, 286, 290, 298, 299, 304, 322, 325, 660, 664, 667, 669
subrange type cs, 93, 95, 642, 642
wf-subrange-assignment vf, 123, 123
Subrange-structure vt, 309
a-subrange-structure vf, 114, 115
Subrange-type vt, 96, 658
d-subrange-type vf, 94, 96
is-subrange-type vf, 123, 155, 322, 323, 325
wf-subrange-type vf, 94, 96
subscript, *see* array index
subseq vf, 569–572, 576, 577, 579
subset operator cs, 20, 190, 648, 651
m-subset vo, 211, 212
substring, 568–571, 574
subtract-offset vo, 357, 357
succ vf, 167, 168
succ lp, 409, 409
successor, 286
is-suffix vf, 38, 614
superset operator cs, 20, 190, 648, 651
m-superset vo, 211, 212
SuspendMe vo, 550
SuspendMe lp, 544, 548, 549, 549
SuspendMeAndActivate vo, 550
SuspendMeAndActivate lp, 548, 550, 550, 551
SWholeIO lm, 430, 431, 455–457
Switch vo, 551
Switch lp, 544, 548, 550, 550, 551, 555
switch-protection vo, 51, 344
swop-exception-context vo, 131, 136, 138, 140, 144, 145, 342, 342
symbol cs, 17
symbol with alternatives cs, 18
symmetric set difference operator cs, 19, 191, 648, 651
synthesize lp, 410, 410
SysClock lm, 607
SYSTEM sm, 1, 11, 12, 15, 70, 98, 99, 170, 189, 217, 253, 259, 260, 264, 265, 279, 284, 288, 324, 347, 348, 350–353, 355, 356, 358, 662, 666, 671, 672
system clock, 607–609
system module, xv, 1, 2, 4, 4, 5, 6, 8, 9, 11, 12, 23–27, 29, 32, 40, 42–44, 70, 99, 217, 261, 265, 270, 271, 347, 348, 368, 380, 389, 392, 395, 661, 665, 666, 671, 676
System-basic-type vt, 309
m-system-basic-type-eq vo, 218, 218
m-system-basic-type-ne vo, 218, 218
System-basic-type-relational-operation vt, 217, 658
m-system-basic-type-relational-operation vo, 207, 218
t-system-basic-type-relational-operation vf, 206, 218
wf-system-basic-type-relational-operation vf, 206, 217
System-basic-type-relational-operator vt, 218, 658
m-system-basic-type-relational-operator vo, 218, 218
System-environment vt, 312
SYSTEM-function-designator vt, 353, 657
System-function-designator vt, 348, 658
m-SYSTEM-function-designator vo, 348, 354
m-system-function-designator vo, 230, 348
t-SYSTEM-function-designator vf, 348, 354
t-system-function-designator vf, 230, 348
wf-SYSTEM-function-designator vf, 348, 354
wf-system-function-designator vf, 230, 348
System-function-procedure-typed vt, 312
System-procedure-designator vt, 347, 658
m-system-procedure-designator vo, 153, 347
wf-system-procedure-designator vf, 152, 347
System-proper-procedure-typed vt, 312
is-system-storage-formal-type vf, 264, 265
is-system-storage-parameter-compatible vf, 263, 264
is-system-storage-parameter vf, 263, 264, 267, 268
is-system-storage-type vf, 265, 324
a-system-storage vf, 114, 120

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

System-value vt, 334
systems programming, xv

t vf, 611
Table-id vt, 536
table-id vo, 436–442, 445–448, 450, 460, 464–470, 477, 478, 489, 490, 501–504, 514, 515, 534, 535, 539
tag field, 107, 109–112, 117–119, 149–151, 183, 185–187, 227, 238–240, 365–367
tag field cs, 110, 110, 644, 644
tag identifier cs, 110, 110, 644, 644
tag type, 110, 111
tag type cs, 110, 110, 644, 644
tag-tagged-variable vf, 118, 118
Tag-variable vt, 313
tag-variable-assignment vo, 150, 151
tag-variant-variable vf, 119
Tagged-variable vt, 313
tagged-variable-assignment vo, 150, 151
tagged-variant vo, 151, 151
is-tags-in-field-list vf, 284, 288, 365, 366, 366, 367
is-tags-in-fields vf, 366, 366
is-tags-in-variant-fields vf, 366, 366
is-tags-in-variant vf, 367, 367
tan vo, 418
`tan` lp, 417, 418, 418, 426, 426
target-for-interrupted-coroutine vo, 343, 372
term cs, 189, 189, 648, 648
term operator cs, 189, 190, 648, 648
term-flush vo, 518, 522
term-get vo, 520, 522
term-put vo, 516–518, 520, 521, 522
`TermFile` lm, 431, 511, 515, 516, 518–520, 522
termfile-check vo, 513, 514
termfile-check-file vo, 513, 514
termfile-Close vo, 514
termfile-dens vv, 519
termfile-flags vv, 513
termfile-flush vo, 518, 519
termfile-free vo, 515, 519
termfile-getname vo, 518
termfile-look vo, 515, 516
termfile-merge-flags vf, 513, 513
termfile-Open vo, 513
termfile-ops vv, 519
termfile-rawread vo, 517
termfile-rawwrite vo, 518
termfile-reset vo, 518
termfile-skip vo, 515, 516
termfile-skiplook vo, 516
termfile-textread vo, 516
termfile-textwrite vo, 517
termfile-writeln vo, 516
is-terminating vo, 41, 340
`TERMINATION` sm, 40, 347, 389
termination vo, 26, 41, 341, 375
Termination-actions vt, 334
termination-actions vo, 26, 341, 341, 341
TERMINATION-function-designator vt, 389, 658
m-TERMINATION-function-designator vo, 348, 390
t-TERMINATION-function-designator vf, 348, 390
wf-TERMINATION-function-designator vf, 348, 390
Termination-stack vt, 334
terminator vv, 606
text I/O operations, 430, 434, 435, 455, 464–466, 503, 512
a-text-item vo, 478, 479, 491, 492, 504–506, 520, 521, 526, 529
textFlag vv, 474
`TextIO` lm, 12, 430, 434, 435, 455, 456, 582
textParseError vv, 470
TextRead vo, 466
`TextRead` lp, 436, 438, 465, 465, 466, 479, 492, 506, 516, 524
TextWrite vo, 466
`TextWrite` lp, 466, 466, 480, 493, 506, 517
tf-add-queue vo, 515–517, 520, 520
tf-clear-queue vo, 518, 521
tf-dec-opened vo, 519, 521
tf-get-opened vo, 519, 521
tf-getinput vo, 520, 520
tf-have-queue vo, 517, 521
tf-inc-opened vo, 514, 521
tf-inspect-queue vo, 515, 516, 520
tf-read-chars-up-to vo, 516, 520
tf-read-up-to vo, 517, 521
tf-skip-queue vo, 516, 520
is-threatened-as-argument vf, 172, 175, 176
is-threatened-in-assignment-statement vf, 172, 172
is-threatened-in-case-statement vf, 172, 173
is-threatened-in-expression vf, 172–174, 174, 174–176
is-threatened-in-for-statement vf, 172, 174
is-threatened-in-function-call vf, 174, 175
is-threatened-in-function-procedure-block vf, 171, 171
is-threatened-in-function-procedure vf, 170, 171
is-threatened-in-if-statement vf, 172, 173
is-threatened-in-infix-expression vf, 174, 175
is-threatened-in-loop-statement vf, 172, 174
is-threatened-in-prefix-expression vf, 174, 175
is-threatened-in-procedure-call vf, 172, 172
is-threatened-in-procedure vf, 170, 170, 171
is-threatened-in-proper-procedure-block vf, 171, 171
is-threatened-in-proper-procedure vf, 170, 171
is-threatened-in-repeat-statement vf, 172, 173
is-threatened-in-return-statement vf, 172, 172
is-threatened-in-set-constructor vf, 175, 175
is-threatened-in-statement-seq vf, 166, 171, 171, 173, 174
is-threatened-in-statement vf, 171, 171
is-threatened-in-structured-value vf, 175, 175
is-threatened-in-value-constructor vf, 174, 175
is-threatened-in-value-designator vf, 172, 174, 175, 175
is-threatened-in-variable-designator vf, 172, 173, 174, 174, 176

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

is-threatened-in-while-statement vf, 172, 173
is-threatened-in-with-statement vf, 172, 173
threatening, 168–170, 375, 671
to-be-declared vf, 67, 320
to-char vf, 150, 155, 210, 253, 253
to-func-type vf, 60, 61, 78
to-proc-type vf, 59, 59, 75
token cs, 16
tooManyOpen vv, 475
top-exception-context vo, 134, 342, 384, 386, 387, 395
TRANSFER sp, 369, 370, 373, 376, 662, 666
transfer-to-handler vo, 343, 373
m-TRANSFER vo, 371, 376
wf-TRANSFER vf, 370, 376
TRUE pi, 274, 274
TRUNC pi, 288, 303, 303, 304
trunc vf, 306, 306
trunc lp, 410, 410
m-trunc-designator vo, 290, 304
t-trunc-designator vf, 289, 303
wf-trunc-designator vf, 289, 303
TSIZE sp, 253, 260, 284, 288, 350, 352, 353, 365, 670
tsize vf, 269, 284, 288, 303, 360, 366, 366
m-TSIZE vo, 354, 366
t-TSIZE vf, 354, 366
wf-TSIZE vf, 354, 365
Type vt, 91, 658
type declaration cs, 56, 64, 68, 640, 641, 641
type definition cs, 54, 56, 640, 640
type denoter cs, 68, 69, 91, 99, 106, 109, 641, 642, 643, 644
type identifie cs, 60, 641
type identifier cs, 91, 92, 92, 104, 232, 236, 242, 260, 642, 642, 643, 649, 650
type parameter cs, 258, 260, 650, 650
type transfer, *see* CAST
wf-type-and-set-definition vf, 236, 242, 243
Type-declaration vt, 68, 658
d-type-declaration vf, 57, 64, 69, 69
m-type-declaration vo, 57, 66, 69
wf-type-declaration vf, 57, 65, 69
Type-definition vt, 56, 658
d-type-definition vf, 55, 57
m-type-definition vo, 56, 57
wf-type-definition vf, 56, 57
Type-identifier vt, 92, 658
d-type-identifier vf, 78, 92, 93, 105
wf-type-identifier vf, 60, 92, 93, 106
Type-name vt, 309
type-of vf, 93, 96, 110, 111, 232, 237, 242, 261, 298–300, 314, 364
type-of-constant vf, 224, 314
type-of-procedure vf, 224, 316
type-of-set vf, 243, 244, 244
type-of-variable-or-field vf, 183, 224, 315
Type-parameter vt, 260, 658
m-type-parameter vo, 259, 261
wf-type-parameter vf, 259, 261
Typed vt, 308
d-type vf, 69, 70, 92, 97, 99, 100, 102, 107, 109
Typed-expression vt, 260, 659
m-typed-expression vo, 259, 260
wf-typed-expression vf, 259, 260
Typed-type vt, 261
Typed-value vt, 260
Typed-variable vt, 259
Typed-variable-designator vt, 259, 659
m-typed-variable-designator vo, 259, 260
wf-typed-variable-designator vf, 259, 259
a-typed vf, 114, 117–119, 337, 403
is-type vf, 49, 55, 65, 93, 261, 298, 299, 303, 314, 364, 365
Types vt, 308
wf-type vf, 69, 70, 92, 98–100, 102, 107, 109

ulp lp, 409, 409
undefined, 38, 149, 166, 167, 189, 194, 197, 199, 221, 224, 228, 240, 579
UNDEFINED-VALUE ex, 224
underscore, *see* lowline
UNINTERRUPTIBLE pi, 17, 50, 280, 280, 375
is-uniques vf, 95, 113, 537, 614
unit type, 325
UnMakeChan vo, 534
UnMakeChan lp, 531, 534, 534
unqualified vf, 53, 53
unqualified export cs, 46, 47, 640, 640
unqualified import cs, 43, 44, 639, 639
Unqualified-environment vt, 312
Unqualified-export vt, 47, 659
d-unqualified-export vf, 46, 47
wf-unqualified-export vf, 46, 47
Unqualified-import vt, 44, 659
d-unqualified-import vf, 43, 45
wf-unqualified-import vf, 43, 45
unqualify vf, 53, 313–317, 319
unsigned type, 92, 95, 122, 124, 220, 221, 245, 271, 271, 272, 275, 279, 283, 287, 290, 294, 295, 297, 301, 303, 304, 323, 325, 355, 356, 374, 383, 385, 660, 661, 664, 669
wf-unsigned-assignment vf, 123, 124
is-unsigned-type vf, 323, 323
up-to vf, 481, 493, 507, 521, 525, 530
upper case, 413
upper-bound-of vf, 124, 328, 328
UrgencyOf vo, 553
UrgencyOf lp, 553, 553
User-exception vt, 334
UTC, *see* Universal Coordinated Time
UTCDiff vt, 610

VAL pi, 260, 288, 292, 294, 296, 298, 301, 304, 304, 305, 662, 672
m-val-designator vo, 290, 305

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

t-val-designator vf, 289, 305
wf-val-designator vf, 289, 305
val-enumeration vf, 330, 401, 402, 469, 471, 535, 553, 554
valid vv, 606
is-valid-date vf, 609, 610, 610
Valid-sid vt, 562
Value vt, 333
value constructor cs, 189, 231, 648, 649
value designator cs, 152, 156, 189, 222, 225, 227–229, 645, 646, 648, 648, 649
value formal type cs, 102, 104, 643, 643
value of address type cs, 69, 69, 641, 641
value parameter, 9, 34, 61, 62, 72, 102, 258, 260–262, 264–266, 279, 283, 287, 355, 356, 361, 363, 374, 383, 566, 667, 669
value parameter specification cs, 61, 62, 641, 641
Value-constructor vt, 231, 659
m-value-constructor vo, 190, 231
t-value-constructor vf, 190, 231
wf-value-constructor vf, 190, 231
Value-designator vt, 223, 659
m-value-designator vo, 153, 157, 190, 223, 226–228, 230, 284, 288
t-value-designator vf, 152, 157, 173, 190, 223, 226–228, 230, 284, 288
wf-value-designator vf, 152, 157, 190, 223, 226–228, 230, 284, 288
Value-formal-type vt, 104, 659
Value-formal-typed vt, 310
d-value-formal-type vf, 103, 104
wf-value-formal-type vf, 103, 104
value-of-a-variable vo, 188, 224, 224, 228, 401, 453, 454, 551, 560, 579
is-value-parameter-compatible vf, 262, 262, 355, 356, 362, 363, 374, 383
Value-parameter-specification vt, 62, 659
d-value-parameter-specification vf, 61, 62
wf-value-parameter-specification vf, 62, 62
Value-store vt, 332
ValueCard vo, 596
`ValueCard` lp, 596, 596
ValueInt vo, 595
`ValueInt` lp, 595, 595
ValueReal vo, 603
`ValueReal` lp, 603, 603
values-of vf, 97, 114–116, 119, 155, 236, 244, 285, 287, 306, 329, 330, 362, 363
values-of-basic-type vf, 277, 330
values-of-enumeration vf, 330, 330
values-of-subrange vf, 330, 330
Variable vt, 312
variable declaration cs, 54, 64, 69, 640, 641, 641
variable designator cs, 148, 156, 182, 184, 185, 187, 258, 645, 646, 647, 647, 650
variable formal type cs, 102, 103, 643, 643
variable identifier list cs, 69, 69, 641, 641
variable parameter, 34, 61–63, 72, 102, 170, 258, 259, 261–264, 266–268, 283, 287, 353, 400, 661, 671
variable parameter specification cs, 61, 62, 641, 641
Variable-declaration vt, 70, 659
d-variable-declaration vf, 55, 64, 70, 71
m-variable-declaration vo, 56, 66, 71
wf-variable-declaration vf, 56, 65, 70
Variable-designator vt, 182, 659
m-variable-designator vo, 149, 157, 182, 185, 186, 188, 260
t-variable-designator vf, 149, 157, 173, 182, 184, 186, 187, 259, 282, 284, 285, 287, 288, 303, 360, 372, 374, 376, 384
wf-variable-designator vf, 149, 157, 182, 184, 186, 187, 259
Variable-formal-type vt, 103, 659
Variable-formal-typed vt, 310
d-variable-formal-type vf, 103, 103
wf-variable-formal-type vf, 103, 103
Variable-identifier vt, 70, 659
is-variable-or-field vf, 183, 224, 315
is-variable-parameter-compatible vf, 262, 263, 264
Variable-parameter-specification vt, 63, 659
d-variable-parameter-specification vf, 61, 63
wf-variable-parameter-specification vf, 62, 63
Variable-typed vt, 310
variable-value vo, 151, 186, 224, 224, 225, 268, 282, 285, 287
Variables vt, 308
Variant vt, 112, 659
variant, 84, 90, 107, 109–113, 117–119, 149–151, 183, 186, 187, 224, 227, 238–241, 303, 327, 365–367, 637, 638
variant cs, 110, 112, 644, 644
variant else part cs, 110, 110, 644, 644
variant fields cs, 107, 110, 644, 644
variant label cs, 112, 112, 644, 644
variant label list cs, 112, 112, 644, 644
variant list cs, 110, 110, 644, 644
Variant-component vt, 313
Variant-fields vt, 110, 659
Variant-fields-structure vt, 310
a-variant-fields-structure vf, 117, 118
d-variant-fields vf, 108, 110
wf-variant-fields vf, 109, 111
Variant-structure vt, 310
a-variant-structure vf, 119, 119
Variant-variable vt, 313
variant-variable-assignment vo, 150, 151
d-variant vf, 110, 112
a-variants vf, 118, 119, 119
wf-variant vf, 111, 112
vet vo, 478, 489, 490, 502–504, 515, 529
visibility of an identifier, 7, 23, 44, 45, 52, 53, 66, 67, 79, 283, 287, 663, 665, 668, 669, 673
visible-identifiers vf, 32, 35, 44, 45, 47, 48, 53, 55, 65, 319, 319, 319

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

W vo, 594
Wait vo, 551
Wait lp, 548, 550, 551, 551
WE vo, 602
while statement cs, 146, 162, 645, 646
While-statement vt, 162, 659
m-while-statement vo, 147, 162, 162
wf-while-statement vf, 146, 162
white space cs, 21
white space control cs, 21
whoami vo, 549–553, 557, 563
whole number literal cs, 20, 247, 650, 651
m-whole-add vo, 200, 201
m-whole-divide vo, 200, 201
m-whole-div vo, 200, 201
m-whole-mod vo, 200, 201
m-whole-multiply vo, 200, 201
WHOLE-NONPOS-DIV ex, 201
WHOLE-NONPOS-MOD ex, 201
is-whole-number-compatible vf, 121, 122
Whole-number-literal vt, 248, 659
m-whole-number-literal vo, 248, 249
t-whole-number-literal vf, 248, 249
wf-whole-number-literal vf, 248, 249
Whole-number-operation vt, 198, 659
m-whole-number-operation vo, 193, 199
t-whole-number-operation vf, 193, 199
wf-whole-number-operation vf, 192, 199
Whole-number-operator vt, 199, 659
m-whole-number-operator vo, 199, 200
whole-number-result-type vf, 199, 199, 246
Whole-number-type vt, 308
is-whole-number-type vf, 115, 122, 155, 166, 235, 246, 292, 300, 305, 306, 322, 323, 324
WHOLE-OVERFLOW ex, 199
m-whole-rem vo, 200, 201
m-whole-subtract vo, 200, 201
WHOLE-ZERO-DIVISION ex, 201
WHOLE-ZERO-REMAINDER ex, 201
WholeConv lm, 581, 582, 592, 595–598
wholeconv-exception vo, 595, 597, 597
wholeconv-exception-string vo, 597, 598
wholeconv-initialization vo, 598
Wholeconvexception vt, 597
WholeIO lm, 430, 434, 439, 456, 457, 581
WholeStr lm, 581, 582
Wirth, xv
with statement cs, 146, 156, 645, 646
With-statement vt, 156, 659
m-with-statement vo, 147, 157
wf-with-statement vf, 146, 156
word token cs, 16
word-array-structure vv, 353
word-range-structure vv, 353
Write vo, 448
Write lp, 448, 448
write-chars vo, 438, 439, 441, 442, 446, 447, 453
write-eol vo, 438, 452
write-external-IOLib-state vo, 537
write-external-program-state vo, 331, 341
write-external-ProgramArgs-state vo, 528
write-file vo, 482, 495, 509, 537
write-locs vo, 448, 454
WriteCard vo, 442
WriteCard lp, 442, 442
WriteChar vo, 438
WriteChar lp, 438, 438
WriteEng vo, 446
WriteEng lp, 446, 446
WriteFixed vo, 447
WriteFixed lp, 446, 446
writeFlag vv, 474
WriteFloat vo, 445
WriteFloat lp, 445, 445, 445
WriteInt vo, 441
WriteInt lp, 441, 441
WriteLn vo, 438
WriteLn lp, 438, 438, 465, 465, 465, 479, 492, 505, 516
WriteReal vo, 447
WriteReal lp, 447, 447
WriteString vo, 439
WriteString lp, 438, 439
WRONG-TYPE ex, 188, 228
wrongDevice vv, 470
wrongFileType vv, 475
wrongFlags vv, 475
wrongFormat vv, 449
wrongNameFormat vv, 475
wrongPermissions vv, 475
wrongStorageToUnallocate vv, 402

x-closure vf, 29, 30, 43–45, 47, 48, 49
x-closure-typed vf, 49, 49
x-statements-of-case-statement vf, 180, 180
x-statements-of-for-statement vf, 180, 181
x-statements-of-if-statement vf, 180, 180
x-statements-of-loop-statement vf, 180, 181
x-statements-of-repeat-statement vf, 180, 181
x-statements-of-statement vf, 179, 179
x-statements-of-statement-seq vf, 170, 177, 179, 180, 181
x-statements-of-while-statement vf, 180, 180
x-statements-of-with-statement vf, 180, 180

is-zero-array-type vf, 265, 328
is-ZZ-type vf, 323, 323

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

© ISO/IEC

ICS 35.060

Descriptors: data processing, computer software, artificial languages, programming languages, Modula-2.

Price based on 707 pages

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale