INTERNATIONAL STANDARD

ISO/IEC 15291

FDIS
1999-01-11

Information technology — Programming languages — Ada Semantic Interface Specification (ASIS)

Technologies de l'information — Langages de programmation, leurs environnements et interfaces de logiciel de système — Spécification d'interface pour la sémantique Ada

Reference number
ISO/IEC 15291:1998(E) FDIS

Contents

FOREWORD XIII

INTRODUCTION XIV

1 GENERAL 1

1.1 Scope 1
1.1.1 Extent 1
1.1.2 Structure 2
1.1.3 Conformity with this International Standard 3
1.1.3.1 Implementation conformance requirements 3
1.1.3.2 Implementation conformance documentation 4
1.1.3.3 Implementation conformance categories 4
1.1.3.4 Application conformance categories 5
1.1.4 Implementation permissions 6
1.1.4.1 Traditional approach (permission 1) 6
1.1.4.2 Client / server approach (permission 2) 6
1.1.4.3 Distributed traditional approach (permission 3) 6
1.1.4.4 ASIS dynamic client approach (permission 4) 6
1.1.5 Classification of errors 8
1.2 Normative reference 9
1.3 Terms and definitions 9

2 ASIS TECHNICAL CONCEPTS 10

2.1 Ada compilation environment 10
2.1.1 Ada environment 10
2.1.2 ASIS notion of the Ada compilation environment 11
2.1.3 Illegal / inconsistent units in the compilation environment 12
2.2 ASIS queries 12
2.2.1 Structural queries 12
2.2.2 Semantic queries 12
2.2.3 General ASIS query processing 12
2.2.3.1 Elements and element kinds 12
2.2.3.2 Processing specific constructs 14
2.2.3.3 Element list processing 14
2.2.3.4 Operations that apply to all elements 15
2.2.3.5 Semantic references 16
2.3 ASIS package architecture 18
2.4 Application use 20
2.4.1 Establishing ASIS context 20
2.4.2 Required sequencing of calls 21
2.4.3 Notional ASIS application 21
2.4.4 Erroneous applications 24
2.4.5 Usage rules 24
2.4.5.1 General usage rules 24
2.4.5.2 Rules for processing queries for illegal/inconsistent context 25

3 PACKAGE ASIS 27
3.1 type ASIS_Integer 28
3.2 type ASIS_Natural 28
3.3 type ASIS_Positive 28
3.4 type List_Index 28
3.5 type Context 28
3.6 type Element 29
3.7 type Element_List 30
3.8 subtypes of Element and Element_List 30
3.9 Element Kinds 31
3.9.1 type Element_Kinds 31
3.9.2 type Pragma_Kinds 32
3.9.3 type Defining_Name_Kinds 33
3.9.4 type Declaration_Kinds 33
3.9.5 type Trait_Kinds 35
3.9.6 type Declaration_Origins 36
3.9.7 type Mode_Kinds 36
3.9.8 type Subprogram_Default_Kinds 36
3.9.9 type Definition_Kinds 36
3.9.10 type Type_Kinds 37
3.9.11 type Formal_Type_Kinds 37
3.9.12 type Access_Type_Kinds 38
3.9.13 type Root_Type_Kinds 38
3.9.14 type Constraint_Kinds 39
3.9.15 type Discrete_Range_Kinds 39
3.9.16 type Association_Kinds 39
3.9.17 type Expression_Kinds 39
3.9.18 type Operator_Kinds 40
3.9.19 type Attribute_Kinds 41
3.9.20 type Statement_Kinds 42
3.9.21 type Path_Kinds 43
3.9.22 type Clause_Kinds 44
3.9.23 type Representation_Clause_Kinds 44
3.10 type Compilation_Unit 44
3.11 type Compilation_Unit_List 45
3.12 Unit Kinds 45
3.12.1 type Unit_Kinds 45
3.12.2 type Unit_Classes 47
3.12.3 type Unit_Origins 47
3.12.4 type Relation_Kinds 47
3.13 type Traverse_Control 49
3.14 type Program_Text 49

4 PACKAGE ASIS.ERRORS 51
4.1 type Error_Kinds 51

5 PACKAGE ASIS.EXCEPTIONS 52

6 PACKAGE ASIS.IMPLEMENTATION 53
6.1 function ASIS_Version 53
6.2 function ASIS_Implementor 53
6.3 function ASIS_Implementor_Version 53
6.4 function ASIS_Implementor_Information 53
6.5 function Is_Initialized 53
6.6 procedure Initialize 53
6.7 function Is_Finalized 54
6.8 procedure Finalize 54
6.9 function Status 54

6.10 function Diagnosis 54
6.11 procedure Set_Status 55

7 PACKAGE ASIS.IMPLEMENTATION.PERMISSIONS................................56

7.1 function Is_Formal_Parameter_Named_Notation_Supported 56
7.2 function Default_In_Mode_Supported 56
7.3 function Generic_Actual_Part_Normalized 56
7.4 function Record_Component_Associations_Normalized 56
7.5 function Is_Prefix_Call_Supported 57
7.6 function Function_Call_Parameters_Normalized 57
7.7 function Call_Statement_Parameters_Normalized 57
7.8 function Discriminant_Associations_Normalized 57
7.9 function Is_Line_Number_Supported 58
7.10 function Is_Span_Column_Position_Supported 58
7.11 function Is_Commentary_Supported 58
7.12 function Attributes_Are_Supported 58
7.13 function Implicit_Components_Supported 58
7.14 function Object_Declarations_Normalized 58
7.15 function Predefined_Operations_Supported 59
7.16 function Inherited_Declarations_Supported 59
7.17 function Inherited_Subprograms_Supported 59
7.18 function Generic_Macro_Expansion_Supported 59

8 PACKAGE ASIS.ADA_ENVIRONMENTS...60

8.1 function Default_Name 60
8.2 function Default_Parameters 60
8.3 procedure Associate 60
8.4 procedure Open 61
8.5 procedure Close 61
8.6 procedure Dissociate 61
8.7 function Is_Equal 62
8.8 function Is_Identical 62
8.9 function Exists 63
8.10 function Is_Open 63
8.11 function Has_Associations 63
8.12 function Name 63
8.13 function Parameter 64
8.14 function Debug_Image 64

9 PACKAGE ASIS.ADA_ENVIRONMENTS.CONTAINERS...........................65

9.1 type Container 65
9.2 type Container_List 65
9.3 function Defining_Containers 65
9.4 function Enclosing_Context 66
9.5 function Library_Unit_Declaration 66
9.6 function Compilation_Unit_Bodies 67
9.7 function Compilation_Units 67
9.8 function Is_Equal 67
9.9 function Is_Identical 68
9.10 function Name 68

10 PACKAGE ASIS.COMPILATION_UNITS..69

10.1 function Unit_Kind 69
10.2 function Unit_Class 69
10.3 function Unit_Origin 70
10.4 function Enclosing_Context 70

10.5 function Enclosing_Container 70
10.6 function Library_Unit_Declaration 71
10.7 function Compilation_Unit_Body 71
10.8 function Library_Unit_Declarations 71
10.9 function Compilation_Unit_Bodies 72
10.10 function Compilation_Units 72
10.11 function Corresponding_Children 72
10.12 function Corresponding_Parent_Declaration 73
10.13 function Corresponding_Declaration 74
10.14 function Corresponding_Body 76
10.15 function Is_Nil 77
10.16 function Is_Nil 77
10.17 function Is_Equal 77
10.18 function Is_Identical 78
10.19 function Unit_Full_Name 78
10.20 function Unique_Name 78
10.21 function Exist 79
10.22 function Can_Be_Main_Program 79
10.23 function Is_Body_Required 79
10.24 function Text_Name 79
10.25 function Text_Form 80
10.26 function Object_Name 80
10.27 function Object_Form 80
10.28 function Compilation_Command_Line_Options 81
10.29 function Has_Attribute 81
10.30 function Attribute_Value_Delimiter 81
10.31 function Attribute_Values 81
10.32 function Subunits 82
10.33 function Corresponding_Subunit_Parent_Body 82
10.34 function Debug_Image 83

11 PACKAGE ASIS.COMPILATION_UNITS.TIMES 84

11.1 type Time 84
11.2 function Time_Of_Last_Update 84
11.3 function Compilation_CPU_Duration 84
11.4 function Attribute_Time 85

12 PACKAGE ASIS.COMPILATION_UNITS.RELATIONS 86

12.1 type Relationship 86
12.2 constant Nil_Relationship 88
12.3 function Semantic_Dependence_Order 88
12.4 function Elaboration_Order 89

13 PACKAGE ASIS.ELEMENTS 91

13.1 function Unit_Declaration 91
13.2 function Enclosing_Compilation_Unit 91
13.3 function Context_Clause_Elements 92
13.4 function Configuration_Pragmas 92
13.5 function Compilation_Pragmas 93
13.6 function Element_Kind 94
13.7 function Pragma_Kind 94
13.8 function Defining_Name_Kind 95
13.9 function Declaration_Kind 95
13.10 function Trait_Kind 95
13.11 function Declaration_Origin 96
13.12 function Mode_Kind 96

13.13 function Default_Kind 96
13.14 function Definition_Kind 97
13.15 function Type_Kind 97
13.16 function Formal_Type_Kind 97
13.17 function Access_Type_Kind 97
13.18 function Root_Type_Kind 98
13.19 function Constraint_Kind 98
13.20 function Discrete_Range_Kind 98
13.21 function Expression_Kind 99
13.22 function Operator_Kind 99
13.23 function Attribute_Kind 99
13.24 function Association_Kind 99
13.25 function Statement_Kind 100
13.26 function Path_Kind 100
13.27 function Clause_Kind 100
13.28 function Representation_Clause_Kind 100
13.29 function Is_Nil 101
13.30 function Is_Nil 101
13.31 function Is_Equal 101
13.32 function Is_Identical 101
13.33 function Is_Part_Of_Implicit 102
13.34 function Is_Part_Of_Inherited 103
13.35 function Is_Part_Of_Instance 103
13.36 function Enclosing_Element 103
13.37 function Pragmas 105
13.38 function Corresponding_Pragmas 106
13.39 function Pragma_Name_Image 106
13.40 function Pragma_Argument_Associations 106
13.41 function Debug_Image 107
13.42 function Hash 107

14 PACKAGE ASIS.ITERATOR 108

14.1 procedure Traverse_Element 108

15 PACKAGE ASIS.DECLARATIONS 110

15.1 function Names 110
15.2 function Defining_Name_Image 111
15.3 function Position_Number_Image 111
15.4 function Representation_Value_Image 112
15.5 function Defining_Prefix 112
15.6 function Defining_Selector 112
15.7 function Discriminant_Part 113
15.8 function Type_Declaration_View 113
15.9 function Object_Declaration_View 114
15.10 function Initialization_Expression 115
15.11 function Corresponding_Constant_Declaration 115
15.12 function Declaration_Subtype_Mark 116
15.13 function Corresponding_Type_Declaration 117
15.14 function Corresponding_First_Subtype 117
15.15 function Corresponding_Last_Constraint 118
15.16 function Corresponding_Last_Subtype 118
15.17 function Corresponding_Representation_Clauses 119
15.18 function Specification_Subtype_Definition 119
15.19 function Parameter_Profile 120
15.20 function Result_Profile 120
15.21 function Body_Declarative_Items 121

15.22 function Body_Statements 121
15.23 function Body_Exception_Handlers 122
15.24 function Body_Block_Statement 122
15.25 function Is_Name_Repeated 123
15.26 function Corresponding_Declaration 123
15.27 function Corresponding_Body 125
15.28 function Corresponding_Subprogram_Derivation 127
15.29 function Corresponding_Type 127
15.30 function Corresponding_Equality_Operator 128
15.31 function Visible_Part_Declarative_Items 128
15.32 function Is_Private_Present 129
15.33 function Private_Part_Declarative_Items 129
15.34 function Renamed_Entity 130
15.35 function Corresponding_Base_Entity 130
15.36 function Protected_Operation_Items 131
15.37 function Entry_Family_Definition 132
15.38 function Entry_Index_Specification 133
15.39 function Entry_Barrier 133
15.40 function Corresponding_Subunit 133
15.41 function Is_Subunit 134
15.42 function Corresponding_Body_Stub 134
15.43 function Generic_Formal_Part 135
15.44 function Generic_Unit_Name 137
15.45 function Generic_Actual_Part 137
15.46 function Formal_Subprogram_Default 138
15.47 function Corresponding_Generic_Element 139
15.48 function Is_Dispatching_Operation 139

16 PACKAGE ASIS.DEFINITIONS..140

16.1 function Corresponding_Type_Operators 140
16.2 function Parent_Subtype_Indication 141
16.3 function Record_Definition 141
16.4 function Implicit_Inherited_Declarations 141
16.5 function Implicit_Inherited_Subprograms 142
16.6 function Corresponding_Parent_Subtype 143
16.7 function Corresponding_Root_Type 143
16.8 function Corresponding_Type_Structure 144
16.9 function Enumeration_Literal_Declarations 144
16.10 function Integer_Constraint 145
16.11 function Mod_Static_Expression 145
16.12 function Digits_Expression 145
16.13 function Delta_Expression 146
16.14 function Real_Range_Constraint 146
16.15 function Index_Subtype_Definitions 147
16.16 function Discrete_Subtype_Definitions 147
16.17 function Array_Component_Definition 147
16.18 function Access_To_Object_Definition 148
16.19 function Access_To_Subprogram_Parameter_Profile 148
16.20 function Access_To_Function_Result_Profile 149
16.21 function Subtype_Mark 149
16.22 function Subtype_Constraint 150
16.23 function Lower_Bound 150
16.24 function Upper_Bound 151
16.25 function Range_Attribute 151
16.26 function Discrete_Ranges 152

16.27 function Discriminant_Associations 152
16.28 function Component_Subtype_Indication 153
16.29 function Discriminants 154
16.30 function Record_Components 154
16.31 function Implicit_Components 155
16.32 function Discriminant_Direct_Name 155
16.33 function Variants 156
16.34 function Variant_Choices 156
16.35 function Ancestor_Subtype_Indication 157
16.36 function Visible_Part_Items 157
16.37 function Private_Part_Items 157
16.38 function Is_Private_Present 158

17 PACKAGE ASIS.EXPRESSIONS 159

17.1 function Corresponding_Expression_Type 159
17.2 function Value_Image 160
17.3 function Name_Image 160
17.4 function References 161
17.5 function Is_Referenced 161
17.6 function Corresponding_Name_Definition 162
17.7 function Corresponding_Name_Definition_List 164
17.8 function Corresponding_Name_Declaration 164
17.9 function Prefix 165
17.10 function Index_Expressions 165
17.11 function Slice_Range 166
17.12 function Selector 166
17.13 function Attribute_Designator_Identifier 166
17.14 function Attribute_Designator_Expressions 167
17.15 function Record_Component_Associations 167
17.16 function Extension_Aggregate_Expression 168
17.17 function Array_Component_Associations 168
17.18 function Array_Component_Choices 169
17.19 function Record_Component_Choices 169
17.20 function Component_Expression 170
17.21 function Formal_Parameter 170
17.22 function Actual_Parameter 171
17.23 function Discriminant_Selector_Names 172
17.24 function Discriminant_Expression 173
17.25 function Is_Normalized 173
17.26 function Is_Defaulted_Association 174
17.27 function Expression_Parenthesized 174
17.28 function Is_Prefix_Call 175
17.29 function Corresponding_Called_Function 175
17.30 function Function_Call_Parameters 176
17.31 function Short_Circuit_Operation_Left_Expression 177
17.32 function Short_Circuit_Operation_Right_Expression 177
17.33 function Membership_Test_Expression 177
17.34 function Membership_Test_Range 178
17.35 function Membership_Test_Subtype_Mark 178
17.36 function Converted_Or_Qualified_Subtype_Mark 178
17.37 function Converted_Or_Qualified_Expression 179
17.38 function Allocator_Subtype_Indication 179
17.39 function Allocator_Qualified_Expression 179

18 PACKAGE ASIS.STATEMENTS 181

18.1 function Label_Names 181
18.2 function Assignment_Variable_Name 181
18.3 function Assignment_Expression 181
18.4 function Statement_Paths 182
18.5 function Condition_Expression 182
18.6 function Sequence_Of_Statements 182
18.7 function Case_Expression 183
18.8 function Case_Statement_Alternative_Choices 183
18.9 function Statement_Identifier 183
18.10 function Is_Name_Repeated 184
18.11 function While_Condition 184
18.12 function For_Loop_Parameter_Specification 184
18.13 function Loop_Statements 185
18.14 function Is_Declare_Block 185
18.15 function Block_Declarative_Items 185
18.16 function Block_Statements 186
18.17 function Block_Exception_Handlers 186
18.18 function Exit_Loop_Name 186
18.19 function Exit_Condition 187
18.20 function Corresponding_Loop_Exited 187
18.21 function Return_Expression 187
18.22 function Goto_Label 188
18.23 function Corresponding_Destination_Statement 188
18.24 function Called_Name 188
18.25 function Corresponding_Called_Entity 189
18.26 function Call_Statement_Parameters 190
18.27 function Accept_Entry_Index 191
18.28 function Accept_Entry_Direct_Name 191
18.29 function Accept_Parameters 191
18.30 function Accept_Body_Statements 192
18.31 function Accept_Body_Exception_Handlers 192
18.32 function Corresponding_Entry 192
18.33 function Requeue_Entry_Name 193
18.34 function Delay_Expression 193
18.35 function Guard 193
18.36 function Aborted_Tasks 194
18.37 function Choice_Parameter_Specification 194
18.38 function Exception_Choices 194
18.39 function Handler_Statements 195
18.40 function Raised_Exception 195
18.41 function Qualified_Expression 195
18.42 function Is_Dispatching_Call 196
18.43 function Is_Call_On_Dispatching_Operation 196

19 PACKAGE ASIS.CLAUSES 197

19.1 function Clause_Names 197
19.2 function Representation_Clause_Name 197
19.3 function Representation_Clause_Expression 198
19.4 function Mod_Clause_Expression 198
19.5 function Component_Clauses 198
19.6 function Component_Clause_Position 199
19.7 function Component_Clause_Range 199

20 PACKAGE ASIS.TEXT 200

20.1 type Line 200
20.2 type Line_Number 200
20.3 type Line_Number_Positive 201
20.4 type Line_List 201
20.5 type Character_Position 201
20.6 type Character_Position_Positive 201
20.7 type Span 201
20.8 function First_Line_Number 202
20.9 function Last_Line_Number 202
20.10 function Element_Span 202
20.11 function Compilation_Unit_Span 203
20.12 function Compilation_Span 203
20.13 function Is_Nil 203
20.14 function Is_Nil 203
20.15 function Is_Nil 204
20.16 function Is_Equal 204
20.17 function Is_Identical 204
20.18 function Length 204
20.19 function Lines 204
20.20 function Lines 205
20.21 function Lines 206
20.22 function Delimiter_Image 206
20.23 function Element_Image 206
20.24 function Line_Image 207
20.25 function Non_Comment_Image 207
20.26 function Comment_Image 207
20.27 function Is_Text_Available 208
20.28 function Debug_Image 208

21 PACKAGE ASIS.IDS 209

21.1 type Id 209
21.2 function Hash 209
21.3 function "<" 209
21.4 function ">" 209
21.5 function Is_Nil 210
21.6 function Is_Equal 210
21.7 function Create_Id 210
21.8 function Create_Element 210
21.9 function Debug_Image 211

22 PACKAGE ASIS.DATA_DECOMPOSITION (OPTIONAL) 212

22.1 type Record_Component 214
22.2 type Record_Component_List 214
22.3 type Array_Component 215
22.4 type Array_Component_List 215
22.5 type Dimension_Indexes 215
22.6 type Array_Component_Iterator 216
22.7 type Portable_Data 216
22.8 type Type_Model_Kinds 218
22.9 function Type_Model_Kind 218
22.10 function Is_Nil 218
22.11 function Is_Equal 218
22.12 function Is_Identical 219
22.13 function Is_Array 219

22.14 function Is_Record 219
22.15 function Done 220
22.16 procedure Next 220
22.17 procedure Reset 220
22.18 function Array_Index 220
22.19 function Array_Indexes 220
22.20 function Discriminant_Components 221
22.21 function Record_Components 221
22.22 function Record_Components 222
22.23 function Array_Components 223
22.24 function Array_Iterator 223
22.25 function Component_Data_Stream 224
22.26 function Component_Declaration 224
22.27 function Component_Indication 225
22.28 function All_Named_Components 225
22.29 function Array_Length 225
22.30 function Array_Length 226
22.31 function Size 226
22.32 function Size 226
22.33 function Position 227
22.34 function First_Bit 227
22.35 function Last_Bit 228
22.36 function Portable_Constrained_Subtype 228
22.37 function Construct_Artificial_Data_Stream 229

23 PACKAGE ASIS.DATA_DECOMPOSITION.PORTABLE_TRANSFER 231

23.1 generic package Portable_Constrained_Subtype 231
23.2 generic package Portable_Unconstrained_Record_Type 231
23.3 generic package Portable_Array_Type_1 232
23.4 generic package Portable_Array_Type_2 232
23.5 generic package Portable_Array_Type_3 232

ANNEXES

A GLOSSARY 234

B ASIS APPLICATION EXAMPLES 237

B.1 Use to traverse compilation unit 237
B.2 Use to build call tree 240

C MISCELLANEOUS ASIS I/O AND IDL APPROACHES 244

C.1 package Portable_Data_Io 244
C.2 package Asis.Ids.Id_Io 247
C.3 Implementation approach for IDL 250
C.3.1 ASIS API server 250
C.3.2 ASIS API client tool 251
C.3.3 Approach to implement the Traverse_Element generic 251
C.3.4 IDL to implement the Traverse_Element generic 252
C.3.5 Ada code output by the IDL compiler 253

D RATIONALE 254
D.1 Benefits of code analysis 254
D.1.1 Definition 254
D.1.2 Applicability 254
D.1.3 Motivation 255
D.2 Technology for code analysis 255
D.2.1 Code parsers 255
D.2.2 DIANA 256
D.2.3 LRM-interface 257
D.2.4 ASIS 257
D.2.5 Benefits of ASIS standard 259
D.3 Design considerations for ASIS 259
D.3.1 Design goals 259
D.3.2 Major changes from ASIS for ISO 8652:1987 261
D.3.3 Essence of Ada and ASIS 261
D.4 Major issues regarding ASIS 262
D.4.1 Ada environment and compilation units 262
D.4.2 ASIS context and inconsistency 262
D.4.3 Implicit declarations 264
D.4.4 Abstract "=" for private types 264
D.4.5 Usage names and expressions 265
D.4.6 Select alternative 265
D.4.7 Attribute definition clauses 265
D.4.8 Configuration pragmas 265
D.4.9 Queries with additional context parameter 266
D.4.10 Ids 266
D.4.11 Data decomposition 266
D.5 Conclusion 267
D.6 Acronyms 268

BIBLIOGRAPHY 269

INDEX 270

Figures

Figure 1 ASIS implementation permissions 7
Figure 2 ASIS as interface to Ada compilation environment 10
Figure 3 Application interface to ASIS Context 11
Figure 4 Syntactic tree representation of an Ada object declaration 13
Figure 5 Operations on elements 15
Figure 6 Semantic reference using corresponding queries 17
Figure 7 ASIS package architecture 18
Figure C.1 Generation of client/server ASIS artifacts 250

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 15291 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee 22, *Programming languages, their environments and system software interfaces*.

Annexes A, B, C, and D of this International Standard are for information only.

Introduction

The Ada Semantic Interface Specification (ASIS) is an interface between an Ada environment (as defined by ISO/IEC 8652:1995) and any tool requiring information from it. An Ada environment includes valuable semantic and syntactic information. ASIS is an open and published callable interface which gives CASE tool and application developers access to this information. ASIS has been designed to be independent of underlying Ada environment implementations, thus supporting portability of software engineering tools while relieving tool developers from needing to understand the complexities of an Ada environment's proprietary internal representation.

Examples of tools that benefit from the ASIS interface include: automated code monitors, browsers, call tree tools, code reformators, coding standards compliance tools, correctness verifiers, debuggers, dependency tree analysis tools, design tools, document generators, metrics tools, quality assessment tools, reverse engineering tools, re-engineering tools, style checkers, test tools, timing estimators, and translators.

The word "may" as used in this International Standard consistently means "is allowed to" (or "are allowed to"). It is used only to express permission, as in the commonly occurring phrase "an implementation may"; other words (such as "can," "could" or "might") are used to express ability, possibility, capacity, or consequentiality.

The ASIS interface consists of a set of types, subtypes, and subprograms which provide a capability to query the Ada compilation environment for syntactic and semantic information. Package **Asis** is the root of the ASIS interface. It contains common types used throughout the ASIS interface. Important common abstractions include Context, Element, and Compilation_Unit. Type Context helps identify the compilation units considered to be analyzable as part of the Ada compilation environment. Type Element is an abstraction of entities within a logical Ada syntax tree. Type Compilation_Unit is an abstraction for Ada compilation units. In addition, there are two sets of enumeration types called Element Kinds and Unit Kinds. Element Kinds are a set of enumeration types providing a mapping to the Ada syntax. Unit Kinds are a set of enumeration types describing the various kinds of compilation units.

All ASIS subprogram interfaces are provided using child packages. Some child packages also contain type and subtype interfaces local to the child package.

The child package Asis.Implementation provides queries to initialize, finalize, and query the error status of the ASIS implementation. The child package Asis.Ada_Environments encapsulates a set of queries that map physical Ada compilation and program execution environments to logical ASIS environments.

The child package Asis.Compilation_Units defines queries that deal with compilation units and serves as the gateway between Compilation_Units, Elements, and Ada_Environments. The child package Asis.Compilation_Units.Times encapsulates the time related functions used within ASIS. The child package Asis.Compilation_Units.Relations encapsulates semantic relationship concepts used in ASIS.

The child package Asis.Elements defines general Element queries and queries for pragmas. It provides information on the element kinds for further semantic analysis.

The child package Asis.Iterator provides a mechanism to perform an iterative traversal of a logical syntax tree. During the syntax tree traversal, ASIS can analyze the various elements contained within the syntax tree. ASIS can provide the application additional processing via generic procedures, which are instantiated by the application. These additional processing queries decompose as ASIS elements from the logical Ada semantic tree. Queries are provided in the child packages: Clauses, Declarations, Definitions, Expressions, and Statements.

- child package Asis.Clauses - Defines queries dealing with context clauses and representation clauses.
- child package Asis.Declarations - Defines queries dealing with Ada declarations.
- child package Asis.Definitions - Defines queries dealing with the definition portion of Ada object, type, and subtype declarations.
- child package Asis.Expressions - Defines all queries dealing with Ada expressions.
- child package Asis.Statements - Defines queries dealing with Ada statements.

The child package Asis.Text encapsulates a set of operations to access the text of ASIS elements. It defines the operations for obtaining compilation text spans, lines, and images of elements.

The child package Asis.Ids provides a mechanism to efficiently reference ASIS elements in a persistent manner.

To support portability amongst a variety of implementors' compilation environments, certain types and constants have been identified as implementation-defined.

The child package Asis.Errors defines the kinds of errors. The exceptions that can be raised across the ASIS interface are defined in the child package Asis.Exceptions.

The interface supports one optional child package and its single child package:

- child package Asis.Data_Decomposition - The interface also includes an optional capability to decompose data values using the ASIS type information and portable data stream, representing a data value of that type.

Information technology — Programming languages — Ada Semantic Interface Specification (ASIS)

1 General

1.1 Scope

The Ada Semantic Interface Specification (ASIS) is an interface between an Ada environment (as defined by ISO/IEC 8652:1995) and any tool requiring information from this environment. An Ada environment includes valuable semantic and syntactic information. ASIS is an open and published callable interface which gives CASE tool and application developers access to this information. ASIS has been designed to be independent of underlying Ada environment implementations, thus supporting portability of software engineering tools while relieving tool developers from needing to understand the complexities of an Ada environment's proprietary internal representation.

Examples of tools that benefit from the ASIS interface include: automated code monitors, browsers, call tree tools, code reformators, coding standards compliance tools, correctness verifiers, debuggers, dependency tree analysis tools, design tools, document generators, metrics tools, quality assessment tools, reverse engineering tools, re-engineering tools, safety and security tools, style checkers, test tools, timing estimators, and translators.

This International Standard specifies the form and meaning of the ASIS interface to the Ada compilation environment.

This International Standard is applicable to tools and applications needing syntactic and semantic information in the Ada compilation environment.

1.1.1 Extent

This International Standard specifies:

- The form of the ASIS interface;
- Sequencing of ASIS calls;
- The permissible variations within this International Standard, and the manner in which they are to be documented;
- Those violations of this International Standard that a conforming implementation is required to detect, and the effect of attempting to execute a program containing such violations;

This International Standard does not specify:

- Semantics of the interface in the face of simultaneous updates to the Ada compilation environment.
- Semantics of the interface for more than one thread of control.

1.1.2 Structure

This International Standard contains twenty-three clauses and four annexes.

Clause 1 is general in nature providing the scope of this International Standard, normative references, and definitions.

Clause 2 identifies the ASIS technical concepts. Here the Ada compilation environment to which ASIS interfaces is described. The concept of queries is presented. The ASIS package architecture is presented.

The packages that comprise the ASIS International Standard are provided in Clauses 3 through 23. These packages are provided in the correct compilation order and when presented in electronic format are compilable.

- Clause 3 package Asis
- Clause 4 package Asis.Errors
- Clause 5 package Asis.Exceptions
- Clause 6 package Asis.Implementation
- Clause 7 package Asis.Implementation.Permissions
- Clause 8 package Asis.Ada_Environments
- Clause 9 package Asis.Ada_Environments.Containers
- Clause 10 package Asis.Compilation_Units
- Clause 11 package Asis.Compilation_Units.Times
- Clause 12 package Asis.Compilation_Units.Relations
- Clause 13 package Asis.Elements
- Clause 14 package Asis.Iterator
- Clause 15 package Asis.Declarations
- Clause 16 package Asis.Definitions
- Clause 17 package Asis.Expressions
- Clause 18 package Asis.Statements
- Clause 19 package Asis.Clauses
- Clause 20 package Asis.Text
- Clause 21 package Asis.Ids
- Clause 22 package Asis.Data_Decomposition (optional package)
- Clause 23 package Asis.Data_Decomposition.Portable_Transfer

The following annexes are informative:

Annex A: Glossary
Annex B: ASIS Application Examples
Annex C: Miscellaneous ASIS I/O and IDL Approaches
Annex D: Rationale

The major package interfaces visible to ASIS users are identified as clauses facilitating access from the table of contents.

The ASIS interface is compilable. Consequently, Sentinels have been used to mark portions of the ASIS text with comments appropriate to an ASIS implementor and an ASIS user.

The sentinels and their meanings are:

--|ER (Element Reference) These comments mark an element kind reference which acts as a header for those queries that work on this element kind.

--|CR (Child Reference) These sentinel comments follow sentinel comments marking element references (--ER) and reference child element queries that decompose the element into its children.

--|AN (Application Note) These comments describe suggested uses, further analysis, or other notes of interest to ASIS applications.

--|IP (Implementation Permissions) These comments describe permissions given an implementor when implementing the associated type or query.

--|IR (Implementation Requirements) These comments describe additional requirements for conforming implementations.

1.1.3 Conformity with this International Standard

1.1.3.1 Implementation conformance requirements

An *ASIS implementation* includes all the hardware and software that implements the ASIS specification for a given Ada implementation and that provides the functionality required by the ASIS specification. An ASIS *implementor* is a company, institution, or other group (such as a vendor) who develops an ASIS implementation. A conforming ASIS implementation shall meet all of the following criteria:

a) The system shall support all required interfaces defined within this International Standard. These interfaces shall support the functional behavior described herein. All interfaces in the ASIS specification are required unless the interface is specifically identified as being optional. The ASIS specification defines one optional package: Asis.Data_Decomposition. Asis.Data_Decomposition has one child package, Asis.Data_Decomposition.Portable_Transfer.

b) The system may provide additional facilities not required by this International Standard. *Extensions* are non-standard facilities (e.g., other library units, non-standard children of standard ASIS library units, subprograms, etc.) which provide additional information from ASIS types, or modify the behavior of otherwise standard ASIS facilities to provide alternative or additional functionality. Nonstandard extensions shall be identified as such in the system documentation. Nonstandard extensions, when used by an application, may change the behavior of functions or facilities defined by this International Standard. The conformance document shall define an environment in which an application can be run with the behavior specified by this International Standard. In no case except package name conflicts shall such an environment require modification of a Basic Conforming or Fully Conforming ASIS Application. An implementation shall not change package specifications in this International Standard except by:

- Adding “with” clauses, pragmas, representation specifications, comments, and allowable pragmas. Allowable pragmas are those which do not change the semantics of the interface (e.g., List, Optimize, Page).
- Replacing instances of the words <implementation-defined> with appropriate value(s).
- Adding or changing private parts.
- Making any other changes that are lexically transparent to Ada compilers.

c) An ASIS implementation shall not raise Program_Error on elaboration of an ASIS package, or on execution of an ASIS subprogram, due to elaboration order dependencies in the ASIS implementation.

d) Except as explicitly provided for in this International Standard, Standard.Storage_Error is the only exception that should be raised by operations declared in this International Standard.

e) When executed, an implementation of this International Standard shall not be erroneous, as defined by ISO/IEC 8652:1995.

1.1.3.2 Implementation conformance documentation

A conformance document shall be available for an implementation claiming conformance to this International Standard. The conformance document shall have the same structure as this International Standard, with the information presented in the equivalently numbered clauses, and subclauses. The conformance document shall not contain information about extended facilities or capabilities outside the scope of this International Standard.

The conformance document shall contain a statement that indicates the full name, number, and date of the International Standard that applies. The conformance document may also list software standards approved by ISO/IEC or any ISO/IEC member body that are available for use by a Basic or Fully Conforming ASIS Application. Applicable characteristics whose documentation is required by one of these standards, or by standards of government bodies, may also be included.

The conformance document shall describe the behavior of the implementation for all implementation-defined features defined in this International Standard. This requirement shall be met by listing these features and providing either a specific reference to the system documentation or providing full syntax and semantics of these features. The conformance document shall specify the behavior of the implementation for those features where this International Standard states that implementations may vary.

No specifications other than those described in this subclause shall be present in the conformance document.

The phrase “shall be documented” in this International Standard means that documentation of the feature shall appear in the conformance document, as described previously, unless the system documentation is explicitly mentioned.

The system documentation should also contain the information found in the conformance document.

1.1.3.3 Implementation conformance categories

An implementation is required to define all of the subprograms for all of the operations defined in this International Standard, including those whose implementation is optional. *Required functionality* is the subset of ASIS facilities which are not explicitly identified in the ASIS standard as optional. *Optional functionality* is the subset of ASIS facilities which are explicitly identified in the ASIS standard as optional which may legitimately be omitted from a Basic Conforming ASIS implementation. Optional interfaces shall be included in any Fully Conforming ASIS implementation, unless stated otherwise in the ASIS specification. An application that accesses an Ada environment’s semantic tree (e.g., Diana Tree) directly using work-arounds is not considered to be a conformant application. All Conforming Applications fall within one of the categories defined below.

If an unimplemented feature is used, the exception Asis.ASIS_Failed shall be raised and Asis.Implementation_Status shall return the value for Error_Kinds of Not_Implemented_Error.

There are four categories of conforming ASIS implementations:

1.1.3.3.1 Basic conforming ASIS implementation
A Basic Conforming ASIS Implementation is an ASIS implementation supporting all required interfaces defined within this International Standard.

1.1.3.3.2 Fully conforming ASIS implementation
A Fully Conforming ASIS Implementation is an ASIS implementation supporting all required and all optional interfaces defined within this International Standard.

1.1.3.3.3 Basic conforming ASIS implementation using extensions
A Basic Conforming ASIS Implementation Using Extensions is an ASIS implementation that differs from a Basic Conforming ASIS Implementation only in that it uses nonstandard extensions that are consistent with this International Standard. Such an implementation shall fully document its extended facilities, in addition to the documentation required for a Basic Conforming ASIS Implementation.

1.1.3.3.4 Fully conforming ASIS implementation using extensions
A Fully Conforming ASIS Implementation Using Extensions is an ASIS implementation that differs from a Fully Conforming ASIS Implementation only in that it uses nonstandard extensions that are consistent with this International Standard. Such an implementation shall fully document its extended facilities, in addition to the documentation required for a Fully Conforming ASIS Implementation.

1.1.3.4 Application conformance categories
An *ASIS application* is any programming system or any set of software components making use of ASIS queries to obtain information about any set of Ada components. All ASIS applications claiming conformance to this International Standard shall use a Conforming ASIS Implementation with or without extensions.

1.1.3.4.1 Basic conforming ASIS application
A Basic Conforming ASIS Application is an application that only uses the required facilities defined within this International Standard. It shall be portable to any Conforming ASIS Implementation.

1.1.3.4.2 Fully conforming ASIS application
A Fully Conforming ASIS Application is an application that only uses the required facilities and the optional facilities defined within this International Standard. It shall be portable to any Fully Conforming ASIS Implementation.

1.1.3.4.3 Basic conforming ASIS application using extensions
A Basic Conforming ASIS Application Using Extensions is an application that differs from a Basic Conforming ASIS Application only in that it uses nonstandard, implementation provided, extended facilities that are consistent with this International Standard. Such an application should fully document its requirements for these extended facilities. A Basic Conforming ASIS Application Using Extensions may or may not be portable to other Basic or Fully Conforming ASIS Implementation Using Extensions.

1.1.3.4.4 Fully conforming ASIS application using extensions
A Fully Conforming ASIS Application Using Extensions is an application that differs from a Fully Conforming ASIS Application only in that it uses nonstandard, implementation provided, extended facilities that are consistent with this International Standard. Such an application should fully document its requirements for these extended facilities. A Fully Conforming ASIS Application Using Extensions may or may not be portable to other Fully Conforming ASIS Implementation Using Extensions.

1.1.4 Implementation permissions

The ASIS Application Program Interface (API) may be implemented through a variety of approaches. Approaches permitted by this International Standard are based on the traditional approach and the client /server approach. These implementation permissions are depicted in Figure 1 and described below:

1.1.4.1 Traditional approach (permission 1)

Traditionally, the ASIS API implementation is intended to execute on the node containing the implementor's Ada software engineering environment and the desired Ada compilation environment. Because the ASIS API interfaces directly, ASIS performs at its best. It is expected that most ASIS implementors will support this approach as it requires little additional effort when alternative approaches are supported. In Figure 1, the client tool using Permission 1 uses the ASIS specification exactly as specified in this International Standard. ASIS tools and applications are compiled in the implementor's environment.

1.1.4.2 Client / server approach (permission 2)

As an alternative, a client / server approach can be used to implement the ASIS API. Here the ASIS API is supported by a server; ASIS client tools can request ASIS services within the supported network.

Figure 1 identifies four ASIS client tools using permission 2 capable of interfacing with an ASIS Object Request Broker (ORB) server. One client tool is written in Ada, one in Java, one in C++, and one in Smalltalk. The ORB serves as a broker between the client and server on a network consisting of many nodes. Server location and services are registered with the ORB. A client needing the services interfaces with the ORB, who brokers the needed server interface information. The interface between a client and server is written as an interface specification in the Interface Definition Language (IDL). IDL is very different from most computer languages; when IDL is compiled, the interface specification is produced in either Ada, Java, C++, or Smalltalk. In addition, the necessary artifacts are produced to register the client or server interface with the ORB.

1.1.4.3 Distributed traditional approach (permission 3)

The Ada specification created by the compilation of this ASIS API in IDL is semantically equivalent to this ASIS standard, but not syntactically identical. An ASIS Client tool written in Ada interfaces to the ASIS API as specified in this International Standard. As shown in Figure 1, the ASIS API encapsulates the ASIS ORB client as generated from the compilation of the ASIS IDL into Ada. Client tools using either permission 1 or permission 3 are, most likely, identical. Client tools developed using permission 3 can be developed as plug and play.

1.1.4.4 ASIS dynamic client approach (permission 4)

In addition to using traditional compiled tools through the client / server interface, ORBs can provide a Dynamic Interface Invocation (DII) capability where rather general purpose tools can access the interface dynamically. Shown in Figure 1, such a tool behaves more like a browser. It accesses the ASIS IDL as registered with the server and browses through the services provided by the ASIS interface. Use of this capability with ASIS is extremely cumbersome and manually intensive. However, this provides a user access to information across the interface that had not been preprogrammed by a tool.

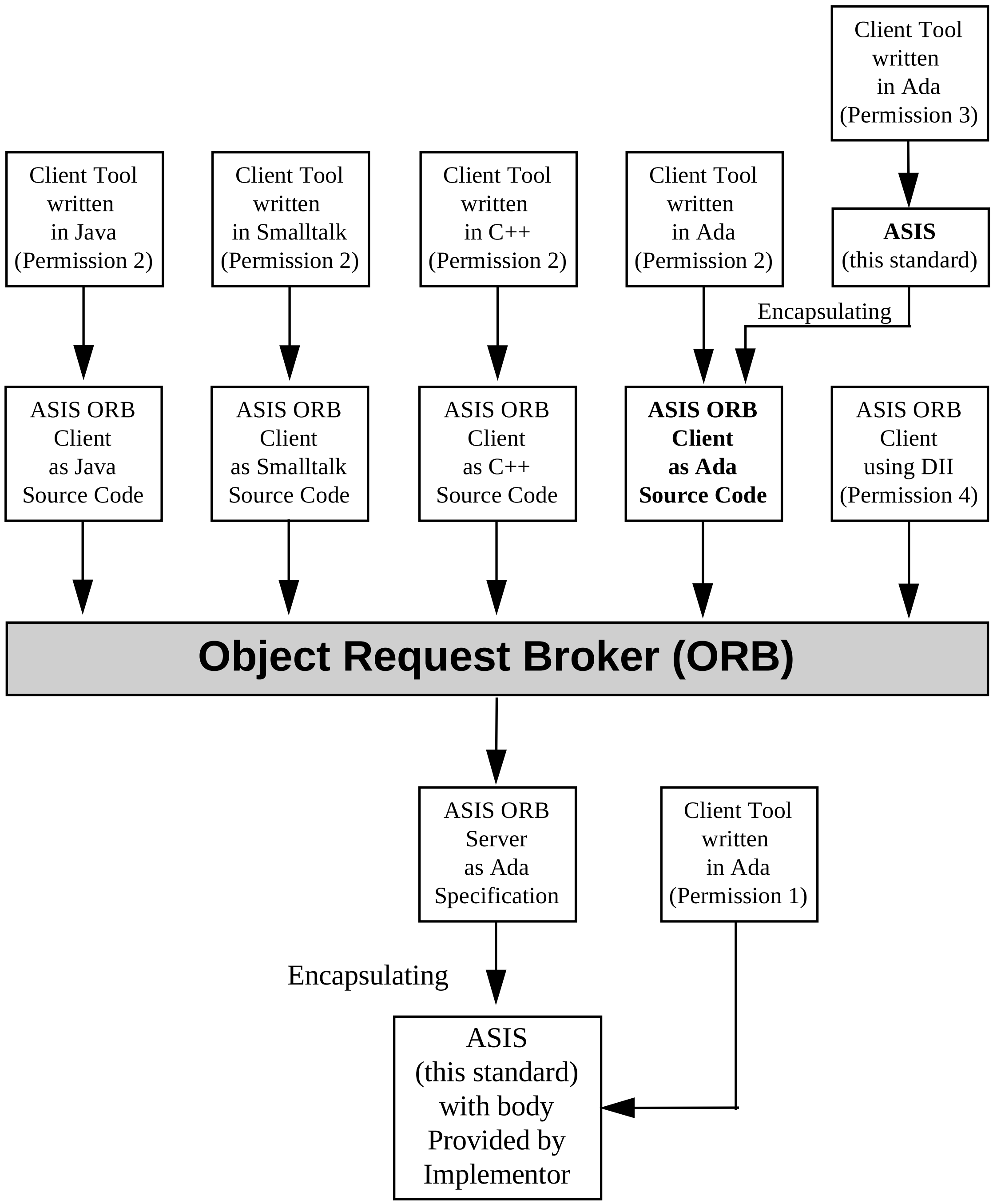

Figure 1 — ASIS implementation permissions

1.1.5 Classification of errors

ASIS reports all operational errors by raising an exception. Whenever an ASIS implementation raises one of the exceptions declared in package Asis.Exceptions, it will previously have set the values returned by the Status and Diagnosis queries to indicate the cause of the error. The possible values for Status are indicated here along with suggestions for the associated contents of the Diagnosis string.

ASIS applications are encouraged to follow this same convention whenever they explicitly raise any ASIS exception to always record a Status and Diagnosis prior to raising the exception. Values of errors along with their general meanings are:

```
Not_An_Error               -- No error is presently recorded
Value_Error                -- Routine argument value invalid
Initialization_Error       -- ASIS is uninitialized
Environment_Error          -- ASIS could not initialize
Parameter_Error            -- Bad Parameter given to Initialize
Capacity_Error             -- Implementation overloaded
Name_Error                 -- Context/unit not found
Use_Error                  -- Context/unit not use/open-able
Data_Error                 -- Context/unit bad/invalid/corrupt
Text_Error                 -- The program text cannot be located
Storage_Error              -- Storage_Error suppressed
Obsolete_Reference_Error   -- Semantic reference is obsolete
Unhandled_Exception_Error  -- Unexpected exception suppressed
Not_Implemented_Error      -- Functionality not implemented
Internal_Error             -- Implementation internal failure
```

Diagnostic messages may be more specific.

A set of exceptions shall be raised for the following circumstances:

- **ASIS_Inappropriate_Context** - Raised when ASIS is passed a Context value that is not appropriate for the operation. This exception typically indicates that a user error has occurred within the application.
- **ASIS_Inappropriate_Compilation_Unit** - Raised when ASIS is passed a Compilation_Unit value that is not appropriate. This exception typically indicates that a user error has occurred within the application.
- **ASIS_Inappropriate_Element -** Raised when ASIS is given an Element value that is not appropriate. This exception typically indicates that a user error has occurred within the application.
- **ASIS_Inappropriate_Line -** Raised when ASIS is given a Line value that is not appropriate.
- **ASIS_Inappropriate_Line_Number** - Raised when ASIS is given a Line_Number value that is not appropriate. This exception typically indicates that a user error has occurred within the application.
- **ASIS_Failed** - All ASIS routines may raise ASIS_Failed whenever they cannot normally complete their operation. This exception typically indicates a failure of the underlying ASIS implementation. This is a catch-all exception that is raised for different reasons in different ASIS implementations.

1.2 Normative reference

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

ISO/IEC 8652:1995, *Information technology — Programming languages — Ada.*

1.3 Terms and definitions

For the purposes of this International Standard, the terms and definitions given in ISO/IEC 8652:1995 and the following apply.

Additional terms are defined throughout this International Standard, indicated by *italic* type. Terms explicitly defined in this International Standard are not to be presumed to refer implicitly to similar terms defined elsewhere. Terms not defined in this International Standard and ISO/IEC 8652:1995 are to be interpreted according to *the Webster's Third New International Dictionary of the English Language*. Informal descriptions of some terms are also given in Annex A, "Glossary".

"ASIS" is used in reference to the acronym "Ada Semantic Interface Specification."
"Asis" is used in reference to the package Asis.

2 ASIS technical concepts

2.1 Ada compilation environment

ASIS is an interface between an Ada environment as defined by ISO/IEC 8652:1995 (the Ada Reference Manual) and any tool requiring information from this environment, as shown in Figure 2.

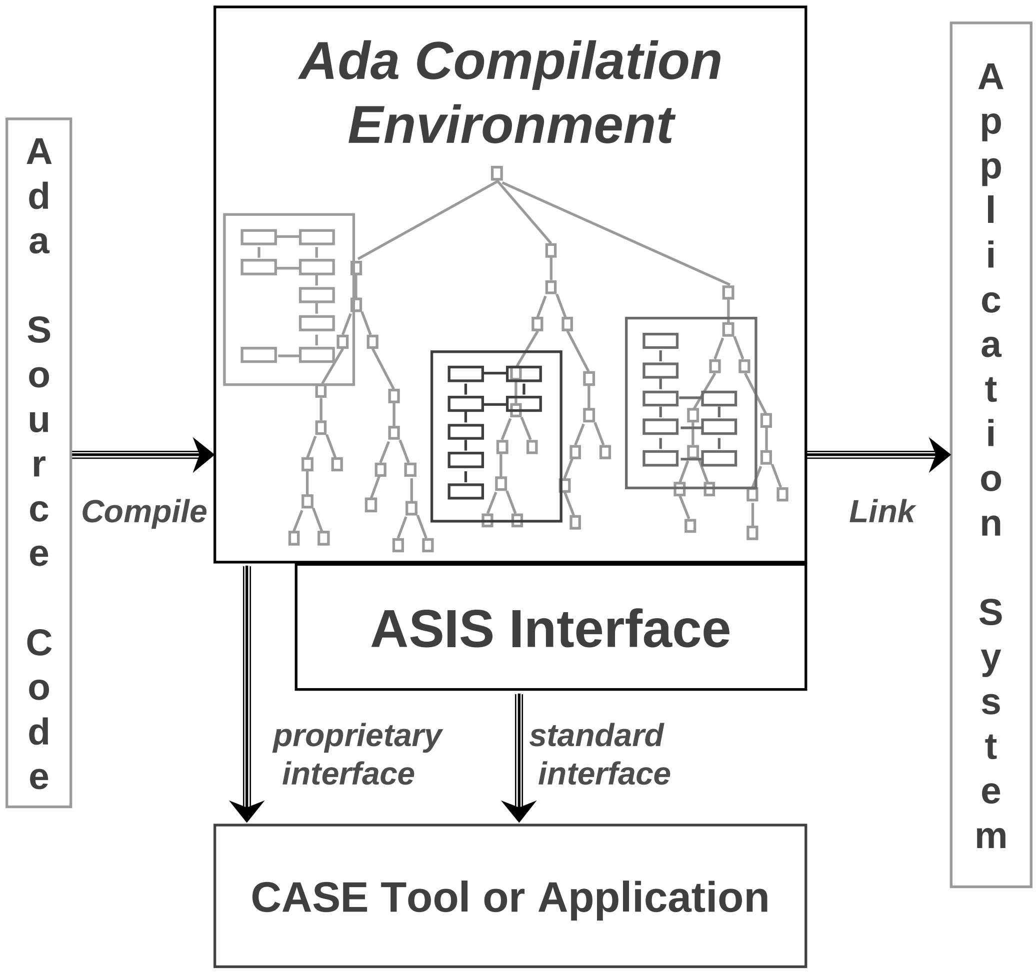

Figure 2 — ASIS as interface to Ada compilation environment

2.1.1 Ada environment

ISO/IEC 8652:1995, 10.1.4(1) provides a notion for this compilation *environment* as: **"**Each compilation unit submitted to the compiler is compiled in the context of an environment declarative_part (or simply environment), which is a conceptual declarative_part that forms the outermost declarative region of the context of any compilation. At run time, an environment forms the declarative_part of the body of the environment task of a partition."

2.1.2 ASIS notion of the Ada compilation environment

However, the mechanisms for creating an environment and for adding and replacing compilation units within an environment are implementation-defined. Some implementor environments create and maintain persistent databases while others do not. Consequently, ASIS requires the user of the interface (i.e., ASIS application) to establish the compilation environment. This is done through the context.

The *context* defines a set of compilation units and configuration pragmas processed by an ASIS application. ASIS provides any information from a context by treating this set as if its elements make up an environment declarative part by modeling some view (most likely - one of the views of the underlying Ada implementation) on the environment. Context is a view of an Ada environment. ASIS requires an application to identify that view of the environment using the procedure Asis.Ada_Environments.Associate, as shown in Figure 3. ASIS may process several different contexts at a time.

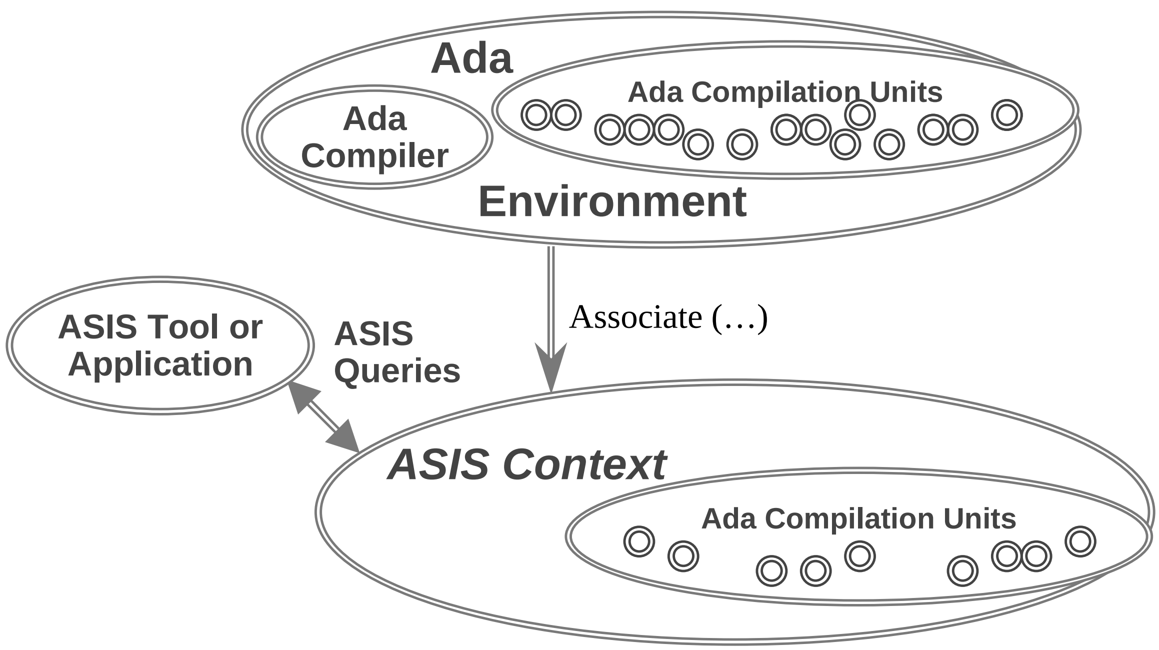

Figure 3 — Application interface to ASIS Context

A context may have one or more Compilation_Units. ASIS has defined Compilation_Unit as an ASIS private type. This type has values which denote an Ada compilation unit or configuration pragma from the environment. Compilation_Unit also is an abstraction, which represents information about some physical object from the "external world". This physical object is treated by the underlying Ada implementation as the corresponding Ada compilation unit or as a result of compiling a configuration pragma. An ASIS *compilation unit* includes the notion of some implementation-defined way to associate the corresponding ASIS object with some physical external object. This is necessary to support ASIS queries such as Time_Of_Last_Update and Text_Name which have no relation to the Reference Manual-defined notion of an Ada compilation unit.

To facilitate the use of context, implementations may support the use of *containers* which are logical collections of ASIS compilation units. For example, some containers can hold compilation units which include Ada predefined types; another container can hold implementation-defined packages. Containers provide an implementation-defined way of grouping the compilation units accessible for an ASIS application through the ASIS queries.

2.1.3 Illegal / inconsistent units in the compilation environment

Ada Implementation permissions allow for illegal and inconsistent units to be in the environment. Because the contents of the Ada environment are Ada-implementation-defined, the ASIS context can contain illegal compilation units. The use of ASIS can result in the exception ASIS_Failed being raised if the Ada environment includes such units.

2.2 ASIS queries

ASIS queries are provided in the form of structural (syntactic) and semantic queries to the Ada compilation environment.

2.2.1 Structural queries

Structural queries are those ASIS queries which provide the top-down decomposition and reverse bottom-up composition of the compilation unit according to its syntax structure. These structural queries are further characterized as:

- "Black-box" queries are those ASIS queries which produce information about compilation units
- "White-box" queries are those ASIS queries which produce information about lexical elements of compilation units.

2.2.2 Semantic queries

Semantic queries are those ASIS queries which express semantic properties of ASIS Elements in terms of other Elements. There are three kinds of semantic queries in ASIS:

- Semantic queries about Elements,
- Semantic queries about Compilation Units, and
- Semantic queries about Dependence Order.

2.2.3 General ASIS query processing

Both structural (syntactic) and semantic queries are facilitated through the notion of ASIS elements and their kinds. Most ASIS queries provide for the processing of specific constructs with respect to the Ada Reference Manual and the processing of their lists.

2.2.3.1 Elements and element kinds

The base object in ASIS is the Asis.Element. *Element* is a common abstraction used to denote the syntax (both explicit and implicit) of ASIS compilation units. Elements correspond to nodes of a hierarchical tree representation of an Ada program. Most elements of the tree have child elements. These children can appear as single elements (possibly with children themselves) or as a list of elements (also possibly with children). As an example, think of an Ada object declaration having three sub-parts or children:

- A list of identifiers,
- A reference to a subtype (subtype indication),
- An initialization expression (possibly absent)

Thus, the declaration: **`A, B : Latitude := 0.0;`**

has a corresponding tree as in Figure 4.

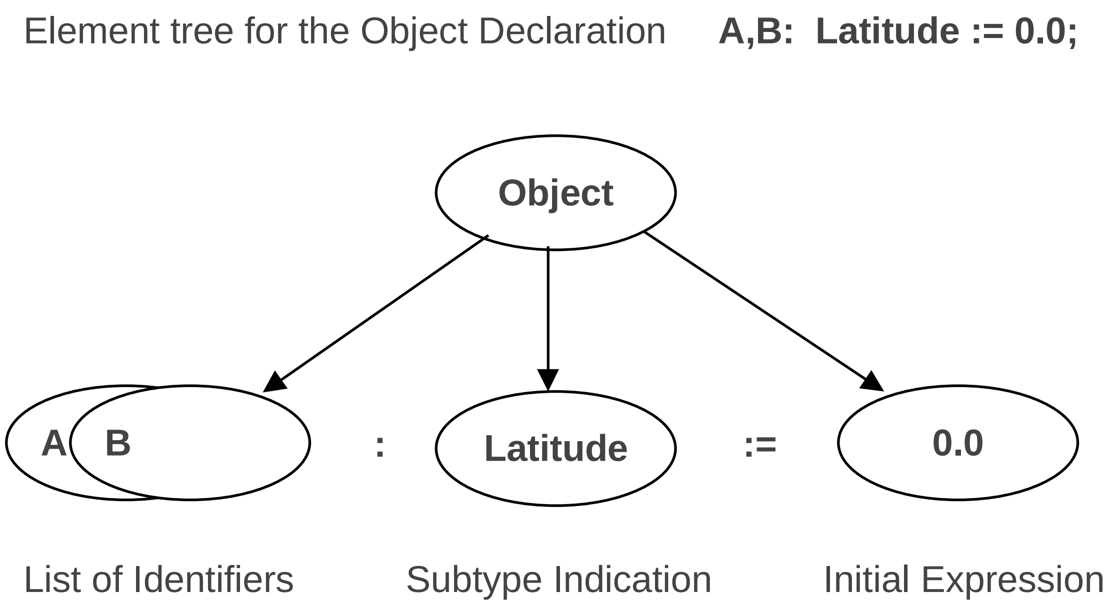

Figure 4 — Syntactic tree representation of an Ada object declaration

ASIS Elements are either *explicit elements*, representing a language construct that appears explicitly in the program text for the compilation unit, or *implicit elements*, representing a language construct that does not exist in the program text for the compilation unit, but could occur at a given place in the program text as a consequence of the semantics of another construct (e.g., an implicit declaration, a generic instantiation).

ASIS provides the ability to visit elements of the tree and ask questions of each element. One key hierarchy of questions begins with: “What kind of element do I have?” Elements of the highest level of the ASIS hierarchy are classified into kinds. The following table identifies the high level hierarchy of kinds, the primary ASIS package to support processing of those kinds, and the references for those kinds in the Ada Reference Manual.

A_Defining_Name	Asis.Definitions	Reference Manual	3, 6
A_Declaration	Asis.Declarations	Reference Manual	3, 5, 6, 7, 8, 9, 10, 11, 12
A_Definition	Asis.Definitions	Reference Manual	3, 7, 9, 12
An_Expression	Asis.Expressions	Reference Manual	2, 4
A_Statement	Asis.Statements	Reference Manual	5, 6, 9, 11, 13
A_Path	Asis.Statements	Reference Manual	5, 9
A_Clause	Asis.Clauses	Reference Manual	8, 10, 13
An_Association	Asis.Expressions	Reference Manual	2, 3, 4, 6, 12
An_Exception_Handler	Asis.Statements	Reference Manual	11.2
A_Pragma	Asis.Elements	Reference Manual	2, 10, 11, 13, B, G, H, I, L

The function Asis.Elements.Element_Kind classifies any element into one of these kinds. Once the client knows that an element is a declaration, for example, it can further classify the element as to what kind of declaration it is with the Asis.Elements.Declaration_Kind function. This leads to case structures like the following.

EXAMPLE Case statement to classify elements:

```
case Asis.Elements.Element_Kind (My_Element) is              -- 13.6
    when Asis.A_Declaration =>                               --  3.9.1
        case Asis.Elements.Declaration_Kind (My_Element) is  -- 13.9
            when Asis.A_Variable_Declaration =>              --  3.9.4
                { statement }
            when others =>
                null;
        end case;
    when Asis.A_Statement =>                                 --  3.9.1
        case Asis.Elements.Statement_Kind (My_Element) is    -- 13.25
            when Asis.A_Block_Statement =>                   --  3.9.20
                { statement }
            when others =>
                null;
        end case;
    when others =>
        null;
end case;
```

In this example, variable declarations and block statements will presumably be processed further. All other element kinds are ignored by falling into the null *when others* alternative.

2.2.3.2 Processing specific constructs

Once this level of classification is determined, ASIS provides a set of functions for processing a specific statement, declaration, or other element kinds. The following functions are available for processing object declarations:

```
function Names (Declaration : Asis.Declaration)                      -- 15.1
        return Asis.Defining_Name_List;
function Object_Declaration_View (Declaration : Asis.Declaration)    -- 15.9
        return Asis.Definition;
function Initialization_Expression (Declaration : Asis.Declaration)  -- 15.10
        return Asis.Expression;
```

These functions correspond to each of the three parts of an object declaration. Once it is classified into its kind, each element can be processed further with the provided functions. Some functions “traverse” to other child elements such as the Object_Declaration_View and Initialization_Expression functions above. Since the kind of element that is returned for each of these functions is already known, processing can continue directly with the functions in the Asis.Definitions package (Clause 16) that accept A_Type_Definition elements or the Asis.Expressions package (Clause 17) that accept An_Expression elements. The exception ASIS_Inappropriate_Element is raised whenever an ASIS function is passed an element it is not intended to process. Parameter names and subtype names in the function specifications indicate what kind of input element is expected and what kind of element is returned.

2.2.3.3 Element list processing

Lists of elements are processed with a loop iteration scheme in the following manner:

EXAMPLE Loop iteration scheme

```
    List : constant Asis.Element_List :=                         -- 3.7
        <ASIS function returning a list>;
    An_Element : Asis.Element;                                   -- 3.6
begin
    for I in List’Range loop
        An_Element := List (I);
        Process (An_Element);
    end loop;
```

Functions that return Element_List types generally indicate what type of elements are in the list they return. Thus, processing can sometimes continue with specific calls without first asking what type of element is being processed.

2.2.3.4 Operations that apply to all elements

Figure 5 depicts operations which, in general, apply to all elements.

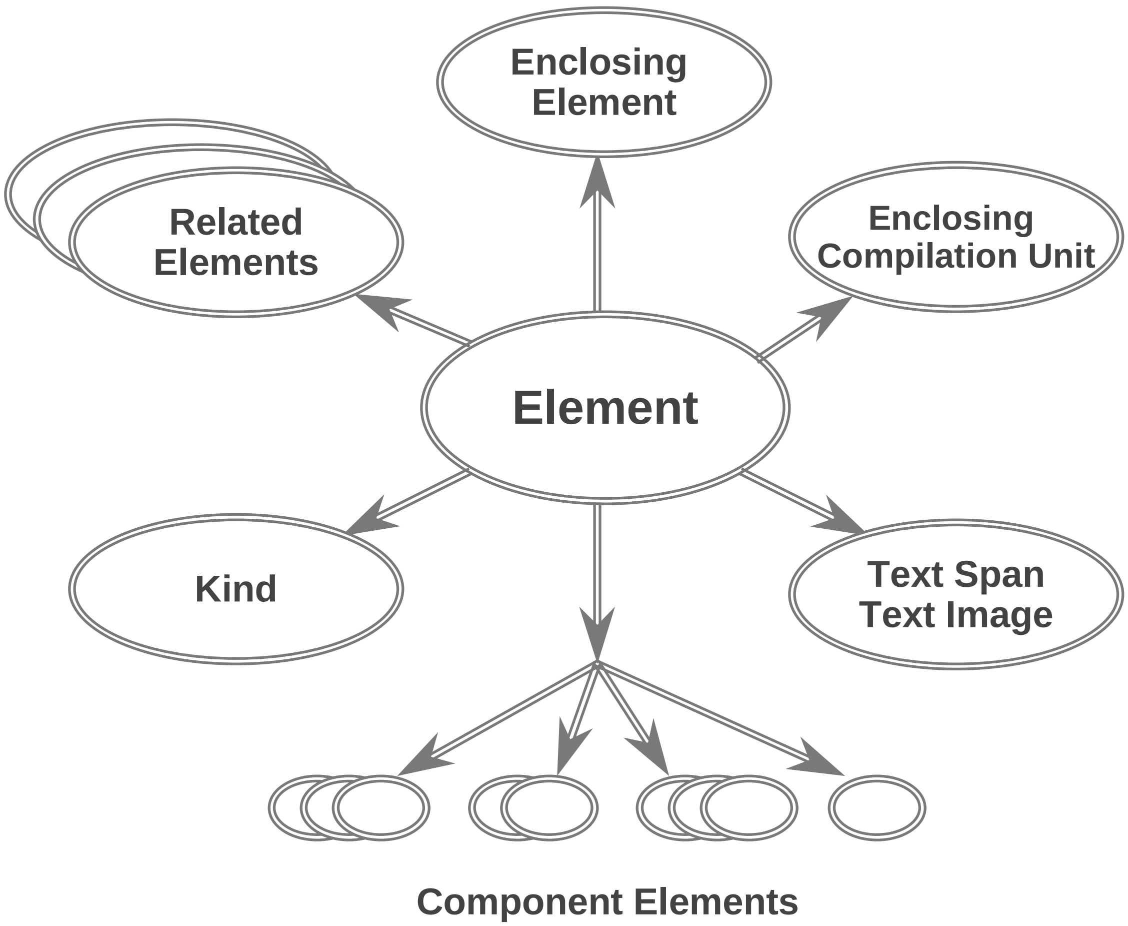

Figure 5 — Operations on elements

The ASIS packages provide some general interfaces that operate on all nodes, such as:

- Asis.Elements.Element_Kind (13.6) - Returns the Element_Kind for the element. Once the Element_Kind is known, the Element can be decomposed into its component elements.
- Asis.Elements.Enclosing_Element (13.36) - Returns the element that contains the current element (one element up in the tree hierarchy).
- Asis.Elements.Enclosing_Compilation_Unit (13.2) - Returns the Compilation_Unit that contains the given element.
- Asis.Text.First_Line_Number (20.8) - Returns the first line number in which the text of the element resides.

- Asis.Text.Last_Line_Number (20.9) - Returns the last line number in which the text of the element resides.
- Asis.Text.Element_Span (20.10) - Returns the span for the element.
- Asis.Text.Lines (20.19) - Returns a list of lines for the element.
- Asis.Text.Element_Image (20.23) - Returns the program text image of any element.
- Asis.Elements.Hash (13.42) - Returns a convenient name for an object of type Asis.Element.
- Asis.Ids.Create_Id (21.7) - Returns a unique Id value corresponding to this element.
- Asis.Elements.Is_Nil (13.30) - Determines whether an element is nil. Some functions return a Nil_Element when a potential element does not exist in the program. This is true for the Initialization_Expression function above when no initial value is present in the declaration.

2.2.3.5 Semantic references

References from one part of an Ada program to another can be traversed with functions whose name begins with “Corresponding_”, such as: Corresponding_Children, Corresponding_Declaration, Corresponding_Body, Corresponding_Type_Declaration, Corresponding_Type, Corresponding_Body_Stub, Corresponding_Name_Definition, Corresponding_Name_Declaration, Corresponding_Loop_Exited, Corresponding_Entry, etc. If an element references another element, the user can traverse to the referenced element with this function. A Nil_Element is returned if no definition traversal is possible.

A slightly more precise statement of the operation of this function is that identifier references point to identifier definitions. Traversal arrives at the identifier definition where the Enclosing_Element function is used to arrive at the complete element declaration.

For example, the user can traverse to the type declaration referenced in an object declaration with the Corresponding_Expression_Type function as shown in Figure 6. Figure 6 also depicts links using the semantic query Corresponding_Name_Declaration. (Note: the thin lined-arrows depict syntactic queries while the thick-lined arrows depict semantic queries.)

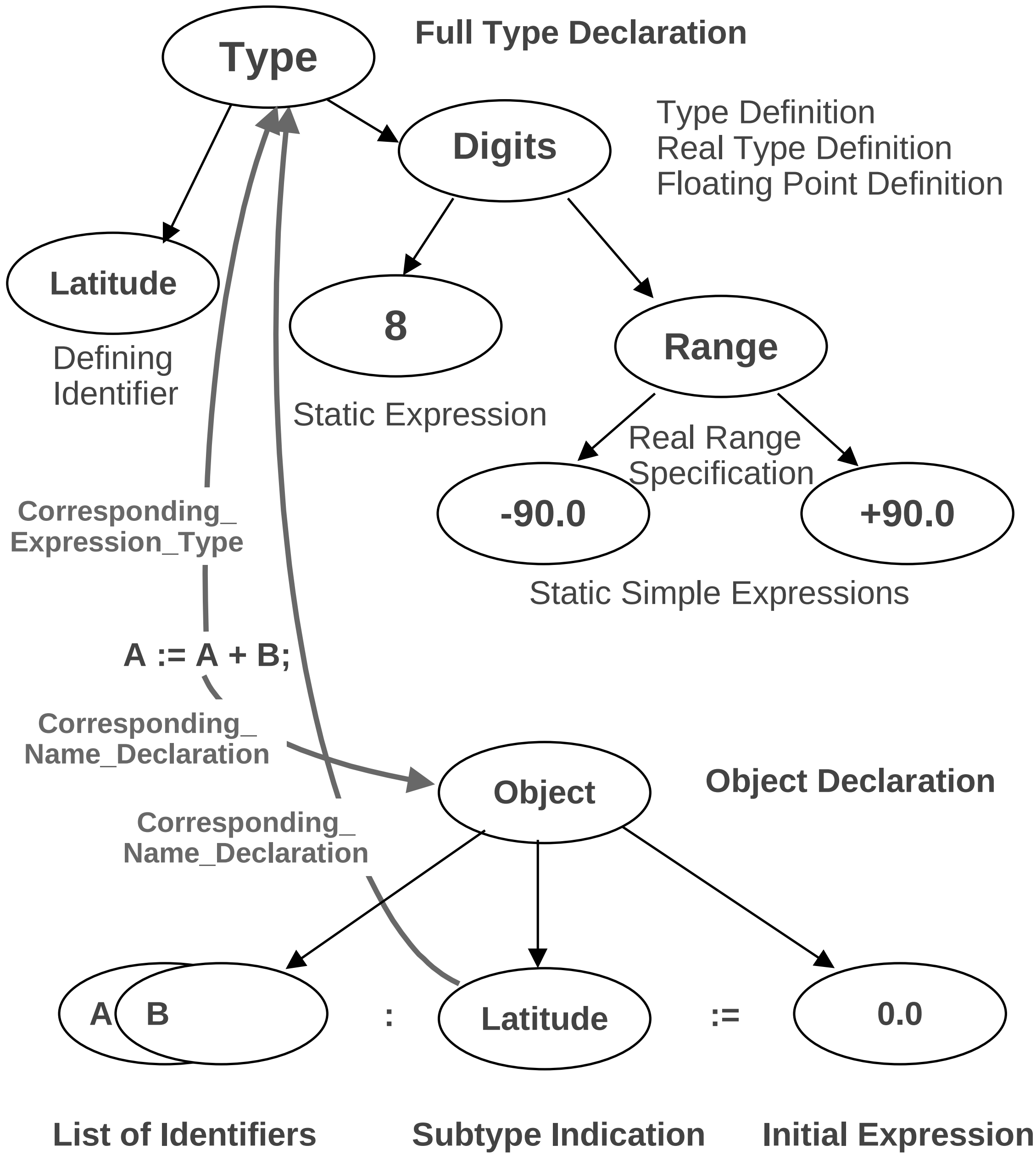

Figure 6 — Semantic reference using corresponding queries

2.3 ASIS package architecture

Figure 7 depicts the package architecture for the ASIS interface. A tool or application using ASIS has visibility to the entire declarative region of package Asis including all child packages. To get a better understanding of how these packages are used, see the examples in Annex B.

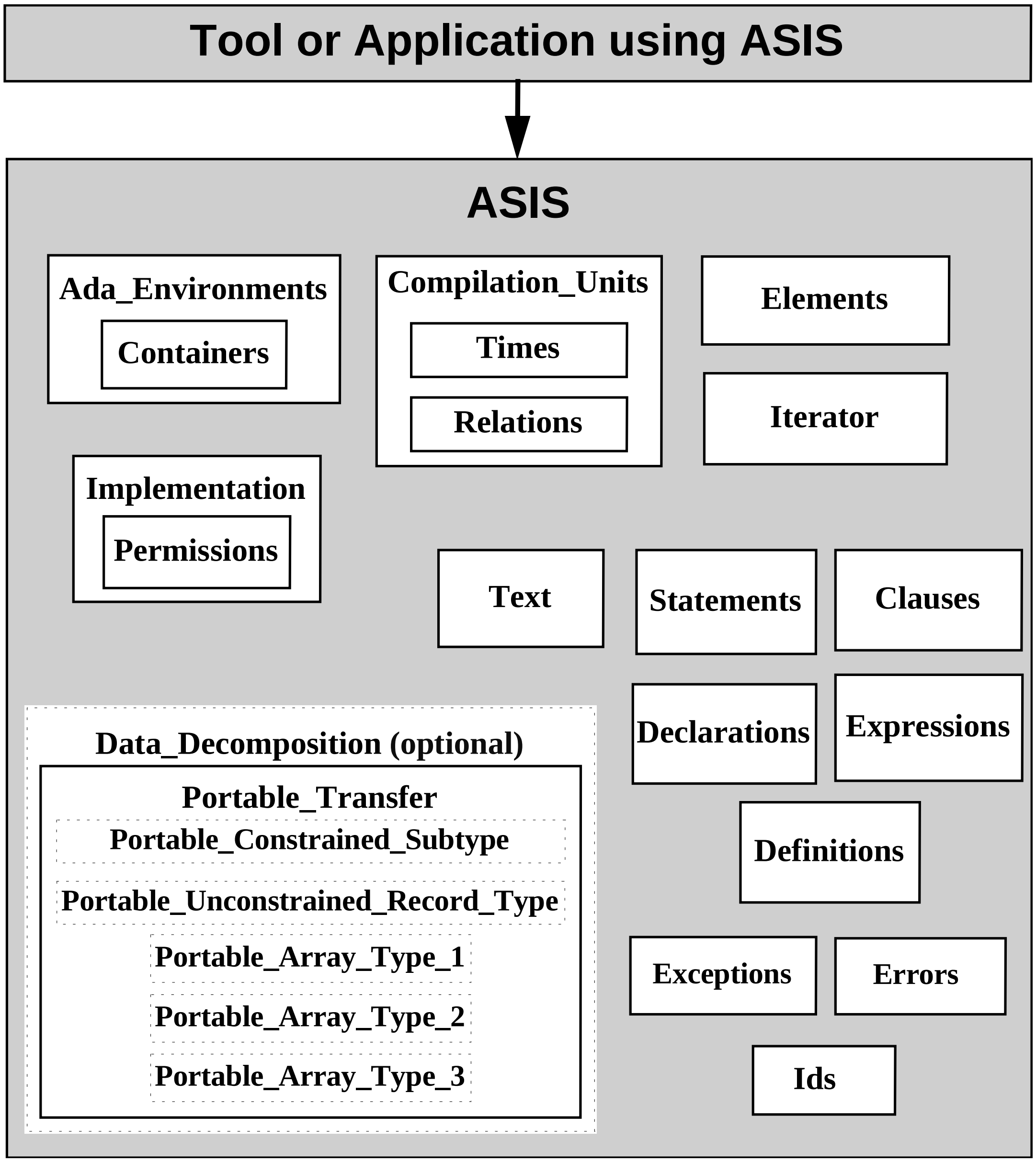

Figure 7 — ASIS package architecture

package ASIS - Package ASIS, together with its children, provides the interface between an Ada environment (as defined by ISO/IEC 8652:1995) and any tool requiring information from it. Valuable semantic and syntactic information is made available via queries through child packages of package ASIS.

Package Asis contains common types and subtypes used for the ASIS interface and its child packages. Important common types include (see Clause 3):

- Type Context helps identify the compilation units considered to be analyzable as part of the Ada compilation environment.
- Type Element is an abstraction of entities within a logical Ada syntax tree.
- Element Kinds are a set of enumeration types providing a mapping to the Ada syntax.
- Type Compilation_Unit is an abstraction for Ada compilation units.
- Unit Kinds are a set of enumeration types describing the various kinds of compilation units.
- Type Traverse_Control provides a mechanism to control iterative traversals of a logical syntax tree.
- Type Program_Text provides an abstraction for the program text for a program's source code.

Package Asis also encapsulates implementor-specific declarations, which are made available to ASIS and its client applications in an implementor-independent manner. Package ASIS is the root of the ASIS interface. All other packages are child packages of package Asis. These packages are:

- **Asis.Ada_Environments** - This child package encapsulates a set of queries that map physical Ada compilation and program execution environments to logical ASIS environments. An ASIS Context is associated with some set of Ada compilation units maintained by an underlying Ada implementation. After this association has been made, this set of units is considered to be part of the compile-time Ada environment, which forms the outermost context of any compilation, as specified in section 10.1.4 of the Ada Reference manual. This same environment context provides the implicit outermost anonymous task during program execution. If an Ada implementation supports the notion of a program library or "library" as specified in section 10(2) of the Ada Reference Manual, then an ASIS Context value can be mapped onto one or more implementor libraries represented by Containers. More than one context may be manipulated at a time. Important interfaces include: Associate, Dissociate, Open, and Close (see Clause 8). The type Container and its supporting functions are provided in the child package **Asis.Ada_Environments.Containers** (see Clause 9).
- **Asis.Implementation** - This child package provides operations to initialize and finalize the ASIS interface. It also provides queries for the error status of the ASIS implementation (see Clause 6). Its child package **Asis.Implementation.Permissions** provides queries to determine options used by an implementation (see Clause 7).
- **Asis.Compilation_Units** - This child package encapsulates a set of queries that implement the ASIS Compilation_Unit abstraction. It defines queries that deal with Compilation_Units and the gateway queries between Compilation_Units, Elements, and Ada_Environments. More than one compilation unit may be manipulated at one time (see Clause 10). The child package **Asis.Compilation_Units.Times** encapsulates the time related functions used within ASIS (see Clause 11). A second child package, **Asis.Compilation_Units.Relations** encapsulates the semantic relationship concepts used in ASIS. *Relation* queries provide references to compilation units that are related, in some specific fashion, to one or more given compilation units (see Clause 12).
- **Asis.Iterator** - This child package encapsulates the Traverse_Element mechanism to perform an iterative traversal of a logical syntax tree. During the traversal, ASIS can analyze the various elements present and provide application processing via generic procedures instantiated by the application using queries to decompose ASIS elements into the logical semantic tree of the Ada program (see Clause 14). This key mechanism is the heart of most ASIS tools; examples of its use are provided in Annex B.

- **Asis.Elements** - This child package encapsulates a set of queries that operate on all elements and some queries specific to A_Pragma elements. Element_Kinds, defined in package Asis are defined as enumeration types describing the various kinds of elements. ASIS offers a hierarchical classification of elements. At the highest level, the Element_Kinds type provides literals that define "kinds" or classes into which all non-nil elements are grouped. Element_Kinds are: Not_An_Element, A_Pragma, A_Defining_Name, A_Declaration, A_Definition, An_Expression, An_Association, A_Statement, A_Path, A_Clause, and An_Exception_Handler. Elements in each of the Element_Kinds classes, with the exception of An_Exception_Handler, can be further classified by a subordinate kind at the next level in the hierarchy (see Clause 13).
- **Asis.Clauses** - This child package encapsulates a set of queries that operate on the A_Clause element (see Clause 19).
- **Asis.Declarations** - This child package encapsulates a set of queries that operate on A_Defining_Name and A_Declaration elements (see Clause 15).
- **Asis.Definitions** - This child package encapsulates a set of queries that operate on A_Definition elements (see Clause 16).
- **Asis.Expressions -** This child package encapsulates a set of queries that operate on An_Expression and An_Association elements (see Clause 17).
- **Asis.Statements -** This child package encapsulates a set of queries that operate on A_Statement, A_Path, and An_Exception_Handler elements (see Clause 18).
- **Asis.Text** - This child package encapsulates a set of operations to access the text of ASIS Elements. This text is represented as logical *lines* from the source code of the external representation of a compilation unit. Type Line is defined to support program text operations (see Clause 20). It assumes no knowledge of the existence, location, or form of the program text.
- **Asis.Errors** - This child package provides an enumeration type identifying the error kinds used for the ASIS interface (see Clause 4).
- **Asis.Exceptions** - This child package identifies all defined ASIS exceptions (see Clause 5).
- **Asis.Ids** - This child package encapsulates a set of operations and queries that implement the ASIS Id abstraction. An *Id* is a way to identify a particular element (i.e., a unique reference to an Element) which is efficient and persistent as long as the environment is not recompiled (see Clause 21).
- **Asis.Data_Decomposition** - This optional child package encapsulates a set of operations to decompose data values using the ASIS type information and a portable data stream, representing a data value of that type (see Clause 22). Its child package **Asis.Data_Decomposition.Portable_Transfer** provides support for logging and delogging of application data using ASIS. Internal packages of Asis.Data_Decomposition.Portable_Transfer include: Portable_Constrained_Subtype, Portable_Unconstrained_Record_Type, Portable_Array_Type_1 (for one dimensional arrays), Portable_Array_Type_2 (for two dimensional arrays), and Portable_Array_Type_3 (for three dimensional arrays) (see Clause 23).

2.4 Application use

The ASIS Interface is provided through child packages of package ASIS. Complete executable examples of application use are provided in Annex B. This section discusses the establishment of ASIS context, required sequencing of calls, erroneous applications, and usage rules.

2.4.1 Establishing ASIS context

An application using ASIS has a context clause for package ASIS and the child packages needed by the application.

An application using all child packages includes the following in its context clause:

```
Asis, Asis.Errors, Asis.Compilation_Units, Asis.Compilation_Units.Times,
Asis.Compilation_Units.Relations, Asis.Ada_Environments, Asis.Implementation,
Asis.Exceptions, Asis.Elements, Asis.Iterator, Asis.Declarations,
Asis.Expressions, Asis.Clauses, Asis.Definitions, Asis.Statements, Asis.Text,
Asis.Ids, Asis.Data_Decomposition; and Asis.Data_Decomposition.Portable_Transfer.
```

Only these packages should be referenced (withed) by a portable application. Other packages, which may be present in a specific ASIS implementation, are not part of this International Standard.

2.4.2 Required sequencing of calls

An ASIS application shall use the following required sequencing of calls:

```
a)  Asis.Implementation.Initialize;          -- Initialize the ASIS interface
b)  Asis.Ada_Environments.Associate(..);     -- Name an Ada Environment
c)  Asis.Ada_Environments.Open(..);          -- Access an Ada Environment
d)  Use the various ASIS queries.            -- Fetch a unit, its attributes,
                                             -- get its Unit_Declaration element,
                                             -- traverse its elements, etc.
e)  Asis.Ada_Environments.Close(..);         -- Drop access to an Ada Environment
f)  Asis.Ada_Environments.Dissociate(..);    -- Release the Ada Environment name
g)  Asis.Implementation.Finalize;            -- Release all resources
```

These calls may be used in a loop. More than one element may be manipulated at one time. (The exact number is subject to implementation/target specific limitations.) An element, obtained from a compilation unit, can continue to be manipulated while the Context, from which the element's compilation unit was obtained, remains open.

2.4.3 Notional ASIS application

ASIS Applications may take on many forms. This section is intended to present a notional ASIS Application using the example of a simple restrictions checker. A restrictions checker is intended to visit every element in an ASIS context to determine if a violation of a safety-critical check has been made. The ASIS context could include all compilation units in the Ada application. Such a restrictions checker might contain a large number of restrictions to check. The example restrictions checker looks for violations of two safety-critical guidelines:

a) Short circuit operators are always used
   (i.e., OR ELSE and AND THEN are used and OR and AND are not used).
b) Tasks are declared at the library level.

A procedure, named Process_Element, is created which contains calls to procedures Check_Short_Circuit and Check_Library_Level_Task, which performs a restrictions check for each of our safety-critical guidelines above.

```
procedure Process_Element (Elem       : in Asis.Element;               -- 3.6
                           Control    : in out Asis.Traverse_Control;  -- 3.13
                           Dummy      : in out boolean) is

begin

    Check_Short_Circuit (Elem);
    Check_Library_Level_Task (Elem);

    -- Additional guidelines can be checked here.

end Process_Element;
```

This procedure will process each element in the Context as controlled by an instantiation to Traverse_Element. The body of the Check_Short_Circuit and Check_Library_Level_Task identify the processing to be performed on each Element.

The Check_Short_Circuit procedure is passed the current Element to be evaluated. The Operator_Kinds function identifies the operator kind. If the Element happens to be An_And_Operator or An_Or_Operator, then a violation exists and must be reported by identifying the line number of the violation. Otherwise, there is no processing for this Element.

```
procedure Check_Short_Circuit( Elem : in Asis.Element) is        --  3.6
       Op_Kind : Asis.Operator_Kinds :=                          --  3.9.18
                     Asis.Elements.Operator_Kind (Elem);         -- 13.22

begin
    case Op_Kind is

    when Asis.An_And_Operator =>                                 --  3.9.18
        Put_Line ("Violation of Short Circuit Operator guideline:");
        Put ("-- Use of AND Operator at line ");
        Put (Asis.Text.Line_Number'Wide_Image                    -- 20.2
                        (Asis.Text.First_Line_Number (Elem)));   -- 20.8
        New_Line;

    when Asis.An_Or_Operator =>                                  --  3.9.18
        Put_Line ("Violation of Short Circuit Operator guideline:");
        Put ("-- Use of OR Operator at line ");
        Put (Asis.Text.Line_Number'Wide_Image                    -- 20.2
                  (Asis.Text.First_Line_Number (Elem)));         -- 20.8
        New_Line;

    when others =>
               null;
    end case;

end Check_Short_Circuit;
```

The Check_Library_Level_Task checks to see if the Element parameter is a Declaration_Kind of A_Task_Type_Declaration, A_Protected_Type_Declaration, A_Single_Task_Declaration, or A_Single_Protected_Declaration. If the Element is such a declaration, then a test Is_Library_Level is performed. If the task is not at the Library level, then a violation is reported along with its line number.

```
procedure Check_Library_Level_Task (Elem : Asis.Element) is               --  3.6
begin
    case Asis.Elements.Declaration_Kind (Elem) is                         -- 13.9

        when Asis.A_Task_Type_Declaration |                               --  3.94
             Asis.A_Protected_Type_Declaration |                          --  3.94
             Asis.A_Single_Task_Declaration |                             --  3.94
             Asis.A_Single_Protected_Declaration =>                       --  3.94

                If not Is_Library_Level
                    (Asis.Elements.Enclosing_Compilation_Unit(Elem)) then -- 13.2
                    Put_Line ("Violation of Tasking guideline:");
                    Put ("-- Non-Library Level Task at Line:");
                    Put (Asis.Text.Line_Number'Wide_Image                 -- 20.2
                        (Asis.Text.First_Line_Number (Elem)));            -- 20.8
                    New_Line;
                end if;

        when others =>
            null;
     end case;


end Check_Library_Level_Task;
```

The function Is_Library_Level returns true when the Unit_Class of the Compilation_Unit is A_Public_Declaration.

```
function Is_Library_Level ( CU : Asis.Compilation_Unit )          --  3.10
    return boolean is
begin

    Case Asis.Compilation_Units.Unit_Class (CU) is                -- 10.2
        when Asis.A_Public_Declaration =>                          --  3.12.2
            return true;
        when others =>
            return false;
    end case;

end Is_Library_Level;
```

So far the procedure Process_Element has been created to check restrictions on an Element in a Compilation_Unit. The next step is to traverse all the Elements in a Compilation_Unit with this check. The following package, called Check_Compilation_Unit, does this with its procedure Find_Violations. Find_Violations prints the name of the Unit_Kind and name of each Compilation_Unit as it checks each element in the Compilation_Unit for restrictions using the procedure Check. Procedure Check is an instantiation of ASIS's Traverse_Element with Process_Element, containing the restriction checks. Traverse_Element provides a traversal of each element in a Compilation_Unit's logical syntax tree. The generic is instantiated with a state, a pre-operation, and a post-operation. In this example, Process_Element is the pre-operation which is executed when we land on each Element in the logical syntax tree. In this example, no processing is needed as we leave the Element, so the third generic parameter is the procedure No_Op. The state is not needed by the application. The package Check_Compilation_Unit provides the procedure Find_Violations for the selected Compilation_Unit. It traverses the logical syntax tree, finding and reporting the short circuit violations and task library level violations.

```
with Asis;
package Check_Compilation_Unit is

    procedure Find_Violations (CU : in Asis.Compilation_Unit);         --  3.10

end Check_Compilation_Unit;

with Asis; with Asis.Elements; with Asis.Iterator; with Asis.Text;
with Ada.Wide_Text_Io; use Ada.Wide_Text_Io;
package body Check_Compilation_Unit is

    procedure Process_Element (Elem    : in Asis.Element;              --  3.6
                               Control : in out Asis.Traverse_Control; --  3.13
                               Dummy   : in out boolean);

    procedure No_Op (Elem    : in Asis.Element;                        --  3.6
                     Control : in out Asis.Traverse_Control;           --  3.13
                     Dummy   : in out boolean);

    procedure Check is new Asis.Iterator.Traverse_Element              -- 14.1
                               (boolean, Process_Element, No_Op);

    Procedure Find_Violations (CU : Asis.Compilation_Unit) is          --  3.10
        Control : Asis.Traverse_Control := Asis.Continue;              --  3.13
        Dummy   : boolean;
    begin
        Put_Line ("Processing " &
            Asis.Unit_Kinds'Wide_Image                                 --  3.12.1
               (Asis.Compilation_Units.Unit_Kind (CU))                 -- 10.1
            & ": " &  (Asis.Compilation_Units.Unit_Full_Name (CU)));   -- 10.19
        Check (Asis.Elements.Unit_Declaration (CU), Control, Dummy);
    end Check_Compilation_Unit;
```

The ASIS application is almost complete. A main program is needed which contains the required sequencing of calls to initialize the ASIS interface, name the Ada environment, access the Ada environment, loop through all Compilation_Units in the ASIS Context with the Find_Violations procedure, and closing/releasing all ASIS resources. The Compilation_Units in the Context are placed into the Unit_List. This is achieved with the following main program, called My_Application.

```
with Asis;                                                             --  3.0
with Asis.Implementation;                                              --  6.0
with Asis.Ada_Environments;                                            --  8.0
with Asis.Compilation_Units;                                           -- 10.0
with Check_Compilation_Unit;

procedure My_Application is
    My_Context : Asis.Context;                                         --  3.5

begin
    Asis.Implementation.Initialize;                                    --  6.6
    Asis.Ada_Environments.Associate (My_Context, "My Context");        --  8.3
    Asis.Ada_Environments.Open (My_Context);                           --  8.4

    declare
        Unit_List :  Asis.Compilation_Unit_List :=                     --  3.11
              Asis.Compilation_Units.Compilation_Units (My_Context);   -- 10.10
    begin
        for I in Unit_List'Range loop
            case Asis.Compilation_Units.Unit_Origin (Unit_List (I)) is  -- 10.3
                when Asis.An_Application_Unit =>                       --  3.12.3
                    Check_Compilation_Unit.Find_Violations (Unit_List (I));
                when others => null;
            end case;
        end loop;
    end;

    Asis.Ada_Environments.Close (My_Context);                          --  8.5
    Asis.Ada_Environments.Dissociate (My_Context);                     --  8.6
    Asis.Implementation.Finalize;                                      --  6.8
end My_Application;
```

2.4.4 Erroneous applications

An ASIS application is erroneous if:

- It uses any ASIS query, other than those exported by Asis.Implementation, while Asis.Implementation.Is_Initialized = False.
- It attempts to use a Context before opening it (exceptions to this are Asis.Ada_Environments: Associate, Dissociate, and the Context query functions; these are the only ones that shall be used before opening a Context).
- It attempts to use a Context after closing it (exceptions to this are Asis.Ada_Environments: Associate, Dissociate, and the Context query functions; these are the only ones that shall be used after closing a Context).
- It attempts to Dissociate or Associate an open Context.
- It attempts to manipulate a Compilation_Unit whose Context has been closed.
- It attempts to obtain Compilation_Unit information from an unopened Context.
- It attempts to manipulate an Element from a Closed Context.

2.4.5 Usage rules

2.4.5.1 General usage rules

The following are general usage rules:

- All queries returning list values always return lists with a 'First of one (1).
- All queries with a Context parameter of mode IN raise ASIS_Inappropriate_Context if the Context value is not open. The Status is set to Value_Error.

- All queries with an Element or Line parameter attempt to detect the use of invalid values and raise ASIS_Inappropriate_Element in response. The Status is set to Value_Error. (An invalid value is one where the associated Context variable has been closed.) Not all ASIS implementations are able to detect the use of invalid values. An application that depends upon the success/failure of this invalid value detection is not portable.
- All queries other than simple Boolean or enumeration value queries, raise ASIS_Inappropriate_Element if passed an Element that is not appropriate to the query. The commentary for each query indicates the appropriate element kinds.
- It is generally inappropriate to mix elements of distinct subtypes (e.g., Passing a statement to a query expecting a declaration is inappropriate). It is also inappropriate to mix kinds of elements within a subtype when a query is expecting a specific kind. (i.e., Passing a type declaration to a query expecting a procedure declaration is inappropriate).
- Any query may raise ASIS_Failed with a Status of Obsolete_Reference_Error if the argument or result is itself, or is part of, an inconsistent compilation unit.
- Any Asis.Text query may raise ASIS_Failed with a Status of Text_Error if the program text cannot be located or retrieved for any reason such as renaming, deletion, corruption, or moving of the text or Ada environment.

2.4.5.2 Rules for processing queries for illegal/inconsistent context

The following provides some general rules to identify processing when some inconsistent subset of Compilation Units is processed during semantic query processing. Processing of inconsistent unit sets is in a certain extent implementation-dependent; the following in general should apply:

Semantic queries about elements:

Semantic queries across compilation boundaries may raise an exception if the units are inconsistent.

Semantic queries about compilation units:

- **Corresponding_Children**: If the declaration of a child is inconsistent with the argument of the query, neither declaration nor body is returned. If the declaration of a child is consistent with the argument, but the body is not, the declaration is returned, and for the body, the result of the Corresponding_Body query applied to the declaration is returned.
- **Corresponding_Parent_Declaration:** If a parent is inconsistent with a child passed as the argument, A_Nonexistent_Declaration shall be returned.
- **Corresponding_Declaration:** If the declaration of an argument Element is inconsistent with the argument, A_Nonexistent_Declaration shall be returned. (For a unit A_Procedure_Body or A_Function_Body kind the solution may be in any case to return Nil_Compilation_Unit if the unit is of A_Public_Declaration_And_Body kind.)
- **Corresponding_Body:** If the argument Element requires a body to be presented to make up a complete partition containing this Element, but the Context does not contain the corresponding body, or the body contained in the Context is inconsistent with the argument Element, A_Nonexistent_Body shall be returned.
- **Subunits:** If a subunit is absent or if it is inconsistent with the argument Element, A_Nonexistent_Body shall be returned for it.
- **Corresponding_Subunit_Parent_Body:** If the corresponding body does not exist in the Context, or if it exists, but is inconsistent with the argument Element, then A_Nonexistent_Body shall be returned.

Semantic queries about semantic dependence order:
The Semantic_Dependence_Order query should never raise an exception when processing inconsistent unit (sub)sets. This query is the only means for an application to know if a given unit is consistent with (some of) its supporters (dependents), and therefore the related semantic processing can give valuable results for this unit.

The remaining clauses in this International Standard are presented as a compilable interface, documented with Ada comments.

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

-- 3 package Asis

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
package Asis is
-------------------------------------------------------------------------------
-- Package Asis encapsulates implementation-specific declarations, which are
-- made available to ASIS and its client applications in an
-- implementation-independent manner.
--
-- Package ASIS is the root of the ASIS interface.
--
-------------------------------------------------------------------------------
-- Abstract
--
-- The Ada Semantic Interface Specification (ASIS) is an interface between an
-- Ada environment as defined by ISO/IEC 8652:1995 (the Ada Reference Manual)
-- and any tool requiring information from this environment. An Ada environment
-- includes valuable semantic and syntactic information. ASIS is an open and
-- published callable interface which gives CASE tool and application
-- developers access to this information. ASIS has been designed to be
-- independent of underlying Ada environment implementations, thus supporting
-- portability of software engineering tools while relieving tool developers
-- from having to understand the complexities of an Ada environment’s
-- proprietary internal representation.
--
-------------------------------------------------------------------------------
-- Package ASIS Types:
--
-- The following types are made visible directly through package Asis:
--       type ASIS_Integer
--       type ASIS_Natural
--       type ASIS_Positive
--       type List_Index
--       type Context
--       type Element
--       type Element_List
--       Element subtypes
--       Element Kinds (set of enumeration types)
--       type Compilation_Unit
--       type Compilation_Unit_List
--       Unit Kinds (set of enumeration types)
--       type Traverse_Control
--       subtype Program_Text
--
-- The ASIS interface uses string parameters for many procedure and function
-- calls. Wide_String is used to convey ASIS environment information.
-- Program_Text, a subtype of Wide_String, is used to convey program text.
-- The Ada type String is not used in the ASIS interface. Neither the Ada
-- types Character nor Wide_Character are used in the ASIS interface.
--
-- Implementation_Defined types and values
--
-- A number of implementation-defined types and constants are used. To make
-- the ASIS specification compile, the following types and constants are
-- provided:

    subtype Implementation_Defined_Integer_Type is Integer;
    Implementation_Defined_Integer_Constant : constant := 2**31-1;

-- In addition, there are several implementation-defined private types.
-- For compilation convenience these types have been represented as
-- enumeration types with the single value of "Implementation_Defined".
-- An implementation may define reasonable types and constants.
-- Please refer to commentary where each is used.
--
```

```
-------------------------------------------------------------------------------------
```
-- 3.1 type ASIS_Integer
```
-------------------------------------------------------------------------------------

    subtype ASIS_Integer is Implementation_Defined_Integer_Type;

-------------------------------------------------------------------------------------
--
-- A numeric subtype that allows each ASIS implementation to place constraints
-- on the lower and upper bounds.  Whenever possible, the range of this type
-- should meet or exceed -(2**31-1) .. 2**31-1.
--
-------------------------------------------------------------------------------------
```
-- 3.2 type ASIS_Natural
```
-------------------------------------------------------------------------------------

    subtype ASIS_Natural is ASIS_Integer range 0 .. ASIS_Integer'Last;

-------------------------------------------------------------------------------------
```
-- 3.3 type ASIS_Positive
```
-------------------------------------------------------------------------------------

    subtype ASIS_Positive is ASIS_Integer range 1 .. ASIS_Integer'Last;

-------------------------------------------------------------------------------------
```
-- 3.4 type List_Index
```
-------------------------------------------------------------------------------------

    List_Index_Implementation_Upper :
        constant ASIS_Positive := Implementation_Defined_Integer_Constant;
    subtype List_Index is ASIS_Positive
        range 1 .. List_Index_Implementation_Upper;

-------------------------------------------------------------------------------------
-- List_Index is a numeric subtype used to establish the upper bound for list
-- size.
-------------------------------------------------------------------------------------
```
-- 3.5 type Context
```
-------------------------------------------------------------------------------------
-- The ASIS Context is a view of a particular implementation of an Ada
-- environment.  ASIS requires an application to identify that view of
-- the Ada environment.  An ASIS Context identifies an Ada environment
-- as defined by ISO/IEC 8652:1995.  The Ada environment is well
-- defined for Ada implementations.  ISO/IEC 8652:1995 provides for an
-- implementation-defined method to enter compilation units into the
-- Ada environment.  Implementation permissions allow for illegal and
-- inconsistent units to be in the environment.  The use of ASIS may
-- result in the exception ASIS_Failed being raised if the Ada
-- environment includes such units.
--
-- Defined by the implementation, an ASIS context is a way to identify
-- a set of Compilation Units to be processed by an ASIS application.
-- This may include things such as the pathname, search rules, etc.,
-- which are attributes of the Ada environment and consequently
-- becomes part of the ASIS Context only because it is a "view" of
-- the Ada environment.
--
-- Because the contents of the Ada environment are (Ada-)implementation
-- defined, the ASIS context may contain illegal compilation units.
-- An ASIS Context is a handle to a set of compilation units accessible
-- by an ASIS application.  The set of compilation units available
-- from an ASIS context may be inconsistent, and may contain illegal
-- compilation units.  The contents are selected from the Ada
-- environment as defined by the corresponding Ada Implementation.
-- ASIS should allow multiple open contexts.
--
-- In the Context abstraction, a logical handle is associated with Name and
-- Parameters values that are used by the implementation to identify and
-- connect to the information in the Ada environment.
--
```

```
-- An ASIS Context is associated with some set of Ada compilation units
-- maintained by an underlying Ada implementation or a stand-alone ASIS
-- implementation.  After this association has been made, this set of units
-- is considered to be part of the compile-time Ada environment, which forms
-- the outermost context of any compilation, as specified in section 10.1.4 of
-- the Ada Reference Manual.  This same environment context provides the
-- implicit outermost anonymous task during program execution.
--
-- Some implementations might not need explicit Name and/or Parameters values to
-- identify their Ada environment.  Other implementations might choose to
-- implement the Ada environment as a single external file in which case the
-- name and parameters values might simply supply the Name, Form, and any other
-- values needed to open such a file.
--
-------------------------------------------------------------------------------------
-- Context shall be an undiscriminated limited private.
-------------------------------------------------------------------------------------

    type Context is limited private;
    Nil_Context : constant Context;

    function "=" (Left  : in Context;
                  Right : in Context)
                  Return Boolean is abstract;

-------------------------------------------------------------------------------------
--
--|IR Implementation Requirement
--|IR
--|IR The concrete mechanism of this association is implementation-specific:
--|IR
--|IR Each ASIS implementation provides the means to construct an ASIS
--|IR Context value that defines the environment declarative_part or
--|IR "context" from which ASIS can obtain library units.
--
-------------------------------------------------------------------------------------
```

-- 3.6 type Element

```
-------------------------------------------------------------------------------------
-- The Ada lexical element abstraction (a private type).
--
-- The Element type is a distinct abstract type representing handles for the
-- lexical elements that form the text of compilation units.  Elements deal
-- with the internal or "textual" view of compilation units.
--
-- Operations are provided that split a Compilation_Unit object into one
-- Element and two Element lists:
--
-- a) A context clause represented by an Element_List containing
--    with clauses, use clauses, and pragmas.
--
-- b) An Element associated with the declaration.
--
-- c) A list of pragmas, that are not part of the context clause but which
--    nonetheless affect the compilation of the unit.
--
-------------------------------------------------------------------------------------
-- ASIS Elements are representations of the syntactic and semantic information
-- available from most Ada environments.
--
-- The ASIS Element type shall be an undiscriminated private type.
-------------------------------------------------------------------------------------

    type Element is private;
    Nil_Element : constant Element;

    function "=" (Left  : in Element;
                  Right : in Element)
                  Return Boolean is abstract;
```

```
-------------------------------------------------------------------------------------
```

-- 3.7 type Element_List

```
-------------------------------------------------------------------------------------

    type Element_List is array (List_Index range <>) of Element;

    Nil_Element_List : constant Element_List;

-------------------------------------------------------------------------------------
```

-- 3.8 subtypes of Element and Element_List

```
-------------------------------------------------------------------------------------

    subtype Access_Type_Definition          is Element;
    subtype Association                     is Element;
    subtype Association_List                is Element_List;
    subtype Case_Statement_Alternative      is Element;
    subtype Clause                          is Element;
    subtype Component_Clause                is Element;
    subtype Component_Clause_List           is Element_List;
    subtype Component_Declaration           is Element;
    subtype Component_Definition            is Element;
    subtype Constraint                      is Element;
    subtype Context_Clause                  is Element;
    subtype Context_Clause_List             is Element_List;
    subtype Declaration                     is Element;
    subtype Declaration_List                is Element_List;
    subtype Declarative_Item_List           is Element_List;
    subtype Definition                      is Element;
    subtype Definition_List                 is Element_List;
    subtype Discrete_Range                  is Element;
    subtype Discrete_Range_List             is Element_List;
    subtype Discrete_Subtype_Definition     is Element;
    subtype Discriminant_Association        is Element;
    subtype Discriminant_Association_List   is Element_List;
    subtype Discriminant_Specification_List is Element_List;
    subtype Defining_Name                   is Element;
    subtype Defining_Name_List              is Element_List;
    subtype Exception_Handler               is Element;
    subtype Exception_Handler_List          is Element_List;
    subtype Expression                      is Element;
    subtype Expression_List                 is Element_List;
    subtype Formal_Type_Definition          is Element;
    subtype Generic_Formal_Parameter        is Element;
    subtype Generic_Formal_Parameter_List   is Element_List;
    subtype Identifier                      is Element;
    subtype Identifier_List                 is Element_List;
    subtype Name                            is Element;
    subtype Name_List                       is Element_List;
    subtype Parameter_Specification         is Element;
    subtype Parameter_Specification_List    is Element_List;
    subtype Path                            is Element;
    subtype Path_List                       is Element_List;
    subtype Pragma_Element                  is Element;
    subtype Pragma_Element_List             is Element_List;
    subtype Range_Constraint                is Element;
    subtype Record_Component                is Element;
    subtype Record_Component_List           is Element_List;
    subtype Record_Definition               is Element;
    subtype Representation_Clause           is Element;
    subtype Representation_Clause_List      is Element_List;
    subtype Root_Type_Definition            is Element;
    subtype Select_Alternative              is Element;
    subtype Statement                       is Element;
    subtype Statement_List                  is Element_List;
    subtype Subtype_Indication              is Element;
    subtype Subtype_Mark                    is Element;
    subtype Type_Definition                 is Element;
    subtype Variant                         is Element;
    subtype Variant_Component_List          is Element_List;
    subtype Variant_List                    is Element_List;

--
-------------------------------------------------------------------------------------
```

```
-----------------------------------------------------------------------------------------
```

-- 3.9 Element Kinds

```
-----------------------------------------------------------------------------------------
-- Element Kinds are enumeration types describing various kinds of elements.
-- These element kinds are only used by package Asis.Elements.
-----------------------------------------------------------------------------------------
```

-- 3.9.1 type Element_Kinds

```
-----------------------------------------------------------------------------------------
-- Element_Kinds Hierarchy
--
-- ASIS offers hierarchical classification of elements.  At the highest
-- level, the Element_Kinds type provides literals that define "kinds" or
-- classes listed below into which all non-nil elements are grouped.  Elements
-- in each of the Element_Kinds classes, with the exception of
-- An_Exception_Handler, can be further classified by a subordinate kind at
-- the next level in the hierarchy.  Several subordinate kinds also have
-- additional subordinate kinds.
--
-- For example, Element_Kinds'A_Declaration might be classified into
-- Declaration_Kinds'A_Parameter_Specification which might be further
-- classified into Trait_Kinds'An_Access_Definition_Trait.
-- This fully identifies the syntax of an element such as:
--
--       (Who : access Person)
--
-- All Element_Kinds and subordinate kinds Queries are in Asis.Elements.
--
-- It is not necessary to strictly follow the hierarchy; any element can be
-- classified by any subordinate kind from any level.  However, meaningful
-- results will only be obtained from subordinate kinds that are appropriate.
-- These are designated within the hierarchy shown below:
--
--        Element_Kinds         -> Subordinate Kinds
-----------------------------------------------------------------------------------------
--   Key: Read "->" as "is further classified by its"
--
--        A_Pragma              -> Pragma_Kinds
--
--        A_Defining_Name       -> Defining_Name_Kinds
--                                          -> Operator_Kinds
--
--        A_Declaration         -> Declaration_Kinds
--                                          -> Trait_Kinds
--                                          -> Declaration_Origins
--                                          -> Mode_Kinds
--                                          -> Subprogram_Default_Kinds
--
--        A_Definition          -> Definition_Kinds
--                                          -> Trait_Kinds
--                                          -> Type_Kinds
--                                                     -> Trait_Kinds
--                                          -> Formal_Type_Kinds
--                                                     -> Trait_Kinds
--                                          -> Access_Type_Kinds
--                                          -> Root_Type_Kinds
--                                          -> Constraint_Kinds
--                                          -> Discrete_Range_Kinds
--
--        An_Expression         -> Expression_Kinds
--                                          -> Operator_Kinds
--                                          -> Attribute_Kinds
--
--        An_Association        -> Association_Kinds
--
--        A_Statement           -> Statement_Kinds
--
--        A_Path                -> Path_Kinds
--
--        A_Clause              -> Clause_Kinds
--                                          -> Representation_Clause_Kinds
--
--        An_Exception_Handler
--
```

```
-------------------------------------------------------------------------------------
-- Element_Kinds - general element classifications
--  Literals                  -- ASIS package with queries for these kinds
-------------------------------------------------------------------------------------

  type Element_Kinds is (

    Not_An_Element,           -- Nil_Element

    A_Pragma,                 -- Asis.Elements

    A_Defining_Name,          -- Asis.Declarations

    A_Declaration,            -- Asis.Declarations

    A_Definition,             -- Asis.Definitions

    An_Expression,            -- Asis.Expressions

    An_Association,           -- Asis.Expressions

    A_Statement,              -- Asis.Statements

    A_Path,                   -- Asis.Statements

    A_Clause,                 -- Asis.Clauses

    An_Exception_Handler);    -- Asis.Statements

-------------------------------------------------------------------------------------
```

-- 3.9.2 type Pragma_Kinds

```
-------------------------------------------------------------------------------------
-- Pragma_Kinds - classifications for pragmas
--  Literals                         -- Reference Manual
-------------------------------------------------------------------------------------

  type Pragma_Kinds is (

    Not_A_Pragma,                    -- An unexpected element
    An_All_Calls_Remote_Pragma,      -- E.2.3(5)
    An_Asynchronous_Pragma,          -- E.4.1(3)
    An_Atomic_Pragma,                -- C.6(3)
    An_Atomic_Components_Pragma,     -- C.6(5)
    An_Attach_Handler_Pragma,        -- C.3.1(4)
    A_Controlled_Pragma,             -- 13.11.3(3)
    A_Convention_Pragma,             -- B.1(7), M.1(5)
    A_Discard_Names_Pragma,          -- C.5(3)
    An_Elaborate_Pragma,             -- 10.2.1(20)
    An_Elaborate_All_Pragma,         -- 10.2.1(21)
    An_Elaborate_Body_Pragma,        -- 10.2.1(22)
    An_Export_Pragma,                -- B.1(5), M.1(5)
    An_Import_Pragma,                -- B.1(6), M.1(5)
    An_Inline_Pragma,                -- 6.3.2(3)
    An_Inspection_Point_Pragma,      -- H.3.2(3)
    An_Interrupt_Handler_Pragma,     -- C.3.1(2)
    An_Interrupt_Priority_Pragma,    -- D.1(5)
    A_Linker_Options_Pragma,         -- B.1(8)
    A_List_Pragma,                   -- 2.8(21)
    A_Locking_Policy_Pragma,         -- D.3(3)
    A_Normalize_Scalars_Pragma,      -- H.1(3)
    An_Optimize_Pragma,              -- 2.8(23)
    A_Pack_Pragma,                   -- 13.2(3)
    A_Page_Pragma,                   -- 2.8(22)
    A_Preelaborate_Pragma,           -- 10.2.1(3)
    A_Priority_Pragma,               -- D.1(3)
    A_Pure_Pragma,                   -- 10.2.1(14)
    A_Queuing_Policy_Pragma,         -- D.4(3)
    A_Remote_Call_Interface_Pragma,  -- E.2.3(3)
    A_Remote_Types_Pragma,           -- E.2.2(3)
    A_Restrictions_Pragma,           -- 13.12(3)
    A_Reviewable_Pragma,             -- H.3.1(3)
    A_Shared_Passive_Pragma,         -- E.2.1(3)
    A_Storage_Size_Pragma,           -- 13.3(63)
```

```
    A_Suppress_Pragma,                 -- 11.5(4)
    A_Task_Dispatching_Policy_Pragma, -- D.2.2(2)
    A_Volatile_Pragma,                 -- C.6(4)
    A_Volatile_Components_Pragma,      -- C.6(6)

    An_Implementation_Defined_Pragma, -- 2.8(14)

    An_Unknown_Pragma);                -- Unknown to ASIS

-------------------------------------------------------------------------------------
```

-- 3.9.3 type Defining_Name_Kinds

```
-------------------------------------------------------------------------------------
-- Defining_Name_Kinds - names defined by declarations and specifications.
--  Literals                                 -- Reference Manual
-------------------------------------------------------------------------------------

  type Defining_Name_Kinds is (

    Not_A_Defining_Name,                     -- An unexpected element

    A_Defining_Identifier,                   -- 3.1(4)
    A_Defining_Character_Literal,            -- 3.5.1(4)
    A_Defining_Enumeration_Literal,          -- 3.5.1(3)
    A_Defining_Operator_Symbol,              -- 6.1(9)
    A_Defining_Expanded_Name);               -- 6.1(7)
                                             -- program unit name defining_identifier

-------------------------------------------------------------------------------------
```

-- 3.9.4 type Declaration_Kinds

```
-------------------------------------------------------------------------------------
-- Declaration_Kinds - declarations and specifications having defining name literals.
--  Literals                               -- Reference Manual -> Subordinate Kinds
-------------------------------------------------------------------------------------

  type Declaration_Kinds is (

    Not_A_Declaration,                     -- An unexpected element

    An_Ordinary_Type_Declaration,          -- 3.2.1(3)
      -- a full_type_declaration of the form:
      -- type defining_identifier [known_discriminant_part] is type_definition;

    A_Task_Type_Declaration,               -- 9.1(2)
    A_Protected_Type_Declaration,          -- 9.4(2)
    An_Incomplete_Type_Declaration,        -- 3.2.1(2),3.10(2)
    A_Private_Type_Declaration,            -- 3.2.1(2),7.3(2) -> Trait_Kinds
    A_Private_Extension_Declaration,       -- 3.2.1(2),7.3(3) -> Trait_Kinds

    A_Subtype_Declaration,                 -- 3.2.2(2)

    A_Variable_Declaration,                -- 3.3.1(2) -> Trait_Kinds
    A_Constant_Declaration,                -- 3.3.1(4) -> Trait_Kinds
    A_Deferred_Constant_Declaration,       -- 3.3.1(6),7.4(2) -> Trait_Kinds
    A_Single_Task_Declaration,             -- 3.3.1(2),9.1(3)
    A_Single_Protected_Declaration,        -- 3.3.1(2),9.4(2)

    An_Integer_Number_Declaration,         -- 3.3.2(2)
    A_Real_Number_Declaration,             -- 3.5.6(2)

    An_Enumeration_Literal_Specification,  -- 3.5.1(3)

    A_Discriminant_Specification,          -- 3.7(5)   -> Trait_Kinds
    A_Component_Declaration,               -- 3.8(6)

    A_Loop_Parameter_Specification,        -- 5.5(4)   -> Trait_Kinds

    A_Procedure_Declaration,               -- 6.1(4)   -> Trait_Kinds
    A_Function_Declaration,                -- 6.1(4)   -> Trait_Kinds

    A_Parameter_Specification,             -- 6.1(15)  -> Trait_Kinds
                                           --          -> Mode_Kinds
    A_Procedure_Body_Declaration,          -- 6.3(2)
    A_Function_Body_Declaration,           -- 6.3(2)
```

```
    A_Package_Declaration,                  -- 7.1(2)
    A_Package_Body_Declaration,             -- 7.2(2)

    An_Object_Renaming_Declaration,         -- 8.5.1(2)
    An_Exception_Renaming_Declaration,      -- 8.5.2(2)
    A_Package_Renaming_Declaration,         -- 8.5.3(2)
    A_Procedure_Renaming_Declaration,       -- 8.5.4(2)
    A_Function_Renaming_Declaration,        -- 8.5.4(2)
    A_Generic_Package_Renaming_Declaration, -- 8.5.5(2)
    A_Generic_Procedure_Renaming_Declaration,-- 8.5.5(2)
    A_Generic_Function_Renaming_Declaration, -- 8.5.5(2)

    A_Task_Body_Declaration,                -- 9.1(6)
    A_Protected_Body_Declaration,           -- 9.4(7)
    An_Entry_Declaration,                   -- 9.5.2(2)
    An_Entry_Body_Declaration,              -- 9.5.2(5)
    An_Entry_Index_Specification,           -- 9.5.2(2)

    A_Procedure_Body_Stub,                  -- 10.1.3(3)
    A_Function_Body_Stub,                   -- 10.1.3(3)
    A_Package_Body_Stub,                    -- 10.1.3(4)
    A_Task_Body_Stub,                       -- 10.1.3(5)
    A_Protected_Body_Stub,                  -- 10.1.3(6)

    An_Exception_Declaration,               -- 11.1(2)
    A_Choice_Parameter_Specification,       -- 11.2(4)

    A_Generic_Procedure_Declaration,        -- 12.1(2)
    A_Generic_Function_Declaration,         -- 12.1(2)
    A_Generic_Package_Declaration,          -- 12.1(2)

    A_Package_Instantiation,                -- 12.3(2)
    A_Procedure_Instantiation,              -- 12.3(2)
    A_Function_Instantiation,               -- 12.3(2)

    A_Formal_Object_Declaration,            -- 12.4(2)  -> Mode_Kinds
    A_Formal_Type_Declaration,              -- 12.5(2)
    A_Formal_Procedure_Declaration,         -- 12.6(2)  -> Subprogram_Default_Kinds
    A_Formal_Function_Declaration,          -- 12.6(2)  -> Subprogram_Default_Kinds
    A_Formal_Package_Declaration,           -- 12.7(2)
    A_Formal_Package_Declaration_With_Box); -- 12.7(3)

-- The following Declaration_Kinds subtypes are not used by ASIS but are
-- provided for the convenience of the ASIS implementor:

  subtype A_Type_Declaration is Declaration_Kinds range
            An_Ordinary_Type_Declaration .. A_Private_Extension_Declaration;

  subtype A_Full_Type_Declaration is Declaration_Kinds range
            An_Ordinary_Type_Declaration .. A_Protected_Type_Declaration;

  subtype An_Object_Declaration is Declaration_Kinds range
            A_Variable_Declaration .. A_Single_Protected_Declaration;

  subtype A_Number_Declaration is Declaration_Kinds range
            An_Integer_Number_Declaration .. A_Real_Number_Declaration;

  subtype A_Renaming_Declaration is Declaration_Kinds range
            An_Object_Renaming_Declaration ..
            A_Generic_Function_Renaming_Declaration;

  subtype A_Body_Stub is Declaration_Kinds range
            A_Procedure_Body_Stub .. A_Protected_Body_Stub;

  subtype A_Generic_Declaration is Declaration_Kinds range
            A_Generic_Procedure_Declaration .. A_Generic_Package_Declaration;

  subtype A_Generic_Instantiation is Declaration_Kinds range
            A_Package_Instantiation .. A_Function_Instantiation;

  subtype A_Formal_Declaration is Declaration_Kinds range
            A_Formal_Object_Declaration ..
            A_Formal_Package_Declaration_With_Box;
```

```
-------------------------------------------------------------------------------
```

-- 3.9.5 type Trait_Kinds

```
-------------------------------------------------------------------------------
--
-- Trait_Kinds provide a means of further classifying the syntactic structure
-- or "trait" of certain A_Declaration and A_Definition elements.
-- Trait_Kinds are determined only by the presence (or absence) of certain
-- reserved words.  The semantics of an element are not considered.
-- The reserved words of interest here are "abstract", "aliased", "limited",
-- "private", "reverse", and "access" when it appears in an access_definition.
-- Trait_Kinds enumerates all combinations useful in this classification.
--
-- For example, A_Variable_Declaration element that is semantically a
-- limited type because its components are of a limited type is
-- An_Ordinary_Trait, not A_Limited_Trait, since the reserved word "limited"
-- does not appear in its declaration or definition.
--
-- The subordinate Trait_Kinds allow Declaration_Kinds and Definition_Kinds
-- to enumerate fewer higher level elements, and be less cluttered by all
-- possible permutations of syntactic possibilities. For example, in the case
-- of a record_type_definition, Definition_Kinds can provide just two literals
-- that differentiate between ordinary record types and tagged record types:
--
--     A_Record_Type_Definition,              -- 3.8(2)    -> Trait_Kinds
--     A_Tagged_Record_Type_Definition,       -- 3.8(2)    -> Trait_Kinds
--
-- The remaining classification can be accomplished, if desired, using
-- Trait_Kinds to determine if the definition is abstract, or limited, or both.
-- Without Trait_Kinds, Definition_Kinds needs six literals to identify
-- all the syntactic combinations for a record_type_definition.
--
-- Elements expected by the Trait_Kind query are any Declaration_Kinds or
-- Definition_Kinds for which Trait_Kinds is a subordinate kind: the literal
-- definition has "-> Trait_Kinds" following it.  For example, the
-- definitions of:
--
--     A_Discriminant_Specification,              -- 3.7(5)   -> Trait_Kinds
--     A_Component_Declaration,                   -- 3.8(6)
--
-- indicate A_Discriminant_Specification is an expected kind while
-- A_Component_Declaration is unexpected.
--
-- All Declaration_Kinds and Definition_Kinds for which Trait_Kinds is not a
-- subordinate kind, and all other Element_Kinds, are unexpected and are
-- Not_A_Trait.
--
-- An_Ordinary_Trait is any expected element whose syntax does not explicitly
-- contain any of the reserved words listed above.
--
-------------------------------------------------------------------------------
-- Trait_Kinds
--  Literals
-------------------------------------------------------------------------------

  type Trait_Kinds is (

    Not_A_Trait,                        -- An unexpected element

    An_Ordinary_Trait,                  -- The declaration or definition does
                                        -- not contain the reserved words
                                        -- "aliased", "reverse", "private",
                                        -- "limited", "abstract", or
                                        -- "access" in an access_definition

    An_Aliased_Trait,                   -- "aliased" is present
    An_Access_Definition_Trait,         -- "access" in an access_definition is present
    A_Reverse_Trait,                    -- "reverse" is present
    A_Private_Trait,                    -- Only "private" is present
    A_Limited_Trait,                    -- Only "limited" is present
    A_Limited_Private_Trait,            -- "limited" and "private" are present
```

```
    An_Abstract_Trait,                    -- Only "abstract" is present
    An_Abstract_Private_Trait,            -- "abstract" and "private" are present
    An_Abstract_Limited_Trait,            -- "abstract" and "limited" are present
    An_Abstract_Limited_Private_Trait);   -- "abstract", "limited", and "private" are
                                          -- present

-------------------------------------------------------------------------------------
```

-- 3.9.6 type Declaration_Origins

```
-------------------------------------------------------------------------------------
-- Declaration_Origins
--  Literals                              -- Reference Manual
-------------------------------------------------------------------------------------

  type Declaration_Origins is (

    Not_A_Declaration_Origin,             -- An unexpected element

    An_Explicit_Declaration,              -- 3.1(5) explicitly declared in
                                          -- the text of a program, or within
                                          -- an expanded generic template
    An_Implicit_Predefined_Declaration,   -- 3.1(5), 3.2.3(1), A.1(2)
    An_Implicit_Inherited_Declaration);   -- 3.1(5), 3.4(6-35)

-------------------------------------------------------------------------------------
```

-- 3.9.7 type Mode_Kinds

```
-------------------------------------------------------------------------------------
-- Mode_Kinds
--  Literals                -- Reference Manual
-------------------------------------------------------------------------------------

  type Mode_Kinds is (      -- 6.1

    Not_A_Mode,             -- An unexpected element

    A_Default_In_Mode,      -- procedure A(B :        C);
    An_In_Mode,             -- procedure A(B : IN     C);
    An_Out_Mode,            -- procedure A(B :    OUT C);
    An_In_Out_Mode);        -- procedure A(B : IN OUT C);

-------------------------------------------------------------------------------------
```

-- 3.9.8 type Subprogram_Default_Kinds

```
-------------------------------------------------------------------------------------
-- Subprogram_Default_Kinds
--  Literals                -- Reference Manual
-------------------------------------------------------------------------------------

  type Subprogram_Default_Kinds is (    -- 12.6

    Not_A_Default,          -- An unexpected element

    A_Name_Default,         -- with subprogram_specification is default_name;
    A_Box_Default,          -- with subprogram_specification is <>;
    A_Nil_Default);         -- with subprogram_specification;

-------------------------------------------------------------------------------------
```

-- 3.9.9 type Definition_Kinds

```
-------------------------------------------------------------------------------------
-- Definition_Kinds
--  Literals                           -- Reference Manual   -> Subordinate Kinds
-------------------------------------------------------------------------------------

  type Definition_Kinds is (

    Not_A_Definition,                  -- An unexpected element

    A_Type_Definition,                 -- 3.2.1(4)    -> Type_Kinds

    A_Subtype_Indication,              -- 3.2.2(3)
    A_Constraint,                      -- 3.2.2(5)    -> Constraint_Kinds

    A_Component_Definition,            -- 3.6(7)      -> Trait_Kinds
```

```
    A_Discrete_Subtype_Definition,    -- 3.6(6)      -> Discrete_Range_Kinds
    A_Discrete_Range,                 -- 3.6.1(3)    -> Discrete_Range_Kinds

    An_Unknown_Discriminant_Part,     -- 3.7(3)
    A_Known_Discriminant_Part,        -- 3.7(2)

    A_Record_Definition,              -- 3.8(3)
    A_Null_Record_Definition,         -- 3.8(3)

    A_Null_Component,                 -- 3.8(4)
    A_Variant_Part,                   -- 3.8.1(2)
    A_Variant,                        -- 3.8.1(3)

    An_Others_Choice,                 -- 3.8.1(5), 4.3.1(5), 4.3.3(5), 11.2(5)

    A_Private_Type_Definition,        -- 7.3(2)      -> Trait_Kinds
    A_Tagged_Private_Type_Definition, -- 7.3(2)      -> Trait_Kinds
    A_Private_Extension_Definition,   -- 7.3(3)      -> Trait_Kinds

    A_Task_Definition,                -- 9.1(4)
    A_Protected_Definition,           -- 9.4(4)

    A_Formal_Type_Definition);        -- 12.5(3)     -> Formal_Type_Kinds
```

-- 3.9.10 type Type_Kinds

```
----------------------------------------------------------------------------------------
-- Type_Kinds
--  Literals                               -- Reference Manual  -> Subordinate Kinds
----------------------------------------------------------------------------------------

  type Type_Kinds is (

    Not_A_Type_Definition,                 -- An unexpected element

    A_Derived_Type_Definition,             -- 3.4(2)     -> Trait_Kinds
    A_Derived_Record_Extension_Definition, -- 3.4(2)     -> Trait_Kinds

    An_Enumeration_Type_Definition,        -- 3.5.1(2)

    A_Signed_Integer_Type_Definition,      -- 3.5.4(3)
    A_Modular_Type_Definition,             -- 3.5.4(4)

    A_Root_Type_Definition,                -- 3.5.4(14), 3.5.6(3)
                                           --        -> Root_Type_Kinds
    A_Floating_Point_Definition,           -- 3.5.7(2)

    An_Ordinary_Fixed_Point_Definition,    -- 3.5.9(3)
    A_Decimal_Fixed_Point_Definition,      -- 3.5.9(6)

    An_Unconstrained_Array_Definition,     -- 3.6(2)
    A_Constrained_Array_Definition,        -- 3.6(2)

    A_Record_Type_Definition,              -- 3.8(2)     -> Trait_Kinds
    A_Tagged_Record_Type_Definition,       -- 3.8(2)     -> Trait_Kinds

    An_Access_Type_Definition);            -- 3.10(2)    -> Access_Type_Kinds
```

-- 3.9.11 type Formal_Type_Kinds

```
----------------------------------------------------------------------------------------
-- Formal_Type_Kinds
--  Literals                                  -- Reference Manual  -> Subordinate Kinds
----------------------------------------------------------------------------------------

  type Formal_Type_Kinds is (

    Not_A_Formal_Type_Definition,             -- An unexpected element

    A_Formal_Private_Type_Definition,         -- 12.5.1(2)   -> Trait_Kinds
    A_Formal_Tagged_Private_Type_Definition,  -- 12.5.1(2)   -> Trait_Kinds

    A_Formal_Derived_Type_Definition,         -- 12.5.1(3)   -> Trait_Kinds
```

```
    A_Formal_Discrete_Type_Definition,        -- 12.5.2(2)

    A_Formal_Signed_Integer_Type_Definition,  -- 12.5.2(3)
    A_Formal_Modular_Type_Definition,         -- 12.5.2(4)

    A_Formal_Floating_Point_Definition,       -- 12.5.2(5)

    A_Formal_Ordinary_Fixed_Point_Definition, -- 12.5.2(6)
    A_Formal_Decimal_Fixed_Point_Definition,  -- 12.5.2(7)

    A_Formal_Unconstrained_Array_Definition,  -- 3.6(3)
    A_Formal_Constrained_Array_Definition,    -- 3.6(5)

    A_Formal_Access_Type_Definition);         -- 3.10(3),3.10(5)
                                              --         -> Access_Type_Kinds

-------------------------------------------------------------------------------------
```

-- 3.9.12 type Access_Type_Kinds

```
-------------------------------------------------------------------------------------
-- Access_Type_Kinds
--  Literals                             -- Reference Manual
-------------------------------------------------------------------------------------

  type Access_Type_Kinds is ( -- 3.10

    Not_An_Access_Type_Definition,       -- An unexpected element

    A_Pool_Specific_Access_To_Variable,  -- access subtype_indication
    An_Access_To_Variable,               -- access all subtype_indication
    An_Access_To_Constant,               -- access constant subtype_indication

    An_Access_To_Procedure,              -- access procedure
    An_Access_To_Protected_Procedure,    -- access protected procedure
    An_Access_To_Function,               -- access function
    An_Access_To_Protected_Function);    -- access protected function

-- The following Access_Type_Kinds subtypes are not used by ASIS but are
-- provided for the convenience of the ASIS implementor:

  subtype Access_To_Object_Definition is Access_Type_Kinds range
          A_Pool_Specific_Access_To_Variable .. An_Access_To_Constant;

  subtype Access_To_Subprogram_Definition is Access_Type_Kinds range
          An_Access_To_Procedure ..  An_Access_To_Protected_Function;

-------------------------------------------------------------------------------------
```

-- 3.9.13 type Root_Type_Kinds

```
-------------------------------------------------------------------------------------
-- Root_Type_Kinds
--  Literals                               -- Reference Manual
-------------------------------------------------------------------------------------

  type Root_Type_Kinds is (

    Not_A_Root_Type_Definition,            -- An unexpected element

    A_Root_Integer_Definition,             -- 3.4.1(8)
    A_Root_Real_Definition,                -- 3.4.1(8)

    A_Universal_Integer_Definition,        -- 3.4.1(6)
    A_Universal_Real_Definition,           -- 3.4.1(6)
    A_Universal_Fixed_Definition);         -- 3.4.1(6)
```

```
---------------------------------------------------------------------------------------
```

-- 3.9.14 type Constraint_Kinds

```
---------------------------------------------------------------------------------------
-- Constraint_Kinds
--  Literals                               -- Reference Manual
---------------------------------------------------------------------------------------

  type Constraint_Kinds is (

    Not_A_Constraint,                      -- An unexpected element

    A_Range_Attribute_Reference,           -- 3.5(2)
    A_Simple_Expression_Range,             -- 3.2.2, 3.5(3)
    A_Digits_Constraint,                   -- 3.2.2, 3.5.9
    A_Delta_Constraint,                    -- 3.2.2, J.3
    An_Index_Constraint,                   -- 3.2.2, 3.6.1
    A_Discriminant_Constraint);            -- 3.2.2

---------------------------------------------------------------------------------------
```

-- 3.9.15 type Discrete_Range_Kinds

```
---------------------------------------------------------------------------------------
-- Discrete_Range_Kinds
--  Literals                               -- Reference Manual
---------------------------------------------------------------------------------------

  type Discrete_Range_Kinds is (

    Not_A_Discrete_Range,                  -- An unexpected element

    A_Discrete_Subtype_Indication,         -- 3.6.1(6), 3.2.2
    A_Discrete_Range_Attribute_Reference,  -- 3.6.1, 3.5
    A_Discrete_Simple_Expression_Range);   -- 3.6.1, 3.5

---------------------------------------------------------------------------------------
```

-- 3.9.16 type Association_Kinds

```
---------------------------------------------------------------------------------------
-- Association_Kinds
--  Literals                               -- Reference Manual
---------------------------------------------------------------------------------------

  type Association_Kinds is (

    Not_An_Association,                    -- An unexpected element

    A_Pragma_Argument_Association,         -- 2.8
    A_Discriminant_Association,            -- 3.7.1
    A_Record_Component_Association,        -- 4.3.1
    An_Array_Component_Association,        -- 4.3.3
    A_Parameter_Association,               -- 6.4
    A_Generic_Association);                -- 12.3

---------------------------------------------------------------------------------------
```

-- 3.9.17 type Expression_Kinds

```
---------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------
-- Expression_Kinds - general expression classifications
--  Literals                                   -- Reference Manual -> Subordinate Kinds
---------------------------------------------------------------------------------------

  type Expression_Kinds is (

    Not_An_Expression,                         -- An unexpected element

    An_Integer_Literal,                        -- 2.4
    A_Real_Literal,                            -- 2.4.1
    A_String_Literal,                          -- 2.6

    An_Identifier,                             -- 4.1
    An_Operator_Symbol,                        -- 4.1
    A_Character_Literal,                       -- 4.1
    An_Enumeration_Literal,                    -- 4.1
```

```
    An_Explicit_Dereference,                  -- 4.1
    A_Function_Call,                          -- 4.1

    An_Indexed_Component,                     -- 4.1.1
    A_Slice,                                  -- 4.1.2
    A_Selected_Component,                     -- 4.1.3
    An_Attribute_Reference,                   -- 4.1.4  -> Attribute_Kinds

    A_Record_Aggregate,                       -- 4.3
    An_Extension_Aggregate,                   -- 4.3
    A_Positional_Array_Aggregate,             -- 4.3
    A_Named_Array_Aggregate,                  -- 4.3

    An_And_Then_Short_Circuit,                -- 4.4
    An_Or_Else_Short_Circuit,                 -- 4.4

    An_In_Range_Membership_Test,              -- 4.4
    A_Not_In_Range_Membership_Test,           -- 4.4
    An_In_Type_Membership_Test,               -- 4.4
    A_Not_In_Type_Membership_Test,            -- 4.4

    A_Null_Literal,                           -- 4.4
    A_Parenthesized_Expression,               -- 4.4

    A_Type_Conversion,                        -- 4.6
    A_Qualified_Expression,                   -- 4.7

    An_Allocation_From_Subtype,               -- 4.8
    An_Allocation_From_Qualified_Expression); -- 4.8

------------------------------------------------------------------------------------
```

-- 3.9.18 type Operator_Kinds

```
------------------------------------------------------------------------------------
-- Operator_Kinds - classification of the various Ada predefined operators
--  Literals                           -- Reference Manual
------------------------------------------------------------------------------------

  type Operator_Kinds is (             -- 4.5

    Not_An_Operator,                   -- An unexpected element

    An_And_Operator,                   -- and
    An_Or_Operator,                    -- or
    An_Xor_Operator,                   -- xor
    An_Equal_Operator,                 -- =
    A_Not_Equal_Operator,              -- /=
    A_Less_Than_Operator,              -- <
    A_Less_Than_Or_Equal_Operator,     -- <=
    A_Greater_Than_Operator,           -- >
    A_Greater_Than_Or_Equal_Operator,  -- >=
    A_Plus_Operator,                   -- +
    A_Minus_Operator,                  -- -
    A_Concatenate_Operator,            -- &
    A_Unary_Plus_Operator,             -- +
    A_Unary_Minus_Operator,            -- -
    A_Multiply_Operator,               -- *
    A_Divide_Operator,                 -- /
    A_Mod_Operator,                    -- mod
    A_Rem_Operator,                    -- rem
    An_Exponentiate_Operator,          -- **
    An_Abs_Operator,                   -- abs
    A_Not_Operator);                   -- not
```

```
------------------------------------------------------------------------------------------
```

-- 3.9.19 type Attribute_Kinds

```
------------------------------------------------------------------------------------------
-- Attribute_Kinds - classifications for all known Ada attributes
--  Literals                       -- Reference Manual
------------------------------------------------------------------------------------------

  type Attribute_Kinds is (

    Not_An_Attribute,              -- An unexpected element

    An_Access_Attribute,           -- 3.10.2(24), 3.10.2(32), K(2), K(4)
    An_Address_Attribute,          -- 13.3(11), J.7.1(5), K(6)
    An_Adjacent_Attribute,         -- A.5.3(48), K(8)
    An_Aft_Attribute,              -- 3.5.10(5), K(12)
    An_Alignment_Attribute,        -- 13.3(23), K(14)
    A_Base_Attribute,              -- 3.5(15), K(17)
    A_Bit_Order_Attribute,         -- 13.5.3(4), K(19)
    A_Body_Version_Attribute,      -- E.3(4), K(21)
    A_Callable_Attribute,          -- 9.9(2), K(23)
    A_Caller_Attribute,            -- C.7.1(14), K(25)
    A_Ceiling_Attribute,           -- A.5.3(33), K(27)
    A_Class_Attribute,             -- 3.9(14), 7.3.1(9), K(31), K(34)
    A_Component_Size_Attribute,    -- 13.3(69), K(36)
    A_Compose_Attribute,           -- A.5.3(24), K(38)
    A_Constrained_Attribute,       -- 3.7.2(3), J.4(2), K(42)
    A_Copy_Sign_Attribute,         -- A.5.3(51), K(44)
    A_Count_Attribute,             -- 9.9(5), K(48)
    A_Definite_Attribute,          -- 12.5.1(23), K(50)
    A_Delta_Attribute,             -- 3.5.10(3), K(52)
    A_Denorm_Attribute,            -- A.5.3(9), K(54)
    A_Digits_Attribute,            -- 3.5.8(2), 3.5.10(7), K(56), K(58)
    An_Exponent_Attribute,         -- A.5.3(18), K(60)
    An_External_Tag_Attribute,     -- 13.3(75), K(64)
    A_First_Attribute,             -- 3.5(12), 3.6.2(3), K(68), K(70)
    A_First_Bit_Attribute,         -- 13.5.2(3), K(72)
    A_Floor_Attribute,             -- A.5.3(30), K(74)
    A_Fore_Attribute,              -- 3.5.10(4), K(78)
    A_Fraction_Attribute,          -- A.5.3(21), K(80)
    An_Identity_Attribute,         -- 11.4.1(9), C.7.1(12), K(84), K(86)
    An_Image_Attribute,            -- 3.5(35), K(88)
    An_Input_Attribute,            -- 13.13.2(22), 13.13.2(32), K(92), K(96)
    A_Last_Attribute,              -- 3.5(13), 3.6.2(5), K(102), K(104)
    A_Last_Bit_Attribute,          -- 13.5.2(4), K(106)
    A_Leading_Part_Attribute,      -- A.5.3(54), K(108)
    A_Length_Attribute,            -- 3.6.2(9), K(117)
    A_Machine_Attribute,           -- A.5.3(60), K(119)
    A_Machine_Emax_Attribute,      -- A.5.3(8), K(123)
    A_Machine_Emin_Attribute,      -- A.5.3(7), K(125)
    A_Machine_Mantissa_Attribute,  -- A.5.3(6), K(127)
    A_Machine_Overflows_Attribute, -- A.5.3(12), A.5.4(4), K(129), K(131)
    A_Machine_Radix_Attribute,     -- A.5.3(2), A.5.4(2), K(133), K(135)
    A_Machine_Rounds_Attribute,    -- A.5.3(11), A.5.4(3), K(137), K(139)
    A_Max_Attribute,               -- 3.5(19), K(141)
    A_Max_Size_In_Storage_Elements_Attribute,--   13.11.1(3), K(145)
    A_Min_Attribute,               -- 3.5(16), K(147)
    A_Model_Attribute,             -- A.5.3(68), G.2.2(7), K(151)
    A_Model_Emin_Attribute,        -- A.5.3(65), G.2.2(4), K(155)
    A_Model_Epsilon_Attribute,     -- A.5.3(66), K(157)
    A_Model_Mantissa_Attribute,    -- A.5.3(64), G.2.2(3), K(159)
    A_Model_Small_Attribute,       -- A.5.3(67), K(161)
    A_Modulus_Attribute,           -- 3.5.4(17), K(163)
    An_Output_Attribute,           -- 13.13.2(19), 13.13.2(29), K(165), K(169)
    A_Partition_ID_Attribute,      -- E.1(9), K(173)
    A_Pos_Attribute,               -- 3.5.5(2), K(175)
    A_Position_Attribute,          -- 13.5.2(2), K(179)
    A_Pred_Attribute,              -- 3.5(25), K(181)
    A_Range_Attribute,             -- 3.5(14), 3.6.2(7), K(187), K(189)
    A_Read_Attribute,              -- 13.13.2(6), 13.13.2(14), K(191), K(195)
    A_Remainder_Attribute,         -- A.5.3(45), K(199)
    A_Round_Attribute,             -- 3.5.10(12), K(203)
    A_Rounding_Attribute,          -- A.5.3(36), K(207)
```

```
    A_Safe_First_Attribute,        -- A.5.3(71), G.2.2(5), K(211)
    A_Safe_Last_Attribute,         -- A.5.3(72), G.2.2(6), K(213)
    A_Scale_Attribute,             -- 3.5.10(11), K(215)
    A_Scaling_Attribute,           -- A.5.3(27), K(217)
    A_Signed_Zeros_Attribute,      -- A.5.3(13), K(221)
    A_Size_Attribute,              -- 13.3(40), 13.3(45), K(223), K(228)
    A_Small_Attribute,             -- 3.5.10(2), K(230)
    A_Storage_Pool_Attribute,      -- 13.11(13), K(232)
    A_Storage_Size_Attribute,      -- 13.3(60), 13.11(14), J.9(2), K(234),
                                      -- K(236)
    A_Succ_Attribute,              -- 3.5(22), K(238)
    A_Tag_Attribute,               -- 3.9(16), 3.9(18), K(242), K(244)
    A_Terminated_Attribute,        -- 9.9(3), K(246)
    A_Truncation_Attribute,        -- A.5.3(42), K(248)
    An_Unbiased_Rounding_Attribute,-- A.5.3(39), K(252)
    An_Unchecked_Access_Attribute, -- 13.10(3), H.4(18), K(256)
    A_Val_Attribute,               -- 3.5.5(5), K(258)
    A_Valid_Attribute,             -- 13.9.2(3), H(6), K(262)
    A_Value_Attribute,             -- 3.5(52), K(264)
    A_Version_Attribute,           -- E.3(3), K(268)
    A_Wide_Image_Attribute,        -- 3.5(28), K(270)
    A_Wide_Value_Attribute,        -- 3.5(40), K(274)
    A_Wide_Width_Attribute,        -- 3.5(38), K(278)
    A_Width_Attribute,             -- 3.5(39), K(280)
    A_Write_Attribute,             -- 13.13.2(3), 13.13.2(11), K(282), K(286)

    An_Implementation_Defined_Attribute,  -- Reference Manual, Annex M
    An_Unknown_Attribute);         -- Unknown to ASIS

-------------------------------------------------------------------------------------
```

-- 3.9.20 type Statement_Kinds

```
-------------------------------------------------------------------------------------
-- Statement_Kinds - classifications of Ada statements
--  Literals                              -- Reference Manual
-------------------------------------------------------------------------------------

  type Statement_Kinds is (

    Not_A_Statement,                      -- An unexpected element

    A_Null_Statement,                     -- 5.1
    An_Assignment_Statement,              -- 5.2
    An_If_Statement,                      -- 5.3
    A_Case_Statement,                     -- 5.4

    A_Loop_Statement,                     -- 5.5
    A_While_Loop_Statement,               -- 5.5
    A_For_Loop_Statement,                 -- 5.5
    A_Block_Statement,                    -- 5.6
    An_Exit_Statement,                    -- 5.7
    A_Goto_Statement,                     -- 5.8

    A_Procedure_Call_Statement,           -- 6.4
    A_Return_Statement,                   -- 6.5

    An_Accept_Statement,                  -- 9.5.2
    An_Entry_Call_Statement,              -- 9.5.3

    A_Requeue_Statement,                  -- 9.5.4
    A_Requeue_Statement_With_Abort,       -- 9.5.4

    A_Delay_Until_Statement,              -- 9.6
    A_Delay_Relative_Statement,           -- 9.6

    A_Terminate_Alternative_Statement,    -- 9.7.1
    A_Selective_Accept_Statement,         -- 9.7.1
    A_Timed_Entry_Call_Statement,         -- 9.7.2
    A_Conditional_Entry_Call_Statement,   -- 9.7.3
    An_Asynchronous_Select_Statement,     -- 9.7.4

    An_Abort_Statement,                   -- 9.8
    A_Raise_Statement,                    -- 11.3
    A_Code_Statement);                    -- 13.8
```

```
-------------------------------------------------------------------------------
```

-- 3.9.21 type Path_Kinds

```
-------------------------------------------------------------------------------
--
-- A_Path elements represent execution path alternatives presented by the
-- if_statement, case_statement, and the four forms of select_statement.
-- Each statement path alternative encloses component elements that
-- represent a sequence_of_statements.  Some forms of A_Path elements also
-- have as a component elements that represent a condition, an optional
-- guard, or a discrete_choice_list.
--
-- ASIS treats the select_alternative, entry_call_alternative, and
-- triggering_alternative, as the syntactic equivalent of a
-- sequence_of_statements.  Specifically, the terminate_alternative (terminate;)
-- is treated as the syntactical equivalent of a single statement and are
-- represented as Statement_Kinds'A_Terminate_Alternative_Statement.
-- This allows queries to directly provide the sequence_of_statements enclosed
-- by A_Path elements, avoiding the extra step of returning an element
-- representing such an alternative.
--
-- For example,
--
--     select   -- A_Select_Path enclosing a sequence of two statements
--
--       accept Next_Work_Item(WI : in Work_Item) do
--         Current_Work_Item := WI;
--       end;
--       Process_Work_Item(Current_Work_Item);
--
--     or       -- An_Or_Path enclosing a guard and a sequence of two statements
--
--       when Done_Early =>
--         accept Shut_Down;
--         exit;
--
--     or       -- An_Or_Path enclosing a sequence with only a single statement
--
--       terminate;
--
--     end select;
--
-------------------------------------------------------------------------------
-- Path_Kinds
--  Literals                     -- Reference Manual
-------------------------------------------------------------------------------

  type Path_Kinds is (

    Not_A_Path,                  -- An unexpected element

    An_If_Path,                  -- 5.3:
                                       -- if condition then
                                       --   sequence_of_statements

    An_Elsif_Path,               -- 5.3:
                                       -- elsif condition then
                                       --   sequence_of_statements

    An_Else_Path,                -- 5.3, 9.7.1, 9.7.3:
                                       -- else sequence_of_statements

    A_Case_Path,                 -- 5.4:
                                       -- when discrete_choice_list =>
                                       --   sequence_of_statements

    A_Select_Path,               -- 9.7.1:
                                       -- select [guard] select_alternative
                                       -- 9.7.2, 9.7.3:
                                       -- select entry_call_alternative
                                       -- 9.7.4:
                                       -- select triggering_alternative
```

```
    An_Or_Path,                    -- 9.7.1:
                                         -- or [guard] select_alternative
                                   -- 9.7.2:
                                         -- or delay_alternative

    A_Then_Abort_Path);            -- 9.7.4
                                         -- then abort sequence_of_statements

---------------------------------------------------------------------------------------
```

-- 3.9.22 type Clause_Kinds

```
---------------------------------------------------------------------------------------
-- Clause_Kinds
--  Literals                       -- Reference Manual    -> Subordinate Kinds
---------------------------------------------------------------------------------------

  type Clause_Kinds is (

    Not_A_Clause,                  -- An unexpected element

    A_Use_Package_Clause,          -- 8.4
    A_Use_Type_Clause,             -- 8.4

    A_With_Clause,                 -- 10.1.2

    A_Representation_Clause,       -- 13.1     -> Representation_Clause_Kinds
    A_Component_Clause);           -- 13.5.1

---------------------------------------------------------------------------------------
```

-- 3.9.23 type Representation_Clause_Kinds

```
---------------------------------------------------------------------------------------
-- Representation_Clause_Kinds - varieties of representation clauses
--  Literals                                   -- Reference Manual
---------------------------------------------------------------------------------------

  type Representation_Clause_Kinds is (

    Not_A_Representation_Clause,               -- An unexpected element

    An_Attribute_Definition_Clause,            -- 13.3
    An_Enumeration_Representation_Clause,      -- 13.4
    A_Record_Representation_Clause,            -- 13.5.1
    An_At_Clause);                             -- J.7

---------------------------------------------------------------------------------------
```

-- 3.10 type Compilation_Unit

```
---------------------------------------------------------------------------------------
-- The Ada Compilation Unit abstraction:
--
-- The text of a program is submitted to the compiler in one or more
-- compilations.  Each compilation is a succession of compilation units.
--
-- Compilation units are composed of three distinct parts:
--
-- a) A context clause.
--
-- b) The declaration of a library_item or unit.
--
-- c) Pragmas that apply to the compilation, of which the unit is a part.
--
-- The context clause contains zero or more with clauses, use clauses,
-- pragma elaborates, and possibly other pragmas.
--
-- ASIS treats Pragmas that appear immediately after the context clause and before
-- before the subsequent declaration part as belonging to the context clause part.
--
-- The declaration associated with a compilation unit is one of: a
-- package, a procedure, a function, a generic, or a subunit for normal units.
-- The associated declaration is a Nil_Element for An_Unknown_Unit and
-- Nonexistent units.
--
-- The abstract type Compilation_Unit is a handle for compilation units as a
-- whole.  An object of the type Compilation_Unit deals with the external view
```

```
-- of compilation units such as their relationships with other units or their
-- compilation attributes.
--
-- Compilation_Unit shall be an undiscriminated private type.
----------------------------------------------------------------------------------------

    type Compilation_Unit is private;
    Nil_Compilation_Unit : constant Compilation_Unit;

    function "=" (Left  : in Compilation_Unit;
                  Right : in Compilation_Unit)
                  Return Boolean is abstract;


----------------------------------------------------------------------------------------
```

-- 3.11 type Compilation_Unit_List

```
----------------------------------------------------------------------------------------

    type Compilation_Unit_List is
           array (List_Index range <>) of Compilation_Unit;

    Nil_Compilation_Unit_List : constant Compilation_Unit_List;

----------------------------------------------------------------------------------------
```

-- 3.12 Unit Kinds

```
----------------------------------------------------------------------------------------
-- Unit Kinds are enumeration types describing the various kinds of units.
-- These element kinds are only used by package Asis.Compilation_Units.
----------------------------------------------------------------------------------------
```

-- 3.12.1 type Unit_Kinds

```
----------------------------------------------------------------------------------------
-- Unit_Kinds - the varieties of compilation units of compilations,
-- including compilations having no compilation units but consisting of
-- configuration pragmas or comments.
----------------------------------------------------------------------------------------

  type Unit_Kinds is (

    Not_A_Unit,                  -- A Nil_Compilation_Unit

    A_Procedure,
    A_Function,
    A_Package,

    A_Generic_Procedure,
    A_Generic_Function,
    A_Generic_Package,

    A_Procedure_Instance,
    A_Function_Instance,
    A_Package_Instance,

    A_Procedure_Renaming,
    A_Function_Renaming,
    A_Package_Renaming,

    A_Generic_Procedure_Renaming,
    A_Generic_Function_Renaming,
    A_Generic_Package_Renaming,

    A_Procedure_Body,    -- A unit interpreted only as the completion
                         -- of a procedure, or a unit interpreted as
                         -- both the declaration and body of a library
                         -- procedure. Reference Manual 10.1.4(4)

    A_Function_Body,     -- A unit interpreted only as the completion
                         -- of a function, or a unit interpreted as
                         -- both the declaration and body of a library
                         -- function. Reference Manual 10.1.4(4)

    A_Package_Body,
```

```
    A_Procedure_Body_Subunit,
    A_Function_Body_Subunit,
    A_Package_Body_Subunit,
    A_Task_Body_Subunit,
    A_Protected_Body_Subunit,

    A_Nonexistent_Declaration,   -- A unit that does not exist but is:
                                 -- 1) mentioned in a with clause of
                                 --    another unit or,
                                 -- 2) a required corresponding
                                 --    library_unit_declaration

    A_Nonexistent_Body,          -- A unit that does not exist but is:
                                 -- 1) known to be a corresponding
                                 --    subunit or,
                                 -- 2) a required corresponding
                                 --    library_unit_body

    A_Configuration_Compilation, -- Corresponds to the whole content of a
                                 -- compilation with no compilation_unit,
                                 -- but possibly containing comments,
                                 -- configuration pragmas, or both.
                                 -- A Context is not limited to the number of
                                 -- units of A_Configuration_Compilation kind.
                                 -- A unit of A_Configuration_Compilation
                                 -- does not have a name. This unit
                                 -- represents configuration pragmas that
                                 -- are “in effect”. The only interface that
                                 -- returns this unit kind is
                                 -- Enclosing_Compilation_Unit when given
                                 -- A_Pragma element obtained from
-- Configuration_Pragmas.

    An_Unknown_Unit);            -- An indeterminable or proprietary unit


  subtype A_Subprogram_Declaration is Unit_Kinds range
            A_Procedure ..
            A_Function;

  subtype A_Subprogram_Renaming is Unit_Kinds range
            A_Procedure_Renaming ..
            A_Function_Renaming;

  subtype A_Generic_Unit_Declaration is Unit_Kinds range
            A_Generic_Procedure ..
            A_Generic_Package;

  subtype A_Generic_Unit_Instance is Unit_Kinds range
            A_Procedure_Instance ..
            A_Package_Instance;

  subtype A_Subprogram_Body is Unit_Kinds range
            A_Procedure_Body ..
            A_Function_Body;

  subtype A_Library_Unit_Body is Unit_Kinds range
            A_Procedure_Body ..
            A_Package_Body;

  subtype A_Generic_Renaming is Unit_Kinds range
            A_Generic_Procedure_Renaming ..
            A_Generic_Package_Renaming;

  subtype A_Renaming is Unit_Kinds range
            A_Procedure_Renaming ..
            A_Generic_Package_Renaming;

  subtype A_Subunit is Unit_Kinds range
            A_Procedure_Body_Subunit ..
            A_Protected_Body_Subunit;
```

```
-------------------------------------------------------------------------------------
```
-- 3.12.2 type Unit_Classes
```
-------------------------------------------------------------------------------------
-- Unit_Classes - classification of public, private, body, and subunit.
-------------------------------------------------------------------------------------

  type Unit_Classes is (  -- Reference Manual 10.1.1(12), 10.1.3

    Not_A_Class,              -- A nil, nonexistent, unknown,
                              -- or configuration compilation unit class.

    A_Public_Declaration,     -- library_unit_declaration or
                              -- library_unit_renaming_declaration.

    A_Public_Body,            -- library_unit_body interpreted only as a
                              -- completion.  Its declaration is public.

    A_Public_Declaration_And_Body,
                              -- subprogram_body interpreted as both a
                              -- declaration and body of a library
                              -- subprogram - Reference Manual 10.1.4(4).

    A_Private_Declaration,    -- private library_unit_declaration or
                              -- private library_unit_renaming_declaration.

    A_Private_Body,           -- library_unit_body interpreted only as a
                              -- completion.  Its declaration is private.

    A_Separate_Body);         -- separate (parent_unit_name) proper_body.

-------------------------------------------------------------------------------------
```
-- 3.12.3 type Unit_Origins
```
-------------------------------------------------------------------------------------
-- Unit_Origins - classification of possible unit origination
-------------------------------------------------------------------------------------

  type Unit_Origins is (

    Not_An_Origin,        -- A nil or nonexistent unit origin
                          -- An_Unknown_Unit can be any origin

    A_Predefined_Unit,    -- Ada predefined language environment units
                          -- listed in Annex A(2).  These include
                          -- Standard and the three root library
                          -- units: Ada, Interfaces, and System,
                          -- and their descendants.  i.e., Ada.Text_Io,
                          -- Ada.Calendar, Interfaces.C, etc.

    An_Implementation_Unit,
                          -- Implementation specific library units,
                          -- e.g., runtime support packages, utility
                          -- libraries, etc. It is not required
                          -- that any implementation supplied units
                          -- have this origin.  This is a suggestion.
                          -- Implementations might provide, for
                          -- example, precompiled versions of public
                          -- domain software that could have
                          -- An_Application_Unit origin.

    An_Application_Unit); -- Neither A_Predefined_Unit or
                          -- An_Implementation_Unit

-------------------------------------------------------------------------------------
```
-- 3.12.4 type Relation_Kinds
```
-------------------------------------------------------------------------------------
-- Relation_Kinds - classification of unit relationships

  type Relation_Kinds is (

    Ancestors,

    Descendants,
```

```
---------------------------------------------------------------------------
-- Definition:  ANCESTORS of a unit; DESCENDANTS of a unit
--
-- Ancestors of a library unit are itself, its parent, its parent's
-- parent, and so on.  (Standard is an ancestor of every library unit).
--
-- The Descendants relation is the inverse of the ancestor relation.
-- Reference Manual 10.1.1(11).
---------------------------------------------------------------------------

    Supporters,

---------------------------------------------------------------------------
-- Definition:  SUPPORTERS of a unit
--
-- Supporters of a compilation unit are units on which it semantically
-- depends.  Reference Manual 10.1.1(26).
--
-- The Supporters relation is transitive; units that are supporters of library
-- units mentioned in a with clause of a compilation unit are also supporters
-- of that compilation unit.
--
-- A parent declaration is a Supporter of its descendant units.
--
-- Each library unit mentioned in the with clauses of a compilation unit
-- is a Supporter of that compilation unit and (recursively) any
-- completion, child units, or subunits that are included in the declarative
-- region of that compilation unit.  Reference Manual 8.1(7-10).
--
-- A library_unit_body has as a Supporter, its corresponding
-- library_unit_declaration, if any.
--
-- The parent body of a subunit is a Supporter of the subunit.
--
---------------------------------------------------------------------------

    Dependents,

---------------------------------------------------------------------------
-- Definition:  DEPENDENTS of a unit
--
-- Dependents of a compilation unit are all the compilation units that
-- depend semantically on it.
--
-- The Dependents relation is transitive; Dependents of a unit include the
-- unit's Dependents, each dependent unit's Dependents, and so on.  A unit
-- that is a dependent of a compilation unit also is a dependent
-- of the compilation unit's Supporters.
--
-- Child units are Dependents of their ancestor units.
--
-- A compilation unit that mentions other library units in its with
-- clauses is one of the Dependents of those library units.
--
-- A library_unit_body is a Dependent of its corresponding
-- library_unit_declaration, if any.
--
-- A subunit is a Dependent of its parent body.
--
-- A compilation unit that contains an attribute_reference of a type defined
-- in another compilation unit is a Dependent of the other unit.
--
-- For example:
--
--      If A withs B and B withs C
--      then A directly depends on A, B directly depends on C,
--           A indirectly depends on C, and
--           both A and B are dependents of C.
--
-- Dependencies between compilation units may also be introduced by
-- inline inclusions (Reference Manual 10.1.4(7)) and for certain other compiler
-- optimizations.  These relations are intended to reflect all of these
-- considerations.
```

```
--
---------------------------------------------------------------------------

    Family,

---------------------------------------------------------------------------
-- Definition:  FAMILY of a unit
--
-- The family of a given unit is defined as the set of compilation
-- units that comprise the given unit's declaration, body, descendants,
-- and subunits (and subunits of subunits and descendants, etc.).
---------------------------------------------------------------------------

    Needed_Units);

---------------------------------------------------------------------------
-- Definition:  NEEDED UNITS of a unit; CLOSURE of a unit
--
-- The needed units of a given unit is defined as the set of all
-- the Ada units ultimately needed by that unit to form a partition.
-- Reference Manual 10.2(2-7).
--
-- The term closure is commonly used with similar meaning.
--
-- For example:
--  Assume the body of C has a subunit C.S and the declaration of C has
--  child units C.Y and C.Z.
--
--      If A withs B, B withs C, B withs C.Y, and C does not with a library
--      unit.  Then the needed units of A are:
--        library unit declaration C
--        child library unit declaration C.Y
--        child library unit body C.Y, if any
--        library unit body C
--        subunit C.S
--        library unit declaration B
--        library unit body B, if any
--        library unit declaration A
--        library unit body A, if any
--
--      Child unit C.Z is only part of the Needed_Units if it is needed.
--
------------------------------------------------------------------------------------
```

-- 3.13 type Traverse_Control

```
------------------------------------------------------------------------------------
-- Traverse_Control - controls for the traversal generic provided in package
-- Asis.Iterator. It is defined in package Asis to facilitate automatic translation
-- to IDL (See Annex C for details).
------------------------------------------------------------------------------------

    type Traverse_Control is (

      Continue,               -- Continues the normal depth-first traversal.

      Abandon_Children,       -- Prevents traversal of the current element's
                              -- children.

      Abandon_Siblings,       -- Prevents traversal of the current element's
                              -- children and remaining siblings.

      Terminate_Immediately); -- Does exactly that.

------------------------------------------------------------------------------------
```

-- 3.14 type Program_Text

```
------------------------------------------------------------------------------------

    subtype Program_Text is Wide_String;

------------------------------------------------------------------------------------
```

```
private

    type Context is (Implementation_Defined);
    Nil_Context : constant Context := Implementation_Defined;

    type Element is (Implementation_Defined);
    Nil_Element : constant Element := Implementation_Defined;
    Nil_Element_List : constant Element_List (1 .. 0) :=
        (1 .. 0 => Nil_Element);

    type Compilation_Unit is (Implementation_Defined);
    Nil_Compilation_Unit : constant Compilation_Unit :=
        Implementation_Defined;
    Nil_Compilation_Unit_List : constant Compilation_Unit_List (1 .. 0) :=
        (1 .. 0 => Nil_Compilation_Unit);

end Asis;
```

```
---------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------
```

-- 4 package Asis.Errors

```
---------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------
package Asis.Errors is
---------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------
--
-- ASIS reports all operational errors by raising an exception.  Whenever an
-- ASIS implementation raises one of the exceptions declared in package
-- Asis.Exceptions, it will previously have set the values returned by the
-- Status and Diagnosis queries to indicate the cause of the error.  The
-- possible values for Status are indicated in the definition of Error_Kinds
-- below, with suggestions for the associated contents of the Diagnosis
-- string as a comment.
--
-- The Diagnosis and Status queries are provided in the Asis.Implementation
-- package to supply more information about the reasons for raising any
-- exception.
--
-- ASIS applications are encouraged to follow this same convention whenever
-- they explicitly raise any ASIS exception--always record a Status and
-- Diagnosis prior to raising the exception.
---------------------------------------------------------------------------------------
```

-- 4.1 type Error_Kinds

```
---------------------------------------------------------------------------------------
-- This enumeration type describes the various kinds of errors.
--

  type Error_Kinds is (

        Not_An_Error,                -- No error is presently recorded
        Value_Error,                 -- Routine argument value invalid
        Initialization_Error,        -- ASIS is uninitialized
        Environment_Error,           -- ASIS could not initialize
        Parameter_Error,             -- Bad Parameter given to Initialize
        Capacity_Error,              -- Implementation overloaded
        Name_Error,                  -- Context/unit not found
        Use_Error,                   -- Context/unit not use/open-able
        Data_Error,                  -- Context/unit bad/invalid/corrupt
        Text_Error,                  -- The program text cannot be located
        Storage_Error,               -- Storage_Error suppressed
        Obsolete_Reference_Error,    -- Argument or result is invalid due to
                                     -- and inconsistent compilation unit
        Unhandled_Exception_Error,   -- Unexpected exception suppressed
        Not_Implemented_Error,       -- Functionality not implemented
        Internal_Error);             -- Implementation internal failure

---------------------------------------------------------------------------------------

end Asis.Errors;
```

```
---------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------
```

-- 5 package Asis.Exceptions

```
---------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------
package Asis.Exceptions is
---------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------
-- ASIS exceptions are:

    ASIS_Inappropriate_Context : exception;

---------------------------------------------------------------------------------------
-- Raised when ASIS is passed a Context value that is not appropriate for the
-- operation.  This exception will typically indicate that a user error
-- has occurred within the application.
---------------------------------------------------------------------------------------

    ASIS_Inappropriate_Container : exception;

---------------------------------------------------------------------------------------
-- Raised when ASIS is passed a Container value that is not appropriate for
-- the operation.  This exception will typically indicate that a user error
-- has occurred within the application.
---------------------------------------------------------------------------------------

    ASIS_Inappropriate_Compilation_Unit : exception;

---------------------------------------------------------------------------------------
-- Raised when ASIS is passed a Compilation_Unit value that is not
-- appropriate.  This exception will typically indicate that a user
-- error has occurred within the application.
---------------------------------------------------------------------------------------

    ASIS_Inappropriate_Element : exception;

---------------------------------------------------------------------------------------
-- Raised when ASIS is given an Element value that is not appropriate.  This
-- exception will typically indicate that a user error has occurred within
-- the application.
---------------------------------------------------------------------------------------

    ASIS_Inappropriate_Line : exception;

---------------------------------------------------------------------------------------
-- Raised when ASIS is given a Line value that is not appropriate.
---------------------------------------------------------------------------------------

    ASIS_Inappropriate_Line_Number : exception;

---------------------------------------------------------------------------------------
-- Raised when ASIS is given a Line_Number value that is not appropriate.
-- This exception will typically indicate that a user error has occurred
-- within the application.
---------------------------------------------------------------------------------------

    ASIS_Failed : exception;

---------------------------------------------------------------------------------------
-- This is a catch-all exception that may be raised for different reasons
-- in different ASIS implementations.  All ASIS routines may raise ASIS_Failed
-- whenever they cannot normally complete their operation.  This exception
-- will typically indicate a failure of the underlying ASIS implementation.
---------------------------------------------------------------------------------------

end Asis.Exceptions;
```

```
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
```

-- 6 package Asis.Implementation

```
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
with Asis.Errors;
package Asis.Implementation is
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-- Asis.Implementation provides queries to initialize, finalize, and query the
-- error status of the ASIS Implementation.
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------

-----------------------------------------------------------------------------------------
```

-- 6.1 function ASIS_Version

```
-----------------------------------------------------------------------------------------

    function ASIS_Version return Wide_String;

-----------------------------------------------------------------------------------------
```

-- 6.2 function ASIS_Implementor

```
-----------------------------------------------------------------------------------------

    function ASIS_Implementor return Wide_String;

-----------------------------------------------------------------------------------------
```

-- 6.3 function ASIS_Implementor_Version

```
-----------------------------------------------------------------------------------------

    function ASIS_Implementor_Version return Wide_String;

-----------------------------------------------------------------------------------------
```

-- 6.4 function ASIS_Implementor_Information

```
-----------------------------------------------------------------------------------------

    function ASIS_Implementor_Information return Wide_String;

-----------------------------------------------------------------------------------------
-- Returns values which identify:
--
--  ASIS_Version                 - the version of the ASIS interface, e.g., "2.1"
--  ASIS_Implementor             - the name of the implementor, e.g., "Ada Inc."
--  ASIS_Implementor_Version     - the implementation’s version, e.g., "5.2a"
--  ASIS_Implementor_Information - implementation information, e.g., "Copyright ..."
--
-----------------------------------------------------------------------------------------
```

-- 6.5 function Is_Initialized

```
-----------------------------------------------------------------------------------------

    function Is_Initialized return Boolean;

-----------------------------------------------------------------------------------------
-- Returns True if ASIS is currently initialized.
--
-----------------------------------------------------------------------------------------
```

-- 6.6 procedure Initialize

```
-----------------------------------------------------------------------------------------

    procedure Initialize (Parameters : in Wide_String := "");

-----------------------------------------------------------------------------------------
-- Parameters  - Specifies implementation specific parameters.
--
-- Performs any necessary initialization activities.  This shall be invoked
-- at least once before any other ASIS services are used.  Parameter values
-- are implementation dependent.  The call is ignored if ASIS is already
-- initialized. All ASIS queries and services are ready for use once this
-- call completes.
```

```
--
-- Raises ASIS_Failed if ASIS failed to initialize or if the Parameters
-- argument is invalid.  Status is Environment_Error or Parameter_Error.
--
--|AN Application Note:
--|AN
--|AN The ASIS implementation may be Initialized and Finalized any number of
--|AN times during the operation of an ASIS program.   However, all existing
--|AN Context, Compilation_Unit and Element values become invalid when
--|AN ASIS Is_Finalized.  Subsequent calls to ASIS queries or services using
--|AN such invalid Compilation_Unit or Element values will cause
--|AN ASIS_Inappropriate_Context to be raised.
--
-----------------------------------------------------------------------------------
```

-- 6.7 function Is_Finalized

```
-----------------------------------------------------------------------------------

    function Is_Finalized return Boolean;

-----------------------------------------------------------------------------------
-- Returns True if ASIS is currently finalized or if ASIS has never been
-- initialized.
--
-----------------------------------------------------------------------------------
```

-- 6.8 procedure Finalize

```
-----------------------------------------------------------------------------------

    procedure Finalize (Parameters : in Wide_String := "");

-----------------------------------------------------------------------------------
-- Parameters  - Specifies any implementation required parameter values.
--
-- Performs any necessary ASIS termination activities.  This should be invoked
-- once following the last use of other ASIS queries.  Parameter values are
-- implementation dependent.  The call is ignored if ASIS is already finalized.
-- Subsequent calls to ASIS Environment, Compilation_Unit, and Element queries,
-- are erroneous while the environment Is_Finalized.
--
-- Raises ASIS_Failed if the ASIS implementation failed to finalize.  Status
-- is likely to be Internal_Error and will not be Not_An_Error.
--
-----------------------------------------------------------------------------------
-- Whenever an error condition is detected, and any ASIS exception is raised,
-- an Asis.Errors.Error_Kinds value and a Diagnosis string is stored.  These
-- values can be retrieved by the Status and Diagnosis functions.  The
-- Diagnosis function will retrieve the diagnostic message describing the error.
--
-- Error information always refers to the most recently recorded error.
--
-- Note that Diagnosis values are implementation dependent and may vary
-- greatly among ASIS implementations.
--
-----------------------------------------------------------------------------------
```

-- 6.9 function Status

```
-----------------------------------------------------------------------------------

    function Status return Asis.Errors.Error_Kinds;

-----------------------------------------------------------------------------------
-- Returns the Error_Kinds value for the most recent error.
--
-----------------------------------------------------------------------------------
```

-- 6.10 function Diagnosis

```
-----------------------------------------------------------------------------------

    function Diagnosis return Wide_String;

-----------------------------------------------------------------------------------
-- Returns a string value describing the most recent error.
--
-- Will typically return a null string if Status = Not_An_Error.
```

```
--
---------------------------------------------------------------------------------------
```

-- 6.11 procedure Set_Status

```
---------------------------------------------------------------------------------------

    procedure Set_Status
        (Status    : in Asis.Errors.Error_Kinds := Asis.Errors.Not_An_Error;
         Diagnosis : in Wide_String        := "");

---------------------------------------------------------------------------------------
-- Status      - Specifies the new status to be recorded
-- Diagnosis   - Specifies the new diagnosis to be recorded
--
-- Sets (clears, if the defaults are used) the Status and Diagnosis
-- information.  Future calls to Status will return this Status (Not_An_Error)
-- and this Diagnosis (a null string).
--
-- Raises ASIS_Failed, with a Status of Internal_Error and a Diagnosis of
-- a null string, if the Status parameter is Not_An_Error and the Diagnosis
-- parameter is not a null string.
--
---------------------------------------------------------------------------------------

end Asis.Implementation;
```

```
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
```

-- 7 package Asis.Implementation.Permissions

```
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
package Asis.Implementation.Permissions is
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
```

-- 7.1 function Is_Formal_Parameter_Named_Notation_Supported

```
------------------------------------------------------------------------------------------

    function Is_Formal_Parameter_Named_Notation_Supported return Boolean;

------------------------------------------------------------------------------------------
-- Returns True if it is possible to detect usage of named notation.
--
-- Returns False if this implementation will always change parameter lists
-- using named notation to positional lists in function, subprogram, and
-- entry calls.  In that case, the Formal_Parameter query will always return
-- a Nil_Element unless the parameter list is obtained with Normalized = True.
--
-- This function affects association lists for aggregates, instantiations,
-- discriminant lists, entry calls, and subprogram calls.
--
------------------------------------------------------------------------------------------
```

-- 7.2 function Default_In_Mode_Supported

```
------------------------------------------------------------------------------------------

    function Default_In_Mode_Supported return Boolean;

------------------------------------------------------------------------------------------
-- Returns True if the A_Default_In_Mode kind is supported by this
-- implementation.
--
------------------------------------------------------------------------------------------
```

-- 7.3 function Generic_Actual_Part_Normalized

```
------------------------------------------------------------------------------------------

    function Generic_Actual_Part_Normalized return Boolean;

------------------------------------------------------------------------------------------
-- Returns True if the query Generic_Actual_Part will always return artificial
-- Is_Normalized associations using the defining_identifier instead of the
-- generic_formal_parameter_selector_name, and using default_expression or
-- default_name.
--
-- If Generic_Actual_Part_Normalized then the query Generic_Actual_Part will
-- always behave as if called with Normalized => True.
--
------------------------------------------------------------------------------------------
```

-- 7.4 function Record_Component_Associations_Normalized

```
------------------------------------------------------------------------------------------

    function Record_Component_Associations_Normalized return Boolean;

------------------------------------------------------------------------------------------
-- Returns True if the query Record_Component_Associations will always return
-- artificial Is_Normalized associations using the defining_identifier instead of
-- the component_selector_name.
--
-- If Record_Component_Associations_Normalized then the query
-- Record_Component_Associations will always behave as if called with
-- Normalized => True.
--
```

```
-------------------------------------------------------------------------------
```

-- 7.5 function Is_Prefix_Call_Supported

```
-------------------------------------------------------------------------------

    function Is_Prefix_Call_Supported return Boolean;

-------------------------------------------------------------------------------
-- Returns True if the ASIS implementation has the ability to determine
-- whether calls are in prefix form.
--
-------------------------------------------------------------------------------
```

-- 7.6 function Function_Call_Parameters_Normalized

```
-------------------------------------------------------------------------------

    function Function_Call_Parameters_Normalized return Boolean;

-------------------------------------------------------------------------------
-- Returns True if the query Function_Call_Parameters will always return
-- artificial Is_Normalized associations using the defining_identifier instead of
-- the formal_parameter_selector_name, and using the default_expression.
--
-- If Function_Call_Parameters_Normalized then the query
-- Function_Call_Parameters will always behave as if called with
-- Normalized => True.
--
-------------------------------------------------------------------------------
```

-- 7.7 function Call_Statement_Parameters_Normalized

```
-------------------------------------------------------------------------------

    function Call_Statement_Parameters_Normalized return Boolean;

-------------------------------------------------------------------------------
-- Returns True if the query Call_Statement_Parameters will always return
-- artificial Is_Normalized associations using the defining_identifier instead of
-- the formal_parameter_selector_name, and using the default_expression.
--
-- If Call_Statement_Parameters_Normalized then the query
-- Call_Statement_Parameters will always behave as if called with
-- Normalized => True.
--
----------------------------------------------------------------------------
-- It is not possible to obtain either a normalized or
-- unnormalized Discriminant_Association list for an unconstrained record
-- or derived subtype_indication where the discriminant_association is
-- supplied by default; there is no constraint to query, and a Nil_Element
-- is returned from the query Subtype_Constraint.
--
-------------------------------------------------------------------------------
```

-- 7.8 function Discriminant_Associations_Normalized

```
-------------------------------------------------------------------------------

    function Discriminant_Associations_Normalized return Boolean;

-------------------------------------------------------------------------------
-- Returns True if the query Discriminant_Associations will always return
-- artificial Is_Normalized associations using the defining_identifier instead of
-- the discriminant_selector_name.
--
-- If Discriminant_Associations_Normalized then the query
-- Discriminant_Associations will always behave as if called with
-- Normalized => True.
--
```

```
----------------------------------------------------------------------------------------
```
-- 7.9 function Is_Line_Number_Supported
```
----------------------------------------------------------------------------------------

    function Is_Line_Number_Supported
 return Boolean;

----------------------------------------------------------------------------------------
-- Returns True if the implementation can return valid line numbers for
-- Elements.
--
-- An implementation may choose to ignore line number values in which case
-- this function returns False.
--
----------------------------------------------------------------------------------------
```
-- 7.10 function Is_Span_Column_Position_Supported
```
----------------------------------------------------------------------------------------

    function Is_Span_Column_Position_Supported return Boolean;

----------------------------------------------------------------------------------------
-- Returns True if the implementation can return valid character positions for
-- elements.
--
-- An implementation may choose to ignore column character position values
-- within spans in which case this function returns False.  This function will
-- be False if Is_Line_Number_Supported = False.
--
----------------------------------------------------------------------------------------
```
-- 7.11 function Is_Commentary_Supported
```
----------------------------------------------------------------------------------------

    function Is_Commentary_Supported return Boolean;

----------------------------------------------------------------------------------------
-- Returns True if the implementation can return comments.
--
-- An implementation may choose to ignore comments in the text in which case
-- the function Is_Commentary_Supported returns False.
--
----------------------------------------------------------------------------------------
```
-- 7.12 function Attributes_Are_Supported
```
----------------------------------------------------------------------------------------

    function Attributes_Are_Supported return Boolean;

----------------------------------------------------------------------------------------
-- Returns True if an implementation supports compilation unit attributes.
-- Returns False if all attributes will return Has_Attribute() = False.
--
----------------------------------------------------------------------------------------
```
-- 7.13 function Implicit_Components_Supported
```
----------------------------------------------------------------------------------------

    function Implicit_Components_Supported return Boolean;

----------------------------------------------------------------------------------------
-- Returns True if the implementation provides elements representing
-- implicit implementation-defined record components.
--
----------------------------------------------------------------------------------------
```
-- 7.14 function Object_Declarations_Normalized
```
----------------------------------------------------------------------------------------

    function Object_Declarations_Normalized return Boolean;

----------------------------------------------------------------------------------------
-- Returns True if the implementation normalizes multiple object declarations
-- to an equivalent sequence of single declarations.
--
```

```
-----------------------------------------------------------------------------------------
```

-- 7.15 function Predefined_Operations_Supported

```
-----------------------------------------------------------------------------------------

    function Predefined_Operations_Supported return Boolean;

-----------------------------------------------------------------------------------------
-- Returns True if the implementation supports queries of predefined
-- operations.
--
-----------------------------------------------------------------------------------------
```

-- 7.16 function Inherited_Declarations_Supported

```
-----------------------------------------------------------------------------------------

    function Inherited_Declarations_Supported return Boolean;

-----------------------------------------------------------------------------------------
-- Returns True if the implementation supports queries of inherited
-- declarations.
--
-----------------------------------------------------------------------------------------
```

-- 7.17 function Inherited_Subprograms_Supported

```
-----------------------------------------------------------------------------------------

    function Inherited_Subprograms_Supported return Boolean;

-----------------------------------------------------------------------------------------
-- Returns True if the implementation supports queries of inherited
-- subprograms.
--
-----------------------------------------------------------------------------------------
```

-- 7.18 function Generic_Macro_Expansion_Supported

```
-----------------------------------------------------------------------------------------

    function Generic_Macro_Expansion_Supported return Boolean;

-----------------------------------------------------------------------------------------
-- Returns True if the implementation expands generics using macros to
-- supports queries.
-----------------------------------------------------------------------------------------

end Asis.Implementation.Permissions;
```

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

-- 8 package Asis.Ada_Environments

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
package Asis.Ada_Environments is
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- Asis.Ada_Environments encapsulates a set of queries that map physical Ada
-- compilation and program execution environments to logical ASIS environments.
--
-------------------------------------------------------------------------------
```

-- 8.1 function Default_Name

```
-------------------------------------------------------------------------------

    function Default_Name return Wide_String;

-------------------------------------------------------------------------------
-- Returns the default context name.  If there is no default context name, a
-- null string is returned.
--
-------------------------------------------------------------------------------
```

-- 8.2 function Default_Parameters

```
-------------------------------------------------------------------------------

    function Default_Parameters return Wide_String;

-------------------------------------------------------------------------------
-- Returns the default context parameters.  If there are no default context
-- parameters, a null string is returned.
--
-------------------------------------------------------------------------------
```

-- 8.3 procedure Associate

```
-------------------------------------------------------------------------------

    procedure Associate
                 (The_Context : in out Asis.Context;
                  Name        : in     Wide_String;
                  Parameters  : in     Wide_String := Default_Parameters);

-------------------------------------------------------------------------------
-- The_Context - Specifies the Context to associate
-- Name        - Specifies the name for the Context association
-- Parameters  - Specifies parameters to use when opening the Context
--
-- Used to give name and parameter associations to a Context.  The
-- Has_Associations query is used to test whether or not a Context has
-- been given name and parameter associations.  The Name and Parameters
-- queries are used to examine name and parameter associations.
--
-- A Context has at most one set of name/parameter values associated with
-- it at any time.  Name and parameter associations cannot be modified while a
-- Context is open  Previous name and parameters associations for this Context
-- are replaced by this call.
--
-- ASIS implementations are encouraged, but not required, to validate the
-- Parameters string immediately.  It is recognized that some options cannot
-- be completely validated until the Open is performed.  An invalid Parameters
-- value is reported by raising ASIS_Failed with a Status of Parameter_Error.
--
-- Raises ASIS_Inappropriate_Context if The_Context is open.
--
```

```
-------------------------------------------------------------------------------
```
-- 8.4 procedure Open
```
-------------------------------------------------------------------------------

    procedure Open (The_Context : in out Asis.Context);

-------------------------------------------------------------------------------
-- The_Context - Specifies the Context to open
--
-- Opens the ASIS Context using the Context's associated name and parameter
-- values.
--
-- Raises ASIS_Inappropriate_Context if The_Context is already open or if it
-- is uninitialized (does not have associated name and parameter values).
--
-- Raises ASIS_Failed if The_Context could not be opened for any reason.  The
-- most likely Status values are Name_Error, Use_Error, Data_Error, and
-- Parameter_Error.  Other possibilities include Storage_Error and
-- Capacity_Error.
--
-------------------------------------------------------------------------------
```
-- 8.5 procedure Close
```
-------------------------------------------------------------------------------

    procedure Close (The_Context : in out Asis.Context);

-------------------------------------------------------------------------------
-- The_Context - Specifies the Context to close
--
-- Closes the ASIS Context.  Any previous Context name and parameter
-- associations are retained.  This allows the same Context to be re-opened
-- later with the same associations.
--
-- All Compilation_Unit and Element values obtained from The_Context become
-- invalid when it is closed.  Subsequent calls to ASIS services using such
-- invalid Compilation_Unit or Element values are erroneous.  ASIS
-- implementations will attempt to detect such usage and raise ASIS_Failed in
-- response.  Applications should be aware that the ability to detect the use
-- of such "dangling references" is implementation specific and not all
-- implementations are able to raise ASIS_Failed at the appropriate
-- points.  Thus, applications that attempt to utilize invalid values may
-- exhibit unpredictable behavior.
--
-- Raises ASIS_Inappropriate_Context if The_Context is not open.
--
-------------------------------------------------------------------------------
```
-- 8.6 procedure Dissociate
```
-------------------------------------------------------------------------------

    procedure Dissociate (The_Context : in out Asis.Context);

-------------------------------------------------------------------------------
-- The_Context - Specifies the Context whose name and parameter associations
--               are to be cleared
--
-- Severs all previous associations for The_Context.  A Context that does not
-- have associations (is uninitialized) is returned unchanged.  The
-- variable The_Context is returned to its uninitialized state.
--
-- Contexts that have been given Names and Parameters should be Dissociated
-- when they are no longer necessary.  Some amount of program storage can be
-- tied up by the stored Name and Parameter strings.  This space is only
-- freed when a Context is Dissociated or when ASIS is Finalized.
--
-- This operation has no physical affect on any implementor's Ada environment.
--
-- Raises ASIS_Inappropriate_Context if The_Context is open.
--
```

```
---------------------------------------------------------------------------------------
```

-- 8.7 function Is_Equal

```
---------------------------------------------------------------------------------------

    function Is_Equal (Left  : in Asis.Context;
                       Right : in Asis.Context) return Boolean;

---------------------------------------------------------------------------------------
-- Left    - Specifies the first Context
-- Right   - Specifies the second Context
--
-- Returns True if Left and Right designate the same set of associated
-- compilation units.  The Context variables may be open or closed.
--
-- Unless both Contexts are open, this operation is implemented as a pair of
-- simple string comparisons between the Name and Parameter associations for
-- the two Contexts.  If both Contexts are open, this operation acts as a
-- set comparison and returns True if both sets contain the same units (all
-- unit versions are included in the comparison).
--
--|AN Application Note:
--|AN
--|AN With some implementations, Is_Equal may be True before the Contexts
--|AN are opened, but may be False after the Contexts are open.
--|AN One possible cause for this is a sequence of events such as:
--|AN
--|AN a) ASIS program A opens the Left Context for READ,
--|AN
--|AN b) non-ASIS program B opens the Context for UPDATE, and creates a new
--|AN    version of the implementor Context,
--|AN
--|AN c) ASIS program A opens the Right Context for READ, and gets the new version.
--
---------------------------------------------------------------------------------------
```

-- 8.8 function Is_Identical

```
---------------------------------------------------------------------------------------

    function Is_Identical (Left  : in Asis.Context;
                           Right : in Asis.Context) return Boolean;

---------------------------------------------------------------------------------------
-- Left    - Specifies the first Context
-- Right   - Specifies the second Context
--
-- Returns True if Left and Right both designate the value associated with
-- one specific ASIS Context variable.
--
-- Returns False otherwise or if either Context is not open.
--
--|AN Application Note:
--|AN
--|AN No two physically separate open Context variables are ever Is_Identical.
--|AN The value associated with an open ASIS Context variable is also directly
--|AN associated with every Compilation_Unit or Element derived from that
--|AN Context.  It is possible to obtain these Context values by way of the
--|AN Enclosing_Context and the Enclosing_Compilation_Unit queries.  These
--|AN Context values can be tested for identity with each other or with
--|AN specific Context variables.  An open ASIS Context variable and an
--|AN Enclosing_Context value are only Is_Identical if the Compilation_Unit in
--|AN question was derived specifically from that open ASIS Context variable.
--
```

```
-------------------------------------------------------------------------------
```

-- 8.9 function Exists

```
-------------------------------------------------------------------------------

    function Exists (The_Context : in Asis.Context) return Boolean;

-------------------------------------------------------------------------------
-- The_Context - Specifies a Context with associated name and parameter values
--
-- Returns True if The_Context is open or if The_Context designates an Ada
-- environment that can be determined to exist.
--
-- Returns False for any uninitialized The_Context variable.
--
--|IP Implementation Permissions:
--|IP
--|IP No guarantee is made that The_Context is readable or that an Open
--|IP operation on The_Context would succeed.  The associated
--|IP parameter value for The_Context may not be fully validated by this
--|IP simple existence check.  It may contain information that can only be
--|IP verified by an Open.
--
-------------------------------------------------------------------------------
```

-- 8.10 function Is_Open

```
-------------------------------------------------------------------------------

    function Is_Open (The_Context : in Asis.Context) return Boolean;

-------------------------------------------------------------------------------
-- The_Context - Specifies the Context to check
--
-- Returns True if The_Context is currently open.
--
-------------------------------------------------------------------------------
```

-- 8.11 function Has_Associations

```
-------------------------------------------------------------------------------

    function Has_Associations (The_Context : in Asis.Context) return Boolean;

-------------------------------------------------------------------------------
-- The_Context - Specifies the Context to check
--
-- Returns True if name and parameter values have been associated with
-- The_Context.
--
-- Returns False if The_Context is uninitialized.
--
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
```

-- 8.12 function Name

```
-------------------------------------------------------------------------------

    function Name (The_Context : in Asis.Context) return Wide_String;

-------------------------------------------------------------------------------
-- The_Context - Specifies the Context to check
--
-- Returns the Name value associated with The_Context.
--
-- Returns a null string if The_Context is uninitialized.
--
```

```
-------------------------------------------------------------------------------------
```

-- 8.13 function Parameter

```
-------------------------------------------------------------------------------------

    function Parameters (The_Context : in Asis.Context) return Wide_String;

-------------------------------------------------------------------------------------
-- The_Context - Specifies the Context to check
--
-- Returns the Parameters value associated with The_Context.
--
-- Returns a null string if The_Context is uninitialized.
--
-------------------------------------------------------------------------------------
```

-- 8.14 function Debug_Image

```
-------------------------------------------------------------------------------------

     function Debug_Image (The_Context : in Asis.Context) return Wide_String;

-------------------------------------------------------------------------------------
-- The_Context - Specifies the Context to represent
--
-- Returns a string value containing implementation-defined debugging
-- information associated with The_Context.
--
-- The return value uses Asis.Text.Delimiter_Image to separate lines in
-- multi-line results.  The return value is not terminated with
-- Asis.Text.Delimiter_Image.
--
-- Returns a null string if The_Context is uninitialized.
--
-- These values are intended for two purposes.  They are suitable for
-- inclusion in problem reports sent to the ASIS implementor.  They can be
-- presumed to contain information useful when debugging the implementation
-- itself. They are also suitable for use by the ASIS application when printing
-- simple application debugging messages during application development.
-- They are intended to be, to some worthwhile degree, intelligible to the user.
--

end Asis.Ada_Environments;
```

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

-- 9 package Asis.Ada_Environments.Containers

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
package Asis.Ada_Environments.Containers is
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- Asis.Ada_Environments.Containers
--
-- If an Ada implementation supports the notion of a program library or
-- "library" as specified in Subclause 10(2) of the Ada Reference Manual,
-- then an ASIS Context value can be mapped onto one or more implementor
-- libraries represented by Containers.
--
-------------------------------------------------------------------------------
```

-- 9.1 type Container

```
-------------------------------------------------------------------------------
--
-- The Container abstraction is a logical collection of compilation units.
-- For example, one container might hold compilation units which include Ada
-- predefined library units, another container might hold implementation-defined
-- packages. Alternatively, there might be 26 containers, each holding
-- compilation units that begin with their respective letter of the alphabet.
-- The point is that no implementation-independent semantics are associated
-- with a container; it is simply a logical collection.
--
-- ASIS implementations shall minimally map the Asis.Context to a list of
-- one ASIS Container whose Name is that of the Asis.Context Name.
-------------------------------------------------------------------------------

    type Container is private;
    Nil_Container : constant Container;

    function "=" (Left  : in Container;
                  Right : in Container)
                  Return Boolean is abstract;

-------------------------------------------------------------------------------
```

-- 9.2 type Container_List

```
-------------------------------------------------------------------------------

    type Container_List is
       array (List_Index range <>) of Container;

-------------------------------------------------------------------------------
```

-- 9.3 function Defining_Containers

```
-------------------------------------------------------------------------------

    function Defining_Containers (The_Context : in Asis.Context)
       return Container_List;

-------------------------------------------------------------------------------
-- The_Context - Specifies the Context to define
--
-- Returns a Container_List value that defines the single environment Context.
-- Each Container will have an Enclosing_Context that Is_Identical to the
-- argument The_Context.  The order of Container values in the list is not
-- defined.
--
-- Returns a minimal list of length one if the ASIS Ada implementation does
-- not support the concept of a program library.  In this case, the Container
-- will have the same name as the given Context.
--
-- Raises ASIS_Inappropriate_Context if The_Context is not open.
--
```

```
-------------------------------------------------------------------------------------
```
-- 9.4 function Enclosing_Context
```
-------------------------------------------------------------------------------------

    function Enclosing_Context (The_Container : in Container)
       return Asis.Context;

-------------------------------------------------------------------------------------
-- The_Container - Specifies the Container to query
--
-- Returns the Context value associated with the Container.
--
-- Returns the Context for which the Container value was originally obtained.
-- Container values obtained through the Defining_Containers query will always
-- remember the Context from which they were defined.
--
-- Because Context is limited private, this function is only intended to be
-- used to supply a Context parameter for other queries.
--
-- Raises ASIS_Inappropriate_Container if the Container is a Nil_Container.
--
-------------------------------------------------------------------------------------
```
-- 9.5 function Library_Unit_Declaration
```
-------------------------------------------------------------------------------------

    function Library_Unit_Declarations (The_Container : in Container)
                                        return Asis.Compilation_Unit_List;

-------------------------------------------------------------------------------------
-- The_Container - Specifies the Container to query
--
-- Returns a list of all library_unit_declaration and
-- library_unit_renaming_declaration  elements contained in the Container.  Individual
-- units will appear only once in an order that is not defined.
--
-- A Nil_Compilation_Unit_List is returned if there are no declarations of
-- library units within the Container.
--
-- This query will never return a unit with A_Configuration_Compilation or
-- a Nonexistent unit kind. It will never return a unit with A_Procedure_Body or
-- A_Function_Body unit kind even though the unit is interpreted as both the
-- declaration and body of a library procedure or library function. (Reference
-- Manual 10.1.4(4).
--
-- All units in the result will have an Enclosing_Container value that
-- Is_Identical to the Container.
--
-- Raises ASIS_Inappropriate_Context if the Enclosing_Context(Container)
-- is not open.
--
```

```
-------------------------------------------------------------------------------
```
-- 9.6 function Compilation_Unit_Bodies
```
-------------------------------------------------------------------------------

    function Compilation_Unit_Bodies (The_Container : in Container)
                                     return Asis.Compilation_Unit_List;

-------------------------------------------------------------------------------
-- The_Container - Specifies the Container to query
--
-- Returns a list of all library_unit_body and subunit elements contained in the
-- Container.  Individual units will appear only once in an order that is not
-- defined.
--
-- A Nil_Compilation_Unit_List is returned if there are no bodies within the
-- Container.
--
-- This query will never return a unit with A_Configuration_Compilation or
-- a nonexistent unit kind.
--
-- All units in the result will have an Enclosing_Container value that
-- Is_Identical to the Container.
--
-- Raises ASIS_Inappropriate_Context if the Enclosing_Context(Container)
-- is not open.
--
-------------------------------------------------------------------------------
```
-- 9.7 function Compilation_Units
```
-------------------------------------------------------------------------------

    function Compilation_Units (The_Container : in Container)
                               return Asis.Compilation_Unit_List;

-------------------------------------------------------------------------------
-- The_Container - Specifies the Container to query
--
-- Returns a list of all compilation units contained in the Container.
-- Individual units will appear only once in an order that is not defined.
--
-- A Nil_Compilation_Unit_List is returned if there are no units within the
-- Container.
--
-- This query will never return a unit with A_Configuration_Compilation or
-- a nonexistent unit kind.
--
-- All units in the result will have an Enclosing_Container value that
-- Is_Identical to the Container.
--
-- Raises ASIS_Inappropriate_Context if the Enclosing_Context(Container)
-- is not open.
--
-------------------------------------------------------------------------------
```
-- 9.8 function Is_Equal
```
-------------------------------------------------------------------------------

    function Is_Equal (Left  : in Container;
                       Right : in Container) return Boolean;

-------------------------------------------------------------------------------
-- Left    - Specifies the first Container
-- Right   - Specifies the second Container
--
-- Returns True if Left and Right designate Container values that contain the
-- same set of compilation units.  The Container values may have been defined
-- from different Context values.
--
```

```
-------------------------------------------------------------------------------------
```

-- 9.9 function Is_Identical

```
-------------------------------------------------------------------------------------

    function Is_Identical (Left  : in Container;
                           Right : in Container) return Boolean;

-------------------------------------------------------------------------------------
-- Left    - Specifies the first Container
-- Right   - Specifies the second Container
--
-- Returns True if Is_Equal(Left, Right) and the Container values have been
-- defined from Is_Equal Context values.
--
-------------------------------------------------------------------------------------
```

-- 9.10 function Name

```
-------------------------------------------------------------------------------------

    function Name (The_Container : in Container) return Wide_String;

-------------------------------------------------------------------------------------
-- The_Container - Specifies the Container to name
--
-- Returns the Name value associated with the Container.
--
-- Returns a null string if the Container is a Nil_Container.
--
-------------------------------------------------------------------------------------

private

    type Container is (Implementation_Defined);
    Nil_Container : constant Container := Implementation_Defined;

end Asis.Ada_Environments.Containers;
```

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

-- 10 package Asis.Compilation_Units

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
with Asis.Ada_Environments.Containers;
package Asis.Compilation_Units is
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- Asis.Compilation_Units encapsulates a set of queries that implement the
-- ASIS Compilation_Unit abstraction.
--
-- More than one compilation unit may be manipulated at one time.  (The exact
-- number is subject to implementation specific limitations.)
--
-- A specific Compilation_Unit value is valid (usable) for as long as the ASIS
-- Context variable, used to create it, remains open.  Once an ASIS Context is
-- closed, all associated Compilation_Unit values become invalid.  It is
-- erroneous to use an invalid Compilation_Unit value.
--
-------------------------------------------------------------------------------
```

-- 10.1 function Unit_Kind

```
-------------------------------------------------------------------------------

    function Unit_Kind (Compilation_Unit : in Asis.Compilation_Unit)
                       return Asis.Unit_Kinds;

-------------------------------------------------------------------------------
-- Compilation_Unit    - Specifies the compilation unit to query
--
-- Returns the Unit_Kinds value of the compilation unit.
-- Returns Not_A_Unit for a Nil_Compilation_Unit.
--
-- All Unit_Kinds are expected.
--
-- Returns An_Unknown_Unit for any compilation unit that exists, but that
-- does not have semantic element information available through ASIS.
--
-- Returns a nonexistent kind for units that have name-only entries in the
-- environment Context.  Such entries may exist for names because:
--
-- - They represent an illegal compilation unit added to the environment.
--
-- - They are referenced by some existing unit, but the program text for the
--   referenced unit has never been supplied, compiled, or otherwise
--   inserted into the environment.
--
-- - They represent a separate subunit that has never been supplied,
--   compiled, or otherwise inserted into the environment.
--
-- - The unit may have existed at one time but the semantic information is no
--   longer available.  It may be inconsistent, have been removed by some
--   user or Ada environment operations, or simply have been lost as the
--   result of some sort of failure.
--
-------------------------------------------------------------------------------
```

-- 10.2 function Unit_Class

```
-------------------------------------------------------------------------------

    function Unit_Class (Compilation_Unit : in Asis.Compilation_Unit)
                        return Asis.Unit_Classes;

-------------------------------------------------------------------------------
-- Compilation_Unit    - Specifies the compilation unit to query
--
-- Returns the Unit_Classes value of the compilation unit.
-- Returns Not_A_Class for a Nil_Compilation_Unit.
--
-- All Unit_Kinds are expected.
--
```

```
-------------------------------------------------------------------------------------
```
-- 10.3 function Unit_Origin
```
-------------------------------------------------------------------------------------

    function Unit_Origin (Compilation_Unit : in Asis.Compilation_Unit)
                    return Asis.Unit_Origins;

-------------------------------------------------------------------------------------
-- Compilation_Unit    - Specifies the compilation unit to query
--
-- Returns the Unit_Origins value of the unit.
-- Returns Not_An_Origin for a compilation_unit whose Unit_Kind is
-- Not_A_Unit, An_Unknown_Unit, A_Nonexistent_Declaration, or
-- A_Nonexistent_Body.
--
-- All Unit_Kinds are expected.
--
-------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------
```
-- 10.4 function Enclosing_Context
```
-------------------------------------------------------------------------------------

    function Enclosing_Context (Compilation_Unit : in Asis.Compilation_Unit)
                               return Asis.Context;

-------------------------------------------------------------------------------------
-- Compilation_Unit    - Specifies the unit whose Context is required
--
-- Returns the Context containing the compilation unit.
--
-- Compilation units always remember the ASIS Context and Container from
-- which they were obtained.
--
-- Because Context is limited private, this function is only intended to be
-- used to supply a Context parameter for other queries.  This conveniently
-- eliminates the need to make the original Context visible at the place of
-- each call where a Context parameter is required.
--
-- Two Compilation_Unit values, that represent the same physical compilation
-- units (same Ada implementor Context implementation unit value) will test as
-- Is_Equal, but not Is_Identical, if they were obtained from different open
-- ASIS Context variables.
--
-- Raises ASIS_Inappropriate_Compilation_Unit if the unit is a
-- Nil_Compilation_Unit.
--
-------------------------------------------------------------------------------------
```
-- 10.5 function Enclosing_Container
```
-------------------------------------------------------------------------------------

    function Enclosing_Container (Compilation_Unit : in Asis.Compilation_Unit)
                            return Asis.Ada_Environments.Containers.Container;

-------------------------------------------------------------------------------------
-- Compilation_Unit    - Specifies the unit whose Container is required
--
-- Returns the Container of the Context containing the compilation unit.
-- Compilation units always remember the ASIS Context and Container from
-- which they were obtained.
--
-- Raises ASIS_Inappropriate_Compilation_Unit if the unit is a
-- Nil_Compilation_Unit.
--
```

```
-------------------------------------------------------------------------------
```
-- 10.6 function Library_Unit_Declaration
```
-------------------------------------------------------------------------------

    function Library_Unit_Declaration (Name        : in Wide_String;
                                       The_Context : in Asis.Context)
                                       return Asis.Compilation_Unit;

-------------------------------------------------------------------------------
-- Name        - Specifies the defining program unit name
-- The_Context - Specifies a program Context environment
--
-- Returns the library_unit_declaration or library_unit_renaming_declaration
-- with the name, contained in The_Context.
--
-- This query will never return a unit with A_Configuration_Compilation or
-- a nonexistent unit kind. It will never return a unit with A_Procedure_Body or
-- A_Function_Body unit kind even though the unit is interpreted as both the
-- declaration and body of a library procedure or library function. (Reference
-- Manual 10.1.4(4).
--
-- A Nil_Compilation_Unit is returned if no such declaration exists.
--
-- Any non-Nil result will have an Enclosing_Context value that Is_Identical
-- to the Context.  Never returns a unit with a nonexistent unit kind.
--
-------------------------------------------------------------------------------
```
-- 10.7 function Compilation_Unit_Body
```
-------------------------------------------------------------------------------

    function Compilation_Unit_Body (Name        : in Wide_String;
                                    The_Context : in Asis.Context)
                                    return Asis.Compilation_Unit;

-------------------------------------------------------------------------------
-- Name        - Specifies the defining_program_unit_name
-- The_Context - Specifies a program Context environment
--
-- Returns the library_unit_body or subunit with the name, contained
-- in the library.
--
-- A Nil_Compilation_Unit is returned if no such body exists.
--
-- Any non-Nil result will have an Enclosing_Context value that Is_Identical
-- to The_Context.  Never returns a unit with a nonexistent unit kind.
--
-------------------------------------------------------------------------------
```
-- 10.8 function Library_Unit_Declarations
```
-------------------------------------------------------------------------------

    function Library_Unit_Declarations (The_Context : in Asis.Context)
                                        return Asis.Compilation_Unit_List;

-------------------------------------------------------------------------------
-- The_Context - Specifies a program Context environment
--
-- Returns a list of all library_unit_declaration and
-- library_unit_renaming_declaration elements contained in The_Context.  Individual
-- units will appear only once in an order that is not defined.
--
-- A Nil_Compilation_Unit_List is returned if there are no declarations of
-- library units within The_Context.
--
-- This query will never return a unit with A_Configuration_Compilation or
-- a nonexistent unit kind. It will never return a unit with A_Procedure_Body or
-- A_Function_Body unit kind even though the unit is interpreted as both the
-- declaration and body of a library procedure or library function. (Reference
-- Manual 10.1.4(4).
--
-- All units in the result will have an Enclosing_Context value that
-- Is_Identical to The_Context.
--
```

```
-------------------------------------------------------------------------------
```
-- 10.9 function Compilation_Unit_Bodies
```
-------------------------------------------------------------------------------

    function Compilation_Unit_Bodies (The_Context : in Asis.Context)
                                     return Asis.Compilation_Unit_List;

-------------------------------------------------------------------------------
-- The_Context - Specifies a program Context environment
--
-- Returns a list of all library_unit_body and subunit elements contained in
-- The_Context.  Individual units will appear only once in an order that is not
-- defined.
--
-- A Nil_Compilation_Unit_List is returned if there are no bodies within
-- The_Context.
--
-- This query will never return a unit with A_Configuration_Compilation or
-- a nonexistent unit kind.
--
-- All units in the result will have an Enclosing_Context value that
-- Is_Identical to The_Context.
--
-------------------------------------------------------------------------------
```
-- 10.10 function Compilation_Units
```
-------------------------------------------------------------------------------

    function Compilation_Units (The_Context : in Asis.Context)
                               return Asis.Compilation_Unit_List;

-------------------------------------------------------------------------------
-- The_Context - Specifies a program Context environment
--
-- Returns a list of all compilation units contained in The_Context.
-- Individual units will appear only once in an order that is not defined.
--
-- A Nil_Compilation_Unit_List is returned if there are no units within
-- The_Context.
--
-- This query will never return a unit with A_Configuration_Compilation or
-- a nonexistent unit kind.
--
-- All units in the result will have an Enclosing_Context value that
-- Is_Identical to The_Context.
--
-------------------------------------------------------------------------------
```
-- 10.11 function Corresponding_Children
```
-------------------------------------------------------------------------------

    function Corresponding_Children (Library_Unit : in Asis.Compilation_Unit)
                  return Asis.Compilation_Unit_List;

    function Corresponding_Children (Library_Unit : in Asis.Compilation_Unit;
                                     The_Context  : in Asis.Context)
                  return Asis.Compilation_Unit_List;

-------------------------------------------------------------------------------
-- Library_Unit - Specifies the library unit whose children are desired
-- The_Context  - Specifies a program Context environment
--
-- Returns a list of the child units for the given parent library unit.
--
-- Both the declaration and body (if any) of each child unit are returned.
-- Descendants beyond immediate children (i.e., children of children) are not
-- returned by this query.
--
-- Use the compilation unit relationship queries
-- with a Relation_Kinds of Descendants to create a list of children, children
-- of children, and so on.
--
-- Returns a Nil_Compilation_Unit_List for all library unit arguments that
-- do not have any child units contained in The_Context.
```

```
--
-- These two function calls will always produce identical results:
--
--     Units := Corresponding_Children ( Unit );
--     Units := Corresponding_Children ( Unit, Enclosing_Context ( Unit ));
--
-- Any non-Nil result will have an Enclosing_Context value that Is_Identical
-- to The_Context.
--
-- The Enclosing_Context for any non-Nil result will always be The_Context,
-- regardless of the Enclosing_Context value for the Library_Unit argument.
-- This query is one means of obtaining (Is_Equal) child units
-- from separate ASIS Context values whose underlying implementations
-- overlap.
--
-- Appropriate Unit_Kinds:
--      A_Package
--      A_Generic_Package
--      A_Package_Instance
--
-- Returns Unit_Kinds:
--      A_Procedure
--      A_Function
--      A_Package
--      A_Generic_Procedure
--      A_Generic_Function
--      A_Generic_Package
--      A_Procedure_Instance
--      A_Function_Instance
--      A_Package_Instance
--      A_Procedure_Renaming
--      A_Function_Renaming
--      A_Package_Renaming
--      A_Generic_Procedure_Renaming
--      A_Generic_Function_Renaming
--      A_Generic_Package_Renaming
--      A_Procedure_Body
--      A_Function_Body
--      A_Package_Body
--      An_Unknown_Unit
--
-- If the declaration of a child is inconsistent with the argument of the
-- query, neither the declaration nor the body is returned.  If the
-- declaration of a child is consistent with the argument, but the body
-- is not, the declaration is returned, and for the body, the result of
-- the Corresponding_Body query applied to the declaration is returned.
--
-------------------------------------------------------------------------------------
```

-- 10.12 function Corresponding_Parent_Declaration

```
-------------------------------------------------------------------------------------

    function Corresponding_Parent_Declaration
                (Library_Unit : in Asis.Compilation_Unit)
                return Asis.Compilation_Unit;

    function Corresponding_Parent_Declaration
                (Library_Unit : in Asis.Compilation_Unit;
                 The_Context  : in Asis.Context)
                return Asis.Compilation_Unit;

-------------------------------------------------------------------------------------
-- Library_Unit - Specifies the unit whose parent is desired
-- The_Context  - Specifies a program Context environment
--
-- Returns the parent unit of the given library unit.
--
-- Returns a Nil_Compilation_Unit if the Library_Unit argument represents
-- package Standard.  Root Library_Unit arguments return the package Standard.
--
-- Returns A_Nonexistent_Declaration when the Library_Unit has a
-- parent_unit_name denoted in the defining_program_unit_name but the parent
-- unit is not contained in The_Context.
--
```

```
-- These two function calls will always produce identical results:
--
--     Unit := Corresponding_Parent_Declaration ( Unit );
--     Unit := Corresponding_Parent_Declaration ( Unit, Enclosing_Context ( Unit ));
--
-- Any non-Nil result will have an Enclosing_Context value that Is_Identical
-- to The_Context.
--
-- The Enclosing_Context for any non-Nil result will always be The_Context,
-- regardless of the Enclosing_Context value for the Library_Unit
-- argument.  This query is one means of obtaining (Is_Equal) parent units
-- from separate ASIS Context values whose underlying implementations
-- overlap.
--
-- Appropriate Unit_Kinds:
--      A_Procedure
--      A_Function
--      A_Package
--      A_Generic_Procedure
--      A_Generic_Function
--      A_Generic_Package
--      A_Procedure_Instance
--      A_Function_Instance
--      A_Package_Instance
--      A_Procedure_Renaming
--      A_Function_Renaming
--      A_Package_Renaming
--      A_Generic_Procedure_Renaming
--      A_Generic_Function_Renaming
--      A_Generic_Package_Renaming
--      A_Procedure_Body
--      A_Function_Body
--      A_Package_Body
--
-- Returns Unit_Kinds:
--      Not_A_Unit
--      A_Package
--      A_Generic_Package
--      A_Package_Instance
--      A_Nonexistent_Declaration
--      An_Unknown_Unit
--
-- If a parent is inconsistent with a child passed as the argument,
-- A_Nonexistent_Declaration shall be returned.
--
-------------------------------------------------------------------------------------
```

-- 10.13 function Corresponding_Declaration

```
-------------------------------------------------------------------------------------

    function Corresponding_Declaration
                (Library_Item : in Asis.Compilation_Unit)
                return Asis.Compilation_Unit;

    function Corresponding_Declaration
                (Library_Item : in Asis.Compilation_Unit;
                 The_Context  : in Asis.Context)
                return Asis.Compilation_Unit;

-------------------------------------------------------------------------------------
-- Library_Item - Specifies the library_item whose declaration is desired
-- The_Context  - Specifies a program Context environment
--
-- Returns the corresponding library_unit_declaration, if any, for the
-- library_unit_body.  The corresponding library unit is the unit upon which
-- the library_unit_body depends semantically.
--
-- Returns a unit that Is_Equal to the argument if:
--
-- - the argument is a library_unit_declaration,
--   a library_unit_renaming_declaration, or a subunit.
--
-- - the argument is A_Nonexistent_Declaration or A_Nonexistent_Body.
--
```

```
-- Returns a Nil_Compilation_Unit for library_unit_body arguments that do
-- not have a corresponding library unit contained in The_Context.
--
-- All Unit_Kinds are appropriate except Not_A_Unit.
--
-- Appropriate Unit_Kinds:
--      A_Procedure_Body
--      A_Function_Body
--      A_Package_Body
--      An_Unknown_Unit            -- See Implementation Permissions
--
-- Appropriate Unit_Kinds returning the argument Library_Item:
--      A_Procedure
--      A_Function
--      A_Package
--      A_Generic_Procedure
--      A_Generic_Function
--      A_Generic_Package
--      A_Procedure_Instance
--      A_Function_Instance
--      A_Package_Instance
--      A_Procedure_Renaming
--      A_Function_Renaming
--      A_Package_Renaming
--      A_Generic_Procedure_Renaming
--      A_Generic_Function_Renaming
--      A_Generic_Package_Renaming
--      A_Procedure_Body_Subunit
--      A_Function_Body_Subunit
--      A_Package_Body_Subunit
--      A_Task_Body_Subunit
--      A_Protected_Body_Subunit
--      A_Nonexistent_Declaration
--      A_Nonexistent_Body
--
-- Returns all Unit Kinds.
--
-- If the declaration of an argument Element is inconsistent with the
-- argument, A_Nonexistent_Declaration shall be returned. (For a unit
-- A_Procedure_Body or A_Function_Body kind, the solution may be in any
-- case, to return Nil_Compilation_Unit if the unit is of
-- A_Public_Declaration_And_Body kind.)
--
--|IR Implementation Requirements:
--|IR
--|IR Any non-Nil result will have an Enclosing_Context value that
--|IR Is_Identical to The_Context.
--|IR
--|IR These two function calls will always produce identical results:
--|IR
--|IR    Unit := Corresponding_Declaration( Unit );
--|IR    Unit := Corresponding_Declaration( Unit, Enclosing_Context( Unit ));
--|IR
--|IR The Enclosing_Context for any non-Nil result will always be The_Context,
--|IR regardless of the Enclosing_Context value for the Library_Item
--|IR argument.  This query is one means of obtaining corresponding
--|IR (Is_Equal) units from separate ASIS Context values whose underlying
--|IR implementations overlap.
--
--|IP Implementation Permissions:
--|IP
--|IP The handling of An_Unknown_Unit is implementation specific.  The
--|IP expected use for An_Unknown_Unit is to hide proprietary implementation
--|IP details contained within unit bodies.  In these cases, it should be
--|IP possible to obtain an appropriate library_unit_declaration when
--|IP starting with An_Unknown_Unit.  Some implementors may choose to simply
--|IP return the An_Unknown_Unit argument in all cases.
--
```

```
-------------------------------------------------------------------------------------
```

-- 10.14 function Corresponding_Body

```
-------------------------------------------------------------------------------------

    function Corresponding_Body
                (Library_Item : in Asis.Compilation_Unit)
                return Asis.Compilation_Unit;

    function Corresponding_Body
                (Library_Item : in Asis.Compilation_Unit;
                 The_Context  : in Asis.Context)
                return Asis.Compilation_Unit;

-------------------------------------------------------------------------------------
-- Library_Item - Specifies the library_item whose body is desired
-- The_Context  - Specifies a program Context environment
--
-- Returns the corresponding library_unit_body, if any, for the
-- library_unit_declaration.  The corresponding library_unit_body is the unit
-- that depends semantically on the library_unit_declaration.
--
-- Returns a unit that Is_Equal to the argument if:
--
-- - the argument is a an instance of a library_unit_declaration,
--   a library_unit_body, a library_unit_renaming_declaration, or a subunit.
--
-- - the argument is A_Nonexistent_Declaration or A_Nonexistent_Body.
--
-- Returns a Nil_Compilation_Unit for library_unit_declaration arguments that
-- do not have a corresponding library_unit_body contained in The_Context.
--
-- All Unit_Kinds are appropriate except Not_A_Unit.
--
-- Appropriate Unit_Kinds:
--      A_Procedure
--      A_Function
--      A_Package
--      A_Generic_Procedure
--      A_Generic_Function
--      A_Generic_Package
--      An_Unknown_Unit            -- See Implementation Permissions
--
-- Appropriate Unit_Kinds returning the argument Library_Item:
--      A_Procedure_Body
--      A_Function_Body
--      A_Package_Body
--      A_Procedure_Instance
--      A_Function_Instance
--      A_Package_Instance
--      A_Procedure_Renaming
--      A_Function_Renaming
--      A_Package_Renaming
--      A_Generic_Procedure_Renaming
--      A_Generic_Function_Renaming
--      A_Generic_Package_Renaming
--      A_Procedure_Body_Subunit
--      A_Function_Body_Subunit
--      A_Package_Body_Subunit
--      A_Task_Body_Subunit
--      A_Protected_Body_Subunit
--      A_Nonexistent_Declaration
--      A_Nonexistent_Body
--
-- Returns all Unit Kinds.
--
-- If the argument Element requires a body to be presented to make up a
-- complete partition containing this Element, but The_Context does not
-- contain the corresponding body, or the body contained in The_Context
-- is inconsistent with the argument Element, A_Nonexistent_Body shall
-- be returned.
--
--|IR Implementation Requirements:
--|IR
```

```
--|IR Any non-Nil result will have an Enclosing_Context value that
--|IR Is_Identical to The_Context.
--|IR
--|IR These two function calls will always produce identical results:
--|IR
--|IR     Unit := Corresponding_Body( Unit );
--|IR     Unit := Corresponding_Body( Unit, Enclosing_Context ( Unit ));
--|IR
--|IR The Enclosing_Context for any non-Nil result will always be The_Context,
--|IR regardless of the Enclosing_Context value for the Library_Item
--|IR argument.  This query is one means of obtaining corresponding
--|IR (Is_Equal) units from separate ASIS Context values whose underlying
--|IR implementations overlap.
--
--|IP Implementation Permissions:
--|IP
--|IP The handling of An_Unknown_Unit is implementation specific.  The
--|IP expected use for An_Unknown_Unit is to hide proprietary implementation
--|IP details contained within unit bodies.  In some cases, it could be possible
--|IP to obtain an appropriate library_unit_body when starting with
--|IP An_Unknown_Unit.  Some implementors may choose to simply return the
--|IP An_Unknown_Unit argument in all cases.
--
-------------------------------------------------------------------------------------
```

-- 10.15 function Is_Nil

```
-------------------------------------------------------------------------------------

    function Is_Nil (Right : in Asis.Compilation_Unit)
                    return Boolean;

-------------------------------------------------------------------------------------
-- Right   - Specifies the unit to test
--
-- Returns True if the compilation_unit is a Nil_Compilation_Unit.
--
-------------------------------------------------------------------------------------
```

-- 10.16 function Is_Nil

```
-------------------------------------------------------------------------------------

    function Is_Nil (Right : in Asis.Compilation_Unit_List)
                    return Boolean;

-------------------------------------------------------------------------------------
-- Right   - Specifies the unit list to test
--
-- Returns True if the compilation_unit list has a length of zero.
--
-------------------------------------------------------------------------------------
```

-- 10.17 function Is_Equal

```
-------------------------------------------------------------------------------------

    function Is_Equal (Left  : in Asis.Compilation_Unit;
                       Right : in Asis.Compilation_Unit) return Boolean;

-------------------------------------------------------------------------------------
-- Left    - Specifies the first unit to compare
-- Right   - Specifies the second unit to compare
--
-- Returns True if Left and Right represent the same physical compilation unit
-- or if both are Nil_Compilation_Unit values.  The two units may or may not
-- be from the same ASIS Context variable. ("The same physical compilation
-- unit" have the same version, as defined by Reference Manual E.3(5)
-- and the same program text.)
--
-- Two nonexistent units are Is_Equal if they have the same Name and Unit_Kind.
--
```

```
-------------------------------------------------------------------------------------
```
-- 10.18 function Is_Identical
```
-------------------------------------------------------------------------------------

    function Is_Identical (Left  : in Asis.Compilation_Unit;
                           Right : in Asis.Compilation_Unit) return Boolean;

-------------------------------------------------------------------------------------
-- Left    - Specifies the first unit to compare
-- Right   - Specifies the second unit to compare
--
-- Returns True if Left and Right represent the same physical compilation
-- unit, from the same open ASIS Context variable, or, if both are
-- Nil_Compilation_Unit values. (“The same physical compilation
-- unit” have the same version, as defined by Reference Manual E.3(5)
-- and the same program text.)
--
-- Two nonexistent units are Is_Identical if they have the same
-- Unique_Name and the same Enclosing_Context.
--
-------------------------------------------------------------------------------------
```
-- 10.19 function Unit_Full_Name
```
-------------------------------------------------------------------------------------

    function Unit_Full_Name (Compilation_Unit : in Asis.Compilation_Unit)
                            return Wide_String;

-------------------------------------------------------------------------------------
-- Compilation_Unit    - Specifies the unit whose name is desired
--
-- Returns the string image of the fully expanded Ada name of the given
-- compilation unit.  This may be a simple name ("A") of a root library
-- unit, or an expanded name ("A.B") of a subunit or non-root child unit.
-- An expanded name shall contain the full parent_unit_name as its prefix.

-- Returns a null string only if A_Configuration_Compilation or a
-- Nil_Compilation_Unit is given.
--
-- The case of names returned by this query may vary between implementations.
-- Implementors are encouraged, but not required, to return names in the
-- same case as was used in the original compilation text.
--
-- All Unit_Kinds are appropriate.
--
-------------------------------------------------------------------------------------
```
-- 10.20 function Unique_Name
```
-------------------------------------------------------------------------------------

    function Unique_Name
          (Compilation_Unit : in Asis.Compilation_Unit)
                return Wide_String;

-------------------------------------------------------------------------------------
-- Compilation_Unit    - Specifies the unit whose name is desired
--
-- Returns a string that uniquely identifies the given compilation unit
-- within the underlying Ada Context implementation.  The result may vary
-- depending on the ASIS implementation.  The unique name may include the name
-- and parameters of the Context, file system paths, library files, version
-- numbers, kind, or any other information that an implementation may need
-- to uniquely identify the compilation unit.
--
-- Returns a null string only if a Nil_Compilation_Unit is given.
--
-- All Unit_Kinds are appropriate.
--
```

```
-------------------------------------------------------------------------------
```
-- 10.21 function Exist
```
-------------------------------------------------------------------------------

    function Exists (Compilation_Unit : in Asis.Compilation_Unit)
                    return Boolean;

-------------------------------------------------------------------------------
-- Compilation_Unit    - Specifies the unit to test
--
-- Returns False for any unit with Not_A_Unit or nonexistent kind.
-- Returns True for all other unit kinds.
--
-- All Unit_Kinds are expected.
--
-------------------------------------------------------------------------------
```
-- 10.22 function Can_Be_Main_Program
```
-------------------------------------------------------------------------------

    function Can_Be_Main_Program (Compilation_Unit : in Asis.Compilation_Unit)
                                 return Boolean;

-------------------------------------------------------------------------------
-- Compilation_Unit    - Specifies the unit to test
--
-- Returns True if the Compilation_Unit exists and is a subprogram
-- library_unit_declaration, library_unit_renaming_declaration, or
-- library_unit_body that can be used as a main subprogram.  See Reference
-- Manual 10.2(7).
--
-- Returns False otherwise.
--
-- Results of this function may vary according to the requirements an Ada
-- implementation may impose on a main subprogram.
--
-- All Unit_Kinds are expected.
--
-------------------------------------------------------------------------------
```
-- 10.23 function Is_Body_Required
```
-------------------------------------------------------------------------------

    function Is_Body_Required (Compilation_Unit : in Asis.Compilation_Unit)
                              return Boolean;

-------------------------------------------------------------------------------
-- Compilation_Unit    - Specifies the unit to test
--
-- Returns True if the Compilation_Unit exists and is a library
-- package_declaration that requires a body.  See Reference Manual 7.2(4).
--
-- All Unit_Kinds are expected.
--
-------------------------------------------------------------------------------
```
-- 10.24 function Text_Name
```
-------------------------------------------------------------------------------

     function Text_Name (Compilation_Unit : in Asis.Compilation_Unit)
                        return Wide_String;

-------------------------------------------------------------------------------
-- Compilation_Unit    - Specifies the unit whose text name is desired
--
-- Returns the name of the text, or other structure, that was the source
-- of the compilation that resulted in this Compilation_Unit.  Returns a
-- null string if the unit has a Nil or nonexistent kind, or if the text
-- name is not available for any reason.
--
```

```
-- Ada has no concept of source or text file.
-- Text_Name availability is a required feature of ASIS.
-- Results of this function may vary among implementations.
--
-- All Unit_Kinds are appropriate.
--
---------------------------------------------------------------------------------------
```

-- 10.25 function Text_Form

```
---------------------------------------------------------------------------------------

    function Text_Form (Compilation_Unit : in Asis.Compilation_Unit)
                        return Wide_String;

---------------------------------------------------------------------------------------
-- Compilation_Unit    - Specifies the unit whose text form is desired
--
-- Returns the Form parameter (as for Text_Io.Open) for the text, or
-- other structure, that was the source of the compilation that resulted in
-- this Compilation_Unit.  Returns a null string if the unit has a Nil or
-- nonexistent kind, if the text was created with an empty Form parameter,
-- or if the text Form parameter value is not available for any reason.
--
-- Ada has no concept of source or text file.
-- Text_Form availability is a required feature of ASIS.
-- Results of this function may vary among implementations.
--
-- All Unit_Kinds are appropriate.
--
---------------------------------------------------------------------------------------
```

-- 10.26 function Object_Name

```
---------------------------------------------------------------------------------------

      function Object_Name (Compilation_Unit : in Asis.Compilation_Unit)
                            return Wide_String;

---------------------------------------------------------------------------------------
-- Compilation_Unit    - Specifies the unit whose object name is desired
--
-- Returns the name of the object, or other structure, that contains the
-- binary result of the compilation for this Compilation_Unit.  Returns
-- a null string if the unit has a Nil or nonexistent kind, or if the
-- object name is not available for any reason.
--
-- All Unit_Kinds are appropriate.
--
---------------------------------------------------------------------------------------
```

-- 10.27 function Object_Form

```
---------------------------------------------------------------------------------------

    function Object_Form (Compilation_Unit : in Asis.Compilation_Unit)
                          return Wide_String;

---------------------------------------------------------------------------------------
-- Compilation_Unit    - Specifies the unit whose object form is desired
--
-- Returns the Form parameter (as for Text_Io.Open) for the object, or
-- other structure, that was the machine-code result of the compilation of
-- this Compilation_Unit.  Returns a null string if the unit has a Nil or
-- nonexistent kind, if the object was created with an empty Form parameter,
-- or if the object Form parameter value is not available for any reason.
--
-- All Unit_Kinds are appropriate.
--
```

```
-------------------------------------------------------------------------------------
```
-- 10.28 function Compilation_Command_Line_Options
```
-------------------------------------------------------------------------------------

    function Compilation_Command_Line_Options
                (Compilation_Unit : in Asis.Compilation_Unit)
                return Wide_String;

-------------------------------------------------------------------------------------
-- Compilation_Unit    - Specifies the unit to query
--
-- Returns the command line options used to compile the Compilation_Unit.
-- Returns null string if the unit has a Nil or nonexistent unit kind, or
-- if the command line options are not available for any reason.
--
-- All Unit_Kinds are appropriate.
--
-------------------------------------------------------------------------------------
```
-- 10.29 function Has_Attribute
```
-------------------------------------------------------------------------------------

    function Has_Attribute (Compilation_Unit : in Asis.Compilation_Unit;
                            Attribute        : in Wide_String) return Boolean;

-------------------------------------------------------------------------------------
-- Compilation_Unit    - Specifies the unit to query
-- Attribute           - Specifies the name of the attribute to query
--
-- Returns True if the compilation unit has the given attribute.
--
-- Returns False if the unit is a Nil_Compilation_Unit argument, the
-- Attribute does not exist, or the implementation does not support attributes.
--
-- All Unit_Kinds are expected.
--
-- Results of this query may vary across ASIS implementations.
--
-------------------------------------------------------------------------------------
```
-- 10.30 function Attribute_Value_Delimiter
```
-------------------------------------------------------------------------------------

    function Attribute_Value_Delimiter return Wide_String;

-------------------------------------------------------------------------------------
-- Returns the string used as a delimiter separating individual values
-- within the string Attribute_Values of a compilation unit.
--
-- Results of this query may vary across ASIS implementations.  The result
-- can be a null string for implementations that do not support attributes,
-- or that do not support more than one attribute.
--
-------------------------------------------------------------------------------------
```
-- 10.31 function Attribute_Values
```
-------------------------------------------------------------------------------------

    function Attribute_Values
                (Compilation_Unit : in Asis.Compilation_Unit;
                 Attribute        : in Wide_String)
                return Wide_String;

-------------------------------------------------------------------------------------
-- Compilation_Unit    - Specifies the unit to query
-- Attribute           - Specifies the name of the attribute to query
--
-- Returns a string containing zero or more images of values that are
-- associated with the given attribute.  When more than one value is returned,
-- the Attribute_Value_Delimiter string is used to separate the individual
-- values.  Returns a null string if the unit is a Nil_Compilation_Unit
-- argument, the unit has no values for this Attribute, or the implementation
-- does not support attributes.
--
```

```
-- All Unit_Kinds are appropriate.
--
-- Results of this query may vary across ASIS implementations.
--
-----------------------------------------------------------------------------------------
```

-- 10.32 function Subunits

```
-----------------------------------------------------------------------------------------

    function Subunits (Parent_Body : in Asis.Compilation_Unit)
                      return Asis.Compilation_Unit_List;

    function Subunits (Parent_Body : in Asis.Compilation_Unit;
                       The_Context : in Asis.Context)
                      return Asis.Compilation_Unit_List;

-----------------------------------------------------------------------------------------
-- Parent_Body - Specifies the parent unit to query
-- The_Context - Specifies the program Context to use for context
--
-- Returns a complete list of subunit values, with one value for each body
-- stub that appears in the given Parent_Body.  Returns a
-- Nil_Compilation_Unit_List if the parent unit does not contain any body
-- stubs.  Every unit in the result will have an Enclosing_Context that
-- Is_Identical to The_Context.
--
-- These two function calls will always produce identical results:
--
--      SUnits := Subunits ( PUnit );
--      SUnits := Subunits ( PUnit, Enclosing_Context ( PUnit ));
--
-- The result may include unit values with a nonexistent unit kind.  It
-- includes values for subunits that exist in The_Context as
-- well as values for subunits that do not exist, but whose name can be
-- deduced from the body stub and the name of the parent unit.  These
-- nonexistent units are known to be library_unit_body elements so their unit
-- kind is A_Nonexistent_Body.
--
-- Subunit lists are also available through the Semantic_Dependence_Order
-- query using the Family relation.
--
-- Raises ASIS_Inappropriate_Compilation_Unit if the unit is a
-- Nil_Compilation_Unit.
--
-- If a subunit is absent or if it is inconsistent with the argument Element,
-- A_Nonexistent_Body shall be returned for it.
--
-- Returns Unit_Kinds:
--       A_Nonexistent_Body
--       A_Procedure_Body_Subunit
--       A_Function_Body_Subunit
--       A_Package_Body_Subunit
--       A_Task_Body_Subunit
--       A_Protected_Body_Subunit
--
-----------------------------------------------------------------------------------------
```

-- 10.33 function Corresponding_Subunit_Parent_Body

```
-----------------------------------------------------------------------------------------

    function Corresponding_Subunit_Parent_Body
               (Subunit : in Asis.Compilation_Unit)
                return Asis.Compilation_Unit;

    function Corresponding_Subunit_Parent_Body
               (Subunit     : in Asis.Compilation_Unit;
                The_Context : in Asis.Context)
                return Asis.Compilation_Unit;

-----------------------------------------------------------------------------------------
-- Subunit     - Specifies the subunit to query
-- The_Context - Specifies the program Context to use for context
--
```

```
-- Returns the Compilation_Unit containing the body stub of the given Subunit.
-- Returns a Nil_Compilation_Unit if the subunit parent is not contained in
-- The_Context.  Any non-Nil result will have an Enclosing_Context value that
-- Is_Identical to The_Context.
--
-- These two function calls will always produce identical results:
--
--     PUnit := Corresponding_Subunit_Parent_Body ( SUnit );
--     PUnit := Corresponding_Subunit_Parent_Body ( SUnit,
--                                    Enclosing_Context ( SUnit ));
--
-- Appropriate Unit_Kinds:
--      A_Procedure_Body_Subunit
--      A_Function_Body_Subunit
--      A_Package_Body_Subunit
--      A_Task_Body_Subunit
--      A_Protected_Body_Subunit
--
-- Returns Unit_Kinds:
--      A_Procedure_Body
--      A_Function_Body
--      A_Package_Body
--      A_Procedure_Body_Subunit
--      A_Function_Body_Subunit
--      A_Package_Body_Subunit
--      A_Task_Body_Subunit
--      A_Protected_Body_Subunit
--
-- If the corresponding body does not exist in The_Context, or if it exists,
-- but is inconsistent with the argument Element, then A_Nonexistent_Body
-- shall be returned.
--
---------------------------------------------------------------------------------------
-- To locate the parent of a subunit that is not itself a subunit,
-- repeatedly call Corresponding_Subunit_Parent_Body until a unit that
-- is not a subunit is returned.
--
---------------------------------------------------------------------------------------
```

-- 10.34 function Debug_Image

```
---------------------------------------------------------------------------------------

     function Debug_Image (Compilation_Unit : in Asis.Compilation_Unit)
                          return Wide_String;

---------------------------------------------------------------------------------------
-- Compilation_Unit  - Specifies a unit to convert
--
-- Returns a string value containing implementation-defined debug
-- information associated with the compilation unit.
--
-- The return value uses Asis.Text.Delimiter_Image to separate the lines
-- of multi-line results.  The return value does not end with
-- Asis.Text.Delimiter_Image.
--
-- These values are intended for two purposes.  They are suitable for
-- inclusion in problem reports sent to the ASIS implementor.  They can be
-- presumed to contain information useful when debugging the implementation
-- itself. They are also suitable for use by the ASIS application when
-- printing simple application debugging messages during application
-- development.  They are intended to be, to some worthwhile degree,
-- intelligible to the user.
--
---------------------------------------------------------------------------------------

end Asis.Compilation_Units;
```

```
---------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------
```

-- 11 package Asis.Compilation_Units.Times

```
---------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------
with Ada.Calendar;
package Asis.Compilation_Units.Times is
---------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------
-- Asis.Compilation_Units.Times encapsulates the time related functions used
-- within ASIS.
---------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------
```

-- 11.1 type Time

```
---------------------------------------------------------------------------------------
-- ASIS uses the predefined Ada.Calendar.Time.
-- ASIS uses the predefined Standard.Duration.
-- The constant Nil_ASIS_Time is defined to support time queries where a
-- time is unavailable/unknown.
---------------------------------------------------------------------------------------

     Nil_ASIS_Time : constant Ada.Calendar.Time :=
           Ada.Calendar.Time_Of (Year    => 1901,
                                 Month   => 1,
                                 Day     => 1,
                                 Seconds => 0.0);

---------------------------------------------------------------------------------------
```

-- 11.2 function Time_Of_Last_Update

```
---------------------------------------------------------------------------------------

    function Time_Of_Last_Update (Compilation_Unit : in Asis.Compilation_Unit)
                                 return Ada.Calendar.Time;

---------------------------------------------------------------------------------------
-- Compilation_Unit    - Specifies the unit to query
--
-- Returns the time that this physical compilation unit was most recently
-- updated in its implementor’s Ada Environment.  This will often be the
-- time of its last compilation.  The exact significance of the result is
-- implementation specific.
-- Returns Nil_ASIS_Time if the unit has a Nil or nonexistent unit kind, or if
-- the time of last update is not available, or not meaningful, for any
-- reason.
--
-- All Unit Kinds are appropriate.
--
---------------------------------------------------------------------------------------
```

-- 11.3 function Compilation_CPU_Duration

```
---------------------------------------------------------------------------------------

    function Compilation_CPU_Duration (Compilation_Unit : in Asis.Compilation_Unit)
                                  return Standard.Duration;

---------------------------------------------------------------------------------------
-- Compilation_Unit  - Specifies the unit to query
--
-- Returns the Central Processing Unit (CPU) duration used to compile the physical
-- compilation unit associated with the Compilation_Unit argument.  The exact
-- significance, or accuracy, of the result is implementation specific.  Returns a
-- duration of 0.0 if the unit has a Nil or nonexistent unit kind, or if
-- the CPU duration for the last compilation is not available for any reason.
-- Returns a duration of 86_400.0 if the CPU duration for the last compilation is
-- greater than 1 day.
--
-- All Unit Kinds are appropriate.
--
```

```
-------------------------------------------------------------------------------
```
-- 11.4 function Attribute_Time
```
-------------------------------------------------------------------------------

    function Attribute_Time
                (Compilation_Unit : in Asis.Compilation_Unit;
                 Attribute        : in Wide_String)
                return Ada.Calendar.Time;

-------------------------------------------------------------------------------
-- Compilation_Unit    - Specifies the unit to query
-- Attribute           - Specifies the name of the attribute to query
--
-- Returns the Time value associated with the given attribute.  Returns
-- Nil_ASIS_Time if the argument is a Nil_Compilation_Unit, the unit does
-- not have the given Attribute, or the implementation does not record times
-- for attributes.
--
-- All Unit Kinds are appropriate.
--
-- Results of this query may vary across ASIS implementations.
--
-------------------------------------------------------------------------------

end Asis.Compilation_Units.Times;
```

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

-- 12 package Asis.Compilation_Units.Relations

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
package Asis.Compilation_Units.Relations is
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- Asis.Compilation_Units.Relations encapsulates semantic relationship
-- concepts used in ASIS.
-------------------------------------------------------------------------------
```

-- 12.1 type Relationship

```
-------------------------------------------------------------------------------
-- Relationship queries provide references to compilation units that are
-- related, in some specific fashion, to one or more given compilation units.
-- Compilation units located by these queries are returned as a set of
-- ordered lists.

    type Relationship (Consistent_Length   : Asis.ASIS_Natural;
                       Inconsistent_Length : Asis.ASIS_Natural;
                       Missing_Length      : Asis.ASIS_Natural;
                       Circular_Length     : Asis.ASIS_Natural) is
        record
            Consistent   : Asis.Compilation_Unit_List (1 .. Consistent_Length);
            Inconsistent : Asis.Compilation_Unit_List (1 .. Inconsistent_Length);
            Missing      : Asis.Compilation_Unit_List (1 .. Missing_Length);
            Circular     : Asis.Compilation_Unit_List (1 .. Circular_Length);
        end record;

-- The following describes the semantics of the unit lists returned by the
-- queries Semantic_Dependence_Order and Elaboration_Order:
--
-- Each query returns a set of four lists.  Every unit returned will have the
-- same Enclosing_Context.  The lists are:
--
-- - Consistent: A list of consistent ordered units.
--
-- - Inconsistent: A list of units that are inconsistent with one or more units on
--   which they semantically depend.
--
-- - Missing: A list of units that have missing (nonexistent) related units.
--
-- - Circular: A list of circular semantic dependencies between units.
--
-- These lists are further described as:
--
-- a) Consistent units list:
--
--    The semantics for the ordering of units in the first list are defined by
--    the individual queries.
--
--    Every unit in this list is unique.  No duplicates are returned; no
--    two units in the list are Is_Equal or Is_Identical.
--
-- b) Inconsistent units list:
--
--    The second list is made up of unit pairs.
--
--    Each pairing defines an inconsistent semantic dependence relationship.
--    The right unit of each pair semantically depends on the preceding left
--    unit. All rightmost units of each pair are always inconsistent, and will
--    not appear in the consistent units list.  The leftmost unit can be either
--    consistent or inconsistent. If a leftmost units is consistent, then it
--    also appears in the consistent units list; otherwise the unit is part of
--    an inconsistent transitive relationship.
--
--    The unit pairs are ordered such that there are no forward semantic
--    dependencies between the inconsistent units.  Each inconsistent unit's
--    supporters always precede it in the list.
--
--    As an example, given four units, A withs B, B withs C, and C withs D;
--    if D is replaced, the inconsistent list contains six units with the
```

```
--    three pairs:
--
--       DC  CB  BA
--
--    The list indicates that units C, B, and A are inconsistent (the rightmost
--    units of each pair).  Semantic dependencies such as B depends on C
--    also are indicated.  The units C, B, and A are in an order that could be
--    submitted to the compiler (a possible recompilation order).
--
--    If a unit is inconsistent because the source for the unit has been
--    edited (or otherwise been made inconsistent by some action of the user
--    or implementation) then the unit references Nil_Compilation_Unit as the
--    cause of the inconsistency (e.g., (Nil A Nil B) is a list of two
--    inconsistent units, neither of which can point to a third unit as the
--    cause for their being inconsistent).
--
--|IP Implementation Permissions
--|IP
--|IP An implementation is allowed to use Nil_Compilation_Unit value for
--|IP the first unit of each pair if it cannot determine the supporting unit
--|IP causing the inconsistent semantic dependence.
--
--    For the above example, the list returned is:
--
--          DC DB DA CB CA BA
--
--    This list reports all dependencies:
--
--          D withed by C withed by B withed by A => DC DB DA
--                      C withed by B withed by A => CB CA
--                                  B withed by A => BA
--
-- c) Missing dependence list:
--
--    The third list is made up of unit pairs.   Each pairing consists of a
--    unit followed by a missing related unit needed by the first unit.
--    A missing unit is a required Compilation_Unit, with a known name, with a
--    Unit_Kind that is either A_Nonexistent_Declaration or A_Nonexistent_Body.
--
--    For example:
--
--    Given a list containing the units:  AB AC
--
--      If Unit_Kind(B) = A_Nonexistent_Declaration and
--         Unit_Kind(C) = A_Nonexistent_Body then
--
--      It can be deduced that:
--         A is missing a needed supporter B (A depends semantically on B).
--         A is missing a needed related unit body C (depending on the kind
--         for A, C can be A’s required body or some subunit of A).
--
--    A unit is reported as missing only if the Post-Compilation Rules of Ada
--    determine it to be needed.  Reference Manual 10.2.
--
-- d) Circular dependence list:
--
--    Circular dependencies between compilation units are provided in the
--    fourth list.  There may be more than one set of circular dependencies.
--    The ordering of distinct sets in the list is implementation-defined.
--    This list will never contain nonexistent units.
--
--    The list is made up of unit pairs.  The second unit of each pair depends
--    semantically on the first unit.  A circularity is established when the
--    first unit of a pair also appears as the second unit of a later pair.
--    (See the unit A in the example below; it is the first unit of the first
--    pair and is the second unit of the third pair).  The next set of circular
--    dependent units, if any, starts with the next unit in the list (the unit
--    D in the example below).
--
--    For example:
--
--    Given a list containing the units:  AC CB BA DG GF FE ED
--
--      It can be determined that there are two sets of circularly dependent units:
```

```
--        {A, B, C} and {D, E, F, G}
--
--     The dependencies are:  A depends on B, B depends on C, C depends on A.
--            D depends on E, E depends on F, F depends on G, G depends on D.
--
--   Each circle of dependence is reported exactly once.  It is not reported
--   once for each unit in the circle.
--
-----------------------------------------------------------------------------------------
```

-- 12.2 constant Nil_Relationship

```
-----------------------------------------------------------------------------------------

    Nil_Relationship : constant Relationship :=
            (Consistent_Length   => 0,
             Inconsistent_Length => 0,
             Missing_Length      => 0,
             Circular_Length     => 0,
             Consistent          => Asis.Nil_Compilation_Unit_List,
             Inconsistent        => Asis.Nil_Compilation_Unit_List,
             Missing             => Asis.Nil_Compilation_Unit_List,
             Circular            => Asis.Nil_Compilation_Unit_List);

-----------------------------------------------------------------------------------------
```

-- 12.3 function Semantic_Dependence_Order

```
-----------------------------------------------------------------------------------------
--     Semantic Dependence Relationships    - Reference Manual 10.1.1(24).
--     Elaboration Dependence Relationships - Reference Manual 10.1.1(25).
--
--|AN Application Note:
--|AN
--|AN To properly determine unit consistency, use one of the two semantic
--|AN dependence queries: Elaboration_Order or Semantic_Dependence_Order.
--|AN These queries return a value of the type Relationship, which contains
--|AN lists of consistent, inconsistent, missing and circular units.
--|AN
--|AN For these two queries, the existence of units in one or more of the
--|AN inconsistent, missing, or circular units list means that the consistent
--|AN unit list may not be complete.
--|AN
--|AN Applications that do not check for inconsistent, missing, or circular
--|AN units before using the consistent list might not operate as expected.
--
-----------------------------------------------------------------------------------------

    function Semantic_Dependence_Order
                (Compilation_Units : in Asis.Compilation_Unit_List;
                 Dependent_Units   : in Asis.Compilation_Unit_List;
                 The_Context       : in Asis.Context;
                 Relation          : in Asis.Relation_Kinds)
                return Relationship;

-----------------------------------------------------------------------------------------
-- Compilation_Units   - Specifies a list of pertinent units
-- Dependent_Units     - Specifies dependents used to limit the query
-- The_Context         - Specifies a program Context for context
-- Relation            - Specifies the relationship to query
--
-- Produces a Relationship value containing compilation_unit elements related to the
-- given Compilation_Units by the specified relation.
--
-- The compilation_unit elements in the consistent units list are ordered such that
-- there are no forward semantic dependencies.
--
-- Dependent_Units are ignored unless the Relation is Descendants or
-- Dependents.  The union of units in the needed units of the Dependent_Units
-- list provide a limiting context for the query.  Only members of these
-- needed units are present in the result.
--
-- If the Dependent_Units list is Is_Nil, the context for the search is the
-- entire Context.  The result of such a query is the full (unlimited)
-- list of Dependents for the Compilation_Units.
--
```

```
-- All units in the result will have an Enclosing_Context value that
-- Is_Identical to The_Context.
--
-- Appropriate Unit_Kinds:
--      A_Procedure
--      A_Function
--      A_Package
--      A_Generic_Procedure
--      A_Generic_Function
--      A_Generic_Package
--      A_Procedure_Instance
--      A_Function_Instance
--      A_Package_Instance
--      A_Procedure_Renaming
--      A_Function_Renaming
--      A_Package_Renaming
--      A_Generic_Procedure_Renaming
--      A_Generic_Function_Renaming
--      A_Generic_Package_Renaming
--      A_Procedure_Body
--      A_Function_Body
--      A_Package_Body
--      A_Procedure_Body_Subunit
--      A_Function_Body_Subunit
--      A_Package_Body_Subunit
--      A_Task_Body_Subunit
--      A_Protected_Body_Subunit
--      An_Unknown_Unit            -- See Implementation Permissions
--
-- The Semantic_Dependence_Order query should never raise an exception
-- when processing inconsistent unit (sub)sets.  This query is the only
-- means for an application to know if a given unit is consistent with
-- (some of) its supporters (dependents), and therefore the related
-- semantic processing can give valuable results for this unit.
--
--|IP Implementation Permissions:
--|IP
--|IP The handling of An_Unknown_Unit is implementation specific.  It can be
--|IP possible to obtain Semantic Dependence Relationships when starting
--|IP with a list containing one or more units that are An_Unknown_Unit.
--|IP However, results may vary across ASIS implementations.
--
--|AN Application Note:
--|AN
--|AN Semantic_Dependence_Order defines consistent units to be ordered such
--|AN that there are no forward semantic dependencies.
--
---------------------------------------------------------------------------------------
```

-- 12.4 function Elaboration_Order

```
---------------------------------------------------------------------------------------

    function Elaboration_Order
                (Compilation_Units : in Asis.Compilation_Unit_List;
                 The_Context       : in Asis.Context)
                return Relationship;

---------------------------------------------------------------------------------------
-- Compilation_Units   - Specifies a list of units to elaborate
-- The_Context         - Specifies a program Context for context
--
-- Produces, in elaboration order, a Relationship value containing compilation
-- units required to elaborate the given compilation units.
--
-- The return value contains the set of ordered lists described above for the
-- queries on Semantic Dependence Relationships.  If the inconsistent,
-- missing, and circular lists are empty, the consistent list will contain
-- all units required to elaborate the arguments.
--
--|IP Implementation Permissions:
--|IP
--|IP The Relationship value may include any number of implementation-specific
--|IP runtime support packages.
--
```

```
-- The first unit in the Consistent units list will always be the
-- specification for package Standard.  The list will contain all units
-- required to elaborate the arguments.
--
-- Use the Context_Clause_Elements query to get pragma Elaborate elements
-- for a compilation unit.
--
-- Appropriate Unit_Kinds:
--      A_Procedure
--      A_Function
--      A_Package
--      A_Generic_Procedure
--      A_Generic_Function
--      A_Generic_Package
--      A_Procedure_Instance
--      A_Function_Instance
--      A_Package_Instance
--      A_Procedure_Renaming
--      A_Function_Renaming
--      A_Package_Renaming
--      A_Generic_Procedure_Renaming
--      A_Generic_Function_Renaming
--      A_Generic_Package_Renaming
--      A_Procedure_Body
--      A_Function_Body
--      A_Package_Body
--      A_Procedure_Body_Subunit
--      A_Function_Body_Subunit
--      A_Package_Body_Subunit
--      A_Task_Body_Subunit
--      A_Protected_Body_Subunit
--      An_Unknown_Unit            -- See Implementation Permissions
--
--|IP Implementation Permissions:
--|IP
--|IP The handling of An_Unknown_Unit is implementation specific.  It can
--|IP be possible to obtain Semantic Dependence Relationships when starting
--|IP with a list containing one or more units that are An_Unknown_Unit.
--|IP However, results may vary across ASIS implementations.
--
-------------------------------------------------------------------------------------

end Asis.Compilation_Units.Relations;
```

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

-- 13 package Asis.Elements

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
package Asis.Elements is
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- Asis.Elements encapsulates a set of queries that operate on all elements
-- and some queries specific to A_Pragma elements.
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

-- 13.1 function Unit_Declaration

```
-------------------------------------------------------------------------------
-- Gateway queries between Compilation_Units and Elements.
-------------------------------------------------------------------------------

    function Unit_Declaration (Compilation_Unit : in Asis.Compilation_Unit)
                              return Asis.Declaration;

-------------------------------------------------------------------------------
-- Compilation_Unit    - Specifies the unit to query
--
-- Returns the element representing the declaration of the compilation_unit.
--
-- Returns a Nil_Element if the unit is A_Nonexistent_Declaration,
-- A_Nonexistent_Body, A_Configuration_Compilation, or An_Unknown_Unit.
--
-- All Unit_Kinds are appropriate except Not_A_Unit.
--
-- Returns Declaration_Kinds:
--      Not_A_Declaration
--      A_Function_Body_Declaration
--      A_Function_Declaration
--      A_Function_Instantiation
--      A_Generic_Function_Declaration
--      A_Generic_Package_Declaration
--      A_Generic_Procedure_Declaration
--      A_Package_Body_Declaration
--      A_Package_Declaration
--      A_Package_Instantiation
--      A_Procedure_Body_Declaration
--      A_Procedure_Declaration
--      A_Procedure_Instantiation
--      A_Task_Body_Declaration
--      A_Package_Renaming_Declaration
--      A_Procedure_Renaming_Declaration
--      A_Function_Renaming_Declaration
--      A_Generic_Package_Renaming_Declaration
--      A_Generic_Procedure_Renaming_Declaration
--      A_Generic_Function_Renaming_Declaration
--      A_Protected_Body_Declaration
--
-------------------------------------------------------------------------------
```

-- 13.2 function Enclosing_Compilation_Unit

```
-------------------------------------------------------------------------------

    function Enclosing_Compilation_Unit (Element : in Asis.Element)
                                        return Asis.Compilation_Unit;

-------------------------------------------------------------------------------
-- Element - Specifies an Element whose Compilation_Unit is desired
--
-- Returns the Compilation_Unit that contains the given Element.
--
-- Raises ASIS_Inappropriate_Element if the Element is a Nil_Element.
--
```

```
-------------------------------------------------------------------------------
```
-- 13.3 function Context_Clause_Elements
```
-------------------------------------------------------------------------------

    function Context_Clause_Elements
                (Compilation_Unit : in Asis.Compilation_Unit;
                 Include_Pragmas  : in Boolean := False)
                return Asis.Context_Clause_List;

-------------------------------------------------------------------------------
-- Compilation_Unit - Specifies the unit to query
-- Include_Pragmas  - Specifies whether pragmas are to be returned
--
-- Returns a list of with clauses, use clauses, and pragmas that explicitly
-- appear in the context clause of the compilation unit, in their order of
-- appearance.
--
-- Returns a Nil_Element_List if the unit has A_Nonexistent_Declaration,
-- A_Nonexistent_Body, or An_Unknown_Unit Unit_Kind.
--
--|IR Implementation Requirement:
--|IR
--|IR All pragma Elaborate elements for this unit will appear in this list.  Other
--|IR pragmas will appear in this list, or in the Compilation_Pragmas list, or
--|IR both.
--
--|IP Implementation Permissions:
--|IP
--|IP Implementors are encouraged to use this list to return all pragmas whose
--|IP full effect is determined by their exact textual position.  Pragmas that
--|IP do not have placement dependencies may be returned in either list.  Only
--|IP pragmas that appear in the unit’s context clause are returned
--|IP by this query.  All other pragmas, affecting the compilation of this
--|IP unit, are available from the Compilation_Pragmas query.
--
-- Ada predefined packages, such as package Standard, may or may not have
-- context-clause elements available for processing by applications.  The
-- physical existence of a package Standard is implementation specific.
-- The same is true for other Ada predefined packages, such as Ada.Text_Io and
-- Ada.Direct_Io.  The Origin query can be used to determine whether or not
-- a particular unit is an Ada Predefined unit.
--
--|IP Implementation Permissions:
--|IP
--|IP Results of this query may vary across ASIS implementations.  Some
--|IP implementations normalize all multi-name with clauses and use clauses
--|IP into an equivalent sequence of single-name with clause and use clauses.
--|IP Similarly, an implementation may retain only a single reference to a name
--|IP that appeared more than once in the original context clause.
--|IP Some implementors will return only pragma
--|IP Elaborate elements in this list and return all other pragmas via the
--|IP Compilation_Pragmas query.
--
-- All Unit_Kinds are appropriate except Not_A_Unit.
--
-- Returns Element_Kinds:
--      A_Pragma
--      A_Clause
--
-- Returns Clause_Kinds:
--      A_With_Clause
--      A_Use_Package_Clause
--
-------------------------------------------------------------------------------
```
-- 13.4 function Configuration_Pragmas
```
-------------------------------------------------------------------------------

   function Configuration_Pragmas (The_Context : in Asis.Context)
                                 return Asis.Pragma_Element_List;

-------------------------------------------------------------------------------
-- The_Context    - Specifies the Context to query
--
```

```
-- Returns a list of pragmas that apply to all future compilation_unit elements
-- compiled into The_Context.  Pragmas returned by this query should
-- have appeared in a compilation that had no compilation_unit elements.
-- To the extent that order is meaningful, the pragmas should be in
-- their order of appearance in the compilation.  (The order is implementation
-- dependent, many pragmas have the same effect regardless of order.)
--
-- Returns a Nil_Element_List if there are no such configuration pragmas.
--
-- Returns Element_Kinds:
--      A_Pragma
--
---------------------------------------------------------------------------------------
```

-- 13.5 function Compilation_Pragmas

```
---------------------------------------------------------------------------------------

    function Compilation_Pragmas (Compilation_Unit : in Asis.Compilation_Unit)
                                 return Asis.Pragma_Element_List;

---------------------------------------------------------------------------------------
-- Compilation_Unit    - Specifies the unit to query
--
-- Returns a list of pragmas that apply to the compilation of the unit.
-- To the extent that order is meaningful, the pragmas should be in
-- their order of appearance in the compilation.  (The order is implementation
-- dependent, many pragmas have the same effect regardless of order.)
--
-- There are two sources for the pragmas that appear in this list:
--
-- - Program unit pragmas appearing at the place of a compilation_unit.
--   See Reference Manual 10.1.5(4).
--
-- - Configuration pragmas appearing before the first
--   compilation_unit of a compilation.  See Reference Manual 10.1.5(8).
--
-- This query does not return Elaborate pragmas from the unit context
-- clause of the compilation unit; they do not apply to the compilation,
-- only to the unit.
--
-- Use the Context_Clause_Elements query to obtain a list of all pragmas
-- (including Elaborate pragmas) from the context clause of a compilation unit.
--
-- Pragmas from this query may be duplicates of some or all of the
-- non-Elaborate pragmas available from the Context_Clause_Elements query.
-- Such duplication is simply the result of the textual position of the
-- pragma--globally effective pragmas may appear textually within the context
-- clause of a particular unit, and be returned as part of the Context_Clause
-- for that unit.
--
-- Ada predefined packages, such as package Standard, may or may not have
-- pragmas available for processing by applications.  The physical
-- existence of a package Standard is implementation specific.  The same
-- is true for other Ada predefined packages, such as Ada.Text_Io and
-- Ada.Direct_Io. The Origin query can be used to determine whether or
-- not a particular unit is an Ada Predefined unit.
--
-- Returns a Nil_Element_List if the compilation unit:
--
-- - has no such applicable pragmas.
--
-- - is an An_Unknown_Unit, A_Nonexistent_Declaration, or A_Nonexistent_Body.
--
-- All Unit_Kinds are appropriate except Not_A_Unit.
--
-- Returns Element_Kinds:
--      A_Pragma
--
```

```
------------------------------------------------------------------------------------------
-- Element_Kinds Hierarchy
--
--       Element_Kinds Value      Subordinate Kinds
------------------------------------------------------------------------------------------
--
--   Key: Read "->" as "can be further categorized by its"
--
--       A_Pragma              -> Pragma_Kinds
--
--       A_Defining_Name       -> Defining_Name_Kinds
--                                        -> Operator_Kinds
--
--       A_Declaration         -> Declaration_Kinds
--                                        -> Declaration_Origin
--                                        -> Mode_Kinds
--                                        -> Subprogram_Default_Kinds
--
--       A_Definition          -> Definition_Kinds
--                                        -> Trait_Kinds
--                                        -> Type_Kinds
--                                        -> Formal_Type_Kinds
--                                        -> Access_Type_Kinds
--                                        -> Root_Type_Kinds
--                                        -> Constraint_Kinds
--                                        -> Discrete_Range_Kinds
--
--       An_Expression         -> Expression_Kinds
--                                        -> Operator_Kinds
--                                        -> Attribute_Kinds
--
--       An_Association        -> Association_Kinds
--
--       A_Statement           -> Statement_Kinds
--
--       A_Path                -> Path_Kinds
--
--       A_Clause              -> Clause_Kinds
--                                        -> Representation_Clause_Kinds
--
--       An_Exception_Handler
--
------------------------------------------------------------------------------------------
```

-- 13.6 function Element_Kind

```
------------------------------------------------------------------------------------------

    function Element_Kind (Element : in Asis.Element)
                           return Asis.Element_Kinds;

------------------------------------------------------------------------------------------
-- Element - Specifies the element to query
--
-- Returns the Element_Kinds value of Element.
-- Returns Not_An_Element for a Nil_Element.
--
-- All element kinds are expected.
--
------------------------------------------------------------------------------------------
```

-- 13.7 function Pragma_Kind

```
------------------------------------------------------------------------------------------
    function Pragma_Kind (Pragma_Element : in Asis.Pragma_Element)
                         return Asis.Pragma_Kinds;

------------------------------------------------------------------------------------------
-- Pragma_Element  - Specifies the element to query
--
-- Returns the Pragma_Kinds value of Pragma_Element.
-- Returns Not_A_Pragma for any unexpected element such as a
-- Nil_Element, A_Statement, or A_Declaration.
--
-- Expected Element_Kinds:
--      A_Pragma
--
```

```
-------------------------------------------------------------------------------
```
-- 13.8 function Defining_Name_Kind
```
-------------------------------------------------------------------------------

    function Defining_Name_Kind (Defining_Name : in Asis.Defining_Name)
                                return Asis.Defining_Name_Kinds;

-------------------------------------------------------------------------------
-- Defining_Name   - Specifies the element to query
--
-- Returns the Defining_Name_Kinds value of the Defining_Name.
--
-- Returns Not_A_Defining_Name for any unexpected element such as a
-- Nil_Element, A_Clause, or A_Statement.
--
-- Expected Element_Kinds:
--      A_Defining_Name
--
-------------------------------------------------------------------------------
```
-- 13.9 function Declaration_Kind
```
-------------------------------------------------------------------------------

    function Declaration_Kind (Declaration : in Asis.Declaration)
                              return Asis.Declaration_Kinds;

-------------------------------------------------------------------------------
-- Declaration   - Specifies the element to query
--
-- Returns the Declaration_Kinds value of the Declaration.
--
-- Returns Not_A_Declaration for any unexpected element such as a
-- Nil_Element, A_Definition, or A_Statement.
--
-- Expected Element_Kinds:
--      A_Declaration
--
-------------------------------------------------------------------------------
```
-- 13.10 function Trait_Kind
```
-------------------------------------------------------------------------------

    function Trait_Kind (Element : in Asis.Element)
                        return Asis.Trait_Kinds;

-------------------------------------------------------------------------------
-- Element   - Specifies the Element to query
--
-- Returns the Trait_Kinds value of the Element.
--
-- Returns Not_A_Trait for any unexpected element such as a
-- Nil_Element, A_Statement, or An_Expression.
--
-- Expected Declaration_Kinds:
--     A_Private_Type_Declaration
--     A_Private_Extension_Declaration
--     A_Variable_Declaration
--     A_Constant_Declaration
--     A_Deferred_Constant_Declaration
--     A_Discriminant_Specification
--     A_Loop_Parameter_Specification
--     A_Procedure_Declaration
--     A_Function_Declaration
--     A_Parameter_Specification
--
-- Expected Definition_Kinds:
--     A_Component_Definition
--     A_Private_Type_Definition
--     A_Tagged_Private_Type_Definition
--     A_Private_Extension_Definition
--
```

```
-- Expected Type_Kinds:
--      A_Derived_Type_Definition
--      A_Derived_Record_Extension_Definition
--      A_Record_Type_Definition
--      A_Tagged_Record_Type_Definition
--
-- Expected Formal_Type_Kinds:
--      A_Formal_Private_Type_Definition
--      A_Formal_Tagged_Private_Type_Definition
--      A_Formal_Derived_Type_Definition
--
---------------------------------------------------------------------------------------
```

-- 13.11 function Declaration_Origin

```
---------------------------------------------------------------------------------------

    function Declaration_Origin (Declaration : in Asis.Declaration)
                                 return Asis.Declaration_Origins;

---------------------------------------------------------------------------------------
-- Declaration   - Specifies the Declaration to query
--
-- Returns the Declaration_Origins value of the Declaration.
--
-- Returns Not_A_Declaration_Origin for any unexpected element such as a
-- Nil_Element, A_Definition, or A_Clause.
--
-- Expected Element_Kinds:
--       A_Declaration
--
---------------------------------------------------------------------------------------
```

-- 13.12 function Mode_Kind

```
---------------------------------------------------------------------------------------

    function Mode_Kind (Declaration : in Asis.Declaration)
                                  return Asis.Mode_Kinds;

---------------------------------------------------------------------------------------
-- Declaration   - Specifies the element to query
--
-- Returns the Mode_Kinds value of the Declaration.
--
-- Returns A_Default_In_Mode for an access parameter.
--
-- Returns Not_A_Mode for any unexpected element such as a
-- Nil_Element, A_Definition, or A_Statement.
--
-- Expected Declaration_Kinds:
--       A_Parameter_Specification
--       A_Formal_Object_Declaration
--
---------------------------------------------------------------------------------------
```

-- 13.13 function Default_Kind

```
---------------------------------------------------------------------------------------

    function Default_Kind (Declaration : in Asis.Generic_Formal_Parameter)
                          return Asis.Subprogram_Default_Kinds;

---------------------------------------------------------------------------------------
-- Declaration   - Specifies the element to query
--
-- Returns the Subprogram_Default_Kinds value of the Declaration.
--
-- Returns Not_A_Declaration for any unexpected element such as a
-- Nil_Element, A_Definition, or A_Statement.
--
-- Expected Declaration_Kinds:
--       A_Formal_Function_Declaration
--       A_Formal_Procedure_Declaration
--
```

```
------------------------------------------------------------------------------
```
-- 13.14 function Definition_Kind
```
------------------------------------------------------------------------------

    function Definition_Kind (Definition : in Asis.Definition)
                             return Asis.Definition_Kinds;

------------------------------------------------------------------------------
-- Definition   - Specifies the Definition to query
--
-- Returns the Definition_Kinds value of the Definition.
--
-- Returns Not_A_Definition for any unexpected element such as a
-- Nil_Element, A_Statement, or A_Declaration.
--
-- Expected Element_Kinds:
--      A_Definition
--
------------------------------------------------------------------------------
```
-- 13.15 function Type_Kind
```
------------------------------------------------------------------------------

    function Type_Kind (Definition : in Asis.Type_Definition)
                                  return Asis.Type_Kinds;

------------------------------------------------------------------------------
-- Definition   - Specifies the Type_Definition to query
--
-- Returns the Type_Kinds value of the Definition.
--
-- Returns Not_A_Type_Definition for any unexpected element such as a
-- Nil_Element, A_Statement, or A_Declaration.
--
-- Expected Definition_Kinds:
--      A_Type_Definition
--
------------------------------------------------------------------------------
```
-- 13.16 function Formal_Type_Kind
```
------------------------------------------------------------------------------

    function Formal_Type_Kind
                 (Definition : in Asis.Formal_Type_Definition)
                  return Asis.Formal_Type_Kinds;

------------------------------------------------------------------------------
-- Definition   - Specifies the Formal_Type_Definition to query
--
-- Returns the Formal_Type_Kinds value of the Definition.
--
-- Returns Not_A_Formal_Type_Definition for any unexpected element such as a
-- Nil_Element, A_Statement, or A_Declaration.
--
-- Expected Definition_Kinds:
--      A_Formal_Type_Definition
--
------------------------------------------------------------------------------
```
-- 13.17 function Access_Type_Kind
```
------------------------------------------------------------------------------

    function Access_Type_Kind
                 (Definition : in Asis.Access_Type_Definition)
                  return Asis.Access_Type_Kinds;

------------------------------------------------------------------------------
-- Definition   - Specifies the Access_Type_Definition to query
--
-- Returns the Access_Type_Kinds value of the Definition.
--
-- Returns Not_An_Access_Type_Definition for any unexpected element such as a
-- Nil_Element, A_Statement, or A_Declaration.
--
```

```
-- Expected Type_Kinds:
--      An_Access_Type_Definition
--
-----------------------------------------------------------------------------------
```

-- 13.18 function Root_Type_Kind

```
-----------------------------------------------------------------------------------

    function Root_Type_Kind
                (Definition : in Asis.Root_Type_Definition)
                 return Asis.Root_Type_Kinds;

-----------------------------------------------------------------------------------
-- Definition   - Specifies the Root_Type_Definition to query
--
-- Returns the Root_Type_Kinds value of the Definition.
--
-- Returns Not_A_Root_Type_Definition for any unexpected element such as a
-- Nil_Element, A_Statement, or A_Declaration.
--
-- Expected Type_Kinds:
--      A_Root_Type_Definition
--
-----------------------------------------------------------------------------------
```

-- 13.19 function Constraint_Kind

```
-----------------------------------------------------------------------------------

    function Constraint_Kind
                (Definition : in Asis.Constraint)
                 return Asis.Constraint_Kinds;

-----------------------------------------------------------------------------------
-- Definition   - Specifies the constraint to query
--
-- Returns the Constraint_Kinds value of the Definition.
--
-- Returns Not_A_Constraint for any unexpected element such as a
-- Nil_Element, A_Statement, or A_Declaration.
--
-- Expected Definition_Kinds:
--      A_Constraint
--
-----------------------------------------------------------------------------------
```

-- 13.20 function Discrete_Range_Kind

```
-----------------------------------------------------------------------------------

    function Discrete_Range_Kind
                (Definition : in Asis.Discrete_Range)
                 return Asis.Discrete_Range_Kinds;

-----------------------------------------------------------------------------------
-- Definition   - Specifies the discrete_range to query
--
-- Returns the Discrete_Range_Kinds value of the Definition.
--
-- Returns Not_A_Discrete_Range for any unexpected element such as a
-- Nil_Element, A_Statement, or A_Declaration.
--
-- Expected Definition_Kinds:
--      A_Discrete_Subtype_Definition
--      A_Discrete_Range
--
```

```
-------------------------------------------------------------------------------
```
-- 13.21 function Expression_Kind
```
-------------------------------------------------------------------------------

    function Expression_Kind (Expression : in Asis.Expression)
                             return Asis.Expression_Kinds;

-------------------------------------------------------------------------------
-- Expression   - Specifies the Expression to query
--
-- Returns the Expression_Kinds value of the Expression.
--
-- Returns Not_An_Expression for any unexpected element such as a
-- Nil_Element, A_Statement, or A_Declaration.
--
-- Expected Element_Kinds:
--      An_Expression
--
-------------------------------------------------------------------------------
```
-- 13.22 function Operator_Kind
```
-------------------------------------------------------------------------------

    function Operator_Kind (Element : in Asis.Element)
                           return Asis.Operator_Kinds;

-------------------------------------------------------------------------------
-- Element   - Specifies the Element to query
--
-- Returns the Operator_Kinds value of the A_Defining_Name or An_Expression
-- element.
--
-- Returns Not_An_Operator for any unexpected element such as a
-- Nil_Element, A_Statement, or A_Declaration.
--
-- Expected Defining_Name_Kinds:
--      A_Defining_Operator_Symbol
--
-- Expected Expression_Kinds:
--      An_Operator_Symbol
--
-------------------------------------------------------------------------------
```
-- 13.23 function Attribute_Kind
```
-------------------------------------------------------------------------------

    function Attribute_Kind (Expression : in Asis.Expression)
                            return Asis.Attribute_Kinds;

-------------------------------------------------------------------------------
-- Expression   - Specifies the Expression to query
--
-- Returns the Attribute_Kinds value of the Expression.
--
-- Returns Not_An_Attribute for any unexpected element such as a
-- Nil_Element, A_Statement, or A_Declaration.
--
-- Expected Expression_Kinds:
--      An_Attribute_Reference
--
-------------------------------------------------------------------------------
```
-- 13.24 function Association_Kind
```
-------------------------------------------------------------------------------

    function Association_Kind (Association : in Asis.Association)
                              return Asis.Association_Kinds;

-------------------------------------------------------------------------------
-- Association   - Specifies the Association to query
--
-- Returns the Association_Kinds value of the Association.
--
```

```
-- Returns Not_An_Association for any unexpected element such as a
-- Nil_Element, A_Statement, or A_Declaration.
--
-- Expected Element_Kinds:
--      An_Association
--
-----------------------------------------------------------------------------
```

-- 13.25 function Statement_Kind

```
-----------------------------------------------------------------------------

    function Statement_Kind (Statement : in Asis.Statement)
                            return Asis.Statement_Kinds;

-----------------------------------------------------------------------------
-- Statement   - Specifies the element to query
--
-- Returns the Statement_Kinds value of the statement.
--
-- Returns Not_A_Statement for any unexpected element such as a
-- Nil_Element, A_Definition, or A_Declaration.
--
-- Expected Element_Kinds:
--      A_Statement
--
-----------------------------------------------------------------------------
```

-- 13.26 function Path_Kind

```
-----------------------------------------------------------------------------

    function Path_Kind (Path : in Asis.Path) return Asis.Path_Kinds;

-----------------------------------------------------------------------------
-- Path   - Specifies the Path to query
--
-- Returns the Path_Kinds value of the Path.
--
-- Returns Not_A_Path for any unexpected element such as a
-- Nil_Element, A_Statement, or A_Declaration.
--
-- Expected Element_Kinds:
--      A_Path
--
-----------------------------------------------------------------------------
```

-- 13.27 function Clause_Kind

```
-----------------------------------------------------------------------------

    function Clause_Kind (Clause : in Asis.Clause)
                         return Asis.Clause_Kinds;

-----------------------------------------------------------------------------
-- Clause   - Specifies the element to query
--
-- Returns the Clause_Kinds value of the Clause.
--
-- Returns Not_A_Clause for any unexpected element such as a
-- Nil_Element, A_Definition, or A_Declaration.
--
-- Expected Element_Kinds:
--      A_Clause
--
-----------------------------------------------------------------------------
```

-- 13.28 function Representation_Clause_Kind

```
-----------------------------------------------------------------------------

    function Representation_Clause_Kind
                (Clause : in Asis.Representation_Clause)
                 return Asis.Representation_Clause_Kinds;

-----------------------------------------------------------------------------
-- Clause   - Specifies the element to query
--
-- Returns the Representation_Clause_Kinds value of the Clause.
```

```
--
-- Returns Not_A_Representation_Clause for any unexpected element such as a
-- Nil_Element, A_Definition, or A_Declaration.
--
-- Expected Clause_Kinds:
--      A_Representation_Clause
--
-------------------------------------------------------------------------------
```

-- 13.29 function Is_Nil

```
-------------------------------------------------------------------------------

    function Is_Nil (Right : in Asis.Element) return Boolean;

-------------------------------------------------------------------------------
-- Right   - Specifies the element to check
--
-- Returns True if the program element is the Nil_Element.
--
-------------------------------------------------------------------------------
```

-- 13.30 function Is_Nil

```
-------------------------------------------------------------------------------

    function Is_Nil (Right : in Asis.Element_List) return Boolean;

-------------------------------------------------------------------------------
-- Right   - Specifies the element list to check
--
-- Returns True if the element list has a length of zero.
--
-------------------------------------------------------------------------------
```

-- 13.31 function Is_Equal

```
-------------------------------------------------------------------------------

    function Is_Equal (Left  : in Asis.Element;
                       Right : in Asis.Element) return Boolean;

-------------------------------------------------------------------------------
-- Left    - Specifies the left element to compare
-- Right   - Specifies the right element to compare
--
-- Returns True if Left and Right represent the same physical element,
-- from the same physical compilation unit.  The two elements may or
-- may not be from the same open ASIS Context variable.
--
-- Implies:
--
--      Is_Equal (Enclosing_Compilation_Unit (Left),
--                Enclosing_Compilation_Unit (Right)) = True
--
-------------------------------------------------------------------------------
```

-- 13.32 function Is_Identical

```
-------------------------------------------------------------------------------

    function Is_Identical (Left  : in Asis.Element;
                           Right : in Asis.Element) return Boolean;

-------------------------------------------------------------------------------
-- Left    - Specifies the left element
-- Right   - Specifies the right element
--
-- Returns True if Left and Right represent the same physical element,
-- from the same physical compilation unit, from the same open ASIS
-- Context variable.
--
-- Implies:
--
--      Is_Identical (Enclosing_Compilation_Unit (Left),
--                    Enclosing_Compilation_Unit (Right)) = True
--
```

```
---------------------------------------------------------------------------------------
```

-- 13.33 function Is_Part_Of_Implicit

```
---------------------------------------------------------------------------------------

    function Is_Part_Of_Implicit (Element : in Asis.Element) return Boolean;

---------------------------------------------------------------------------------------
-- Element - Specifies the element to query
--
-- Returns True for any Element that is, or that forms part of, any
-- implicitly declared or specified program Element structure.
--
-- Returns True for any implicit generic child unit specifications or their
-- subcomponents. Reference Manual 10.1.1(19).
--
-- Returns False for a Nil_Element, or any Element that correspond to text
-- which was specified explicitly (typed, entered, written).
--
-- Generic instance specifications and bodies, while implicit, are treated
-- as a special case.  These elements will not normally test as
-- Is_Part_Of_Implicit.  Rather, they are Is_Part_Of_Instance.  They only test
-- as Is_Part_Of_Implicit if one of the following rules applies.  This is
-- done so that it is possible to determine whether a declaration, which
-- happens to occur within an instance, is an implicit result of
-- another declaration which occurs explicitly within the generic template.
--
-- Implicit Elements are those that represent these portions of the Ada
-- language:
--
-- - Reference Manual 4.5.(9)
--
--   o All predefined operator declarations and their component elements are
--     Is_Part_Of_Implicit
--
-- - Reference Manual 3.4(16)
--
--   o Implicit predefined operators of the derived type.
--
-- - Reference Manual 3.4(17-22)
--
--   o Implicit inherited subprogram declarations and their component elements are
--     Is_Part_Of_Implicit
--
-- - Reference Manual 6.4(9) and 12.3(7)
--
--   o Implicit actual parameter expressions (defaults).
--
--   o The A_Parameter_Association that includes a defaulted parameter value Is_Normalized
--     and also Is_Part_Of_Implicit.  The Formal_Parameter and the Actual_Parameter values
--     from such Associations are not Is_Part_Of_Implicit unless they are from default
--     initializations for an inherited subprogram declaration and have an
--     Enclosing_Element that is the parameter specification of the subprogram
--     declaration. (Those elements are shared with (were created by) the original
--     subprogram declaration, or they are naming expressions representing the actual
--     generic subprogram selected at the place of an instantiation for A_Box_Default.)
--
--   o All A_Parameter_Association Kinds from a Normalized list are Is_Part_Of_Implicit.
--
-- - Reference Manual 6.6 (6)
--
--   o Inequality operator declarations for limited private types are Is_Part_Of_Implicit.
--
--   o Depending on the ASIS implementation, a "/=" appearing in the compilation may
--     result in a "NOT" and an "=" in the internal representation.  These two elements
--     test as Is_Part_Of_Implicit because they do not represent text from the original
--     compilation text.
--
-- - Reference Manual 12.3 (16)
--
--   o implicit generic instance specifications and bodies are not Is_Part_Of_Implicit;
--     they are Is_Part_Of_Instance and are only implicit if some other rule makes them
--     so.
--
```

```
-------------------------------------------------------------------------------
```

-- 13.34 function Is_Part_Of_Inherited

```
-------------------------------------------------------------------------------

    function Is_Part_Of_Inherited (Element : in Asis.Element) return Boolean;

-------------------------------------------------------------------------------
-- Element - Specifies the element to query
--
-- Returns True for any Element that is, or that forms part of, an
-- inherited primitive subprogram declaration.
--
-- Returns False for any other Element including a Nil_Element.
--
-------------------------------------------------------------------------------
```

-- 13.35 function Is_Part_Of_Instance

```
-------------------------------------------------------------------------------

    function Is_Part_Of_Instance (Element : in Asis.Element) return Boolean;

-------------------------------------------------------------------------------
-- Element - Specifies the element to test
--
-- Returns True if the Element is part of an implicit generic specification
-- instance or an implicit generic body instance.
--
-- Returns False for explicit, inherited, and predefined Elements that are
-- not the result of a generic expansion.
--
-- Returns False for any implicit generic child unit specifications or
-- their subcomponents. Reference Manual 10.1.1(19).
--
-- Returns False for a Nil_Element.
--
-- Instantiations are not themselves Is_Part_Of_Instance unless they are
-- encountered while traversing a generic instance.
--
-------------------------------------------------------------------------------
```

-- 13.36 function Enclosing_Element

```
-------------------------------------------------------------------------------

    function Enclosing_Element (Element : in Asis.Element) return Asis.Element;

    function Enclosing_Element (Element                    : in Asis.Element;
                                Expected_Enclosing_Element : in Asis.Element)
                            return Asis.Element;

-------------------------------------------------------------------------------
-- Element - Specifies the element to query
-- Expected_Enclosing_Element - Specifies an enclosing element expected to
-- contain the element
--
-- Returns the Element that immediately encloses the given element.  This
-- query is intended to exactly reverse any single parent-to-child element
-- traversal.  For any structural query that returns a subcomponent of an
-- element (or that returns a list of subcomponent elements), the original element
-- can be determined by passing the subcomponent element to this query.
--
--|AN Application Note:
--|AN
--|AN Semantic queries (queries that test the meaning of a program rather
--|AN than its structure) return Elements that usually do not have the original
--|AN argument Element as their parent.
--
```

```
-- Returns a Nil_Element if:
--
-- - the element is the declaration part of a compilation unit
--   (Unit_Declaration).
--
-- - the element is with clause or use clause of a context clause
--   (Context_Clause_Elements).
--
-- - the element is a pragma for a compilation unit
--   (Compilation_Pragmas and Context_Clause_Elements).
--
-- Use Enclosing_Compilation_Unit to get the enclosing compilation unit for
-- any element value other than Nil_Element.
--
-- Raises ASIS_Inappropriate_Element if the Element is a Nil_Element.
--
-- Examples:
--
-- - Given a A_Declaration/A_Full_Type_Declaration in the declarative region
--   of a block statement, returns the A_Statement/A_Block_Statement Element
--   that encloses the type declaration.
--
-- - Given A_Statement, from the sequence of statements within a loop
--   statement, returns the enclosing A_Statement/A_Loop_Statement.
--
-- - Given the An_Expression/An_Identifier selector from an expanded name,
--   returns the An_Expression/A_Selected_Component that represents the
--   combination of the prefix, the dot, and the selector.
--
-- - Given the A_Declaration corresponding to the implicit redeclaration of
--   a child generic for an instantiated parent generic, returns the expanded
--   generic specific template from the parent generic instantiation
--   corresponding to any implicit generic child unit specification given as
--   an argument. Reference Manual 10.1.1(19).
--
--|AN Application Note:
--|AN
--|AN The optional Expected_Enclosing_Element parameter is used only to optimize
--|AN this query.  This speed up is only present for ASIS implementations
--|AN where the underlying implementor's environment does not have "parent
--|AN pointers". For these implementations, this query is implemented as a
--|AN "search". The Enclosing_Compilation_Unit is searched for the argument
--|AN Element. The Expected_Enclosing_Element parameter provides a means of
--|AN shortening the search.
--|AN Note: If the argument Element is not a sub-element of the
--|AN Expected_Enclosing_Element parameter, or if the
--|AN Expected_Enclosing_Element is a Nil_Element, the result of the
--|AN call is a Nil_Element.
--|AN
--|AN Implementations that do not require the Expected_Enclosing_Element
--|AN parameter may ignore it.  They are encouraged, but not required, to test
--|AN the Expected_Enclosing_Element parameter and to determine if it is an
--|AN invalid Element value (its associated Environment Context may be closed)
--|AN
--|AN Portable applications should not use the Expected_Enclosing_Element
--|AN parameter since it can lead to unexpected differences when porting an
--|AN application between ASIS implementations where one implementation uses
--|AN the parameter and the other implementation does not.  Passing a "wrong"
--|AN Expected_Enclosing_Element to an implementation that ignores it, is
--|AN harmless. Passing a "wrong" Expected_Enclosing_Element to an
--|AN implementation that may utilize it, can lead to an unexpected
--|AN Nil_Element result.
--
```

```
-------------------------------------------------------------------------------
```
-- 13.37 function Pragmas
```
-------------------------------------------------------------------------------

    function Pragmas (The_Element : in Asis.Element)
                     return Asis.Pragma_Element_List;

-------------------------------------------------------------------------------
-- The_Element - Specifies the element to query
--
-- Returns the list of pragmas, in their order of appearance, that appear
-- directly within the given The_Element.  Returns only those pragmas that are
-- immediate component elements of the given The_Element.  Pragmas embedded
-- within other component elements are not returned.  For example, returns
-- the pragmas in a package specification, in the statement list of a loop,
-- or in a record component list.
--
-- This query returns exactly those pragmas that are returned by the
-- various queries, that accept these same argument kinds, and that
-- return Declaration_List and Statement_List, where the inclusion of
-- Pragmas is controlled by an Include_Pragmas parameter.
--
-- Returns a Nil_Element_List if there are no pragmas.
--
-- Appropriate Element_Kinds:
--       A_Path                         (pragmas from the statement list +
--                                       pragmas immediately preceding the
--                                       reserved word "when" of the first
--                                       alternative)
--       An_Exception_Handler           (pragmas from the statement list + pragmas
--                                       immediately preceding the reserved word
--                                       "when" of the first exception handler)
--
-- Appropriate Declaration_Kinds:
--       A_Procedure_Body_Declaration    (pragmas from declarative region + statements)
--       A_Function_Body_Declaration     (pragmas from declarative region + statements)
--       A_Package_Declaration           (pragmas from visible + private declarative
--                                        regions)
--       A_Package_Body_Declaration      (pragmas from declarative region + statements)
--       A_Task_Body_Declaration         (pragmas from declarative region + statements)
--       A_Protected_Body_Declaration    (pragmas from declarative region)
--       An_Entry_Body_Declaration       (pragmas from declarative region + statements)
--       A_Generic_Procedure_Declaration (pragmas from formal declarative region)
--       A_Generic_Function_Declaration  (pragmas from formal declarative region)
--       A_Generic_Package_Declaration   (pragmas from formal + visible +
--                                          private declarative regions)
--
-- Appropriate Definition_Kinds:
--       A_Record_Definition             (pragmas from the component list)
--       A_Variant_Part                  (pragmas immediately preceding the
--                                        first reserved word "when" + between
--                                        variants)
--       A_Variant                       (pragmas from the component list)
--       A_Task_Definition               (pragmas from visible + private
--                                        declarative regions)
--       A_Protected_Definition          (pragmas from visible + private
--                                        declarative regions)
--
-- Appropriate Statement_Kinds:
--       A_Loop_Statement                (pragmas from statement list)
--       A_While_Loop_Statement          (pragmas from statement list)
--       A_For_Loop_Statement            (pragmas from statement list)
--       A_Block_Statement               (pragmas from declarative region + statements)
--       An_Accept_Statement             (pragmas from statement list +
--
-- Appropriate Representation_Clause_Kinds:
--       A_Record_Representation_Clause  (pragmas from component specifications)
--
-- Returns Element_Kinds:
--       A_Pragma
--
```

```
-------------------------------------------------------------------------------
```
-- 13.38 function Corresponding_Pragmas
```
-------------------------------------------------------------------------------

    function Corresponding_Pragmas (Element : in Asis.Element)
                                   return Asis.Pragma_Element_List;

-------------------------------------------------------------------------------
-- Element - Specifies the element to query
--
-- Returns the list of pragmas semantically associated with the given element,
-- in their order of appearance, or, in any order that does not affect their
-- relative interpretations.  These are pragmas that directly affect the
-- given element.  For example, a pragma Pack affects the type it names.
--
-- Returns a Nil_Element_List if there are no semantically associated pragmas.
--
--|AN Application Note:
--|AN
--|AN If the argument is a inherited entry declaration from a derived task
--|AN type, all pragmas returned are elements taken from the original task
--|AN type's declarative item list.  Their Enclosing_Element is the original
--|AN type definition and not the derived type definition.
--
-- Appropriate Element_Kinds:
--      A_Declaration
--      A_Statement
--
-- Returns Element_Kinds:
--      A_Pragma
--
-------------------------------------------------------------------------------
```
-- 13.39 function Pragma_Name_Image
```
-------------------------------------------------------------------------------

    function Pragma_Name_Image
            (Pragma_Element : in Asis.Pragma_Element) return Program_Text;

-------------------------------------------------------------------------------
-- Pragma_Element  - Specifies the element to query
--
-- Returns the program text image of the simple name of the pragma.
--
-- The case of names returned by this query may vary between implementors.
-- Implementors are encouraged, but not required, to return names in the
-- same case as was used in the original compilation text.
--
-- Appropriate Element_Kinds:
--      A_Pragma
--
-------------------------------------------------------------------------------
```
-- 13.40 function Pragma_Argument_Associations
```
-------------------------------------------------------------------------------

    function Pragma_Argument_Associations
            (Pragma_Element : in Asis.Pragma_Element)
                return Asis.Association_List;

-------------------------------------------------------------------------------
-- Pragma_Element - Specifies the element to query
--
-- Returns a list of the Pragma_Argument_Associations of the pragma, in their
-- order of appearance.
--
-- Appropriate Element_Kinds:
--      A_Pragma
--
-- Returns Element_Kinds:
--      A_Pragma_Argument_Association
--
```

```
-------------------------------------------------------------------------------
```

-- 13.41 function Debug_Image

```
-------------------------------------------------------------------------------

    function Debug_Image (Element : in Asis.Element) return Wide_String;

-------------------------------------------------------------------------------
-- Element - Specifies the program element to convert
--
-- Returns a string value containing implementation-defined debug
-- information associated with the element.
--
-- The return value uses Asis.Text.Delimiter_Image to separate the lines
-- of multi-line results.  The return value does not end with
-- Asis.Text.Delimiter_Image.
--
-- These values are intended for two purposes.  They are suitable for
-- inclusion in problem reports sent to the ASIS implementor.  They can
-- be presumed to contain information useful when debugging the implementation
-- itself. They are also suitable for use by the ASIS application when
-- printing simple application debugging messages during application
-- development.  They are intended to be, to some worthwhile degree,
-- intelligible to the user.
--
-------------------------------------------------------------------------------
```

-- 13.42 function Hash

```
-------------------------------------------------------------------------------

     function Hash (Element : in Asis.Element) return Asis.ASIS_Integer;

-- The purpose of the hash function is to provide a convenient name for an
-- object of type Asis.Element in order to facilitate application defined I/O
-- and/or other application defined processing.
--
-- The hash function maps Asis.Element objects into N discrete classes
-- ("buckets") of objects.  A good hash function is uniform across its range.
-- It is important to note that the distribution of objects in the
-- application's domain will affect the distribution of the hash function.
-- A good hash measured against one domain will not necessarily be good when
-- fed objects from a different set.
--
-- A continuous uniform hash can be divided by any N and provide a uniform
-- distribution of objects to each of the N discrete classes. A hash value is
-- not unique for each hashed Asis.Element. The application is responsible for
-- handling name collisions of the hashed value.
--
-- The hash function returns a hashed value of type ASIS_Integer. If desired,
-- a user could easily map ASIS_Integer'Range to any smaller range for the
-- hash based on application constraints (i.e., the application implementor
-- can tune the time-space tradeoffs by choosing a small table, implying
-- slower lookups within each "bucket", or a large table, implying faster
-- lookups within each "bucket").

-------------------------------------------------------------------------------

end Asis.Elements;
```

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

-- 14 package Asis.Iterator

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
package Asis.Iterator is
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- Asis.Iterator encapsulates the generic procedure Traverse_Element which
-- allows an ASIS application to perform an iterative traversal of a
-- logical syntax tree. It requires the use of two generic procedures,
-- Pre_Operation, which identifies processing for the traversal, and
-- Post_Operation, which identifies processing after the traversal.
-- The State_Information allows processing state to be passed during the
-- iteration of Traverse_Element.
--
-- Package Asis.Iterator is established as a child package to highlight the
-- iteration capability and to facilitate the translation of ASIS to IDL.
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

-- 14.1 procedure Traverse_Element

```
-------------------------------------------------------------------------------

    generic

        type State_Information is limited private;

        with procedure Pre_Operation
                          (Element : in     Asis.Element;
                           Control : in out Traverse_Control;
                           State   : in out State_Information) is <>;

        with procedure Post_Operation
                          (Element : in     Asis.Element;
                           Control : in out Traverse_Control;
                           State   : in out State_Information) is <>;

    procedure Traverse_Element
                 (Element : in     Asis.Element;
                  Control : in out Traverse_Control;
                  State   : in out State_Information);

-------------------------------------------------------------------------------
-- Element             - Specifies the initial element in the traversal
-- Control             - Specifies what next to do with the traversal
-- State_Information   - Specifies other information for the traversal
--
-- Traverses the element and all its component elements, if any.
-- Component elements are all elements that can be obtained by a combination
-- of the ASIS structural queries appropriate for the given element.
--
-- If an element has one or more component elements, each is called a child
-- element.  An element's parent element is its Enclosing_Element.  Children
-- with the same parent are sibling elements.  The type Traverse_Control uses
-- the terms children and siblings to control the traverse.
--
-- For each element, the formal procedure Pre_Operation is called when first
-- visiting the element.  Each of that element's children are then visited
-- and finally the formal procedure Post_Operation is called for the element.
--
-- The order of Element traversal is in terms of the textual representation of
-- the Elements.  Elements are traversed in left-to-right and top-to-bottom
-- order.
--
-- Traversal of Implicit Elements:
--
-- Implicit elements are not traversed by default.  However, they may be
-- explicitly queried and then passed to the traversal instance.  Implicit
-- elements include implicit predefined operator declarations, implicit
-- inherited subprogram declarations, implicit expanded generic specifications
-- and bodies, default expressions supplied to procedure, function, and entry
-- calls, etc.
```

```
--
-- Applications that wish to traverse these implicit Elements shall query for
-- them at the appropriate places in a traversal and then recursively call
-- their instantiation of the traversal generic.  (Implicit elements provided
-- by ASIS do not cover all possible Ada implicit constructs.  For example,
-- implicit initializations for variables of an access type are not provided
-- by ASIS.)
--
-- Traversal of Association lists:
--
-- Argument and association lists for procedure calls, function calls, entry
-- calls, generic instantiations, and aggregates are traversed in their
-- unnormalized forms, as if the Normalized parameter was False for those
-- queries.  Implementations that always normalize certain associations may
-- return Is_Normalized associations.  See the Implementation Permissions
-- for the queries Discriminant_Associations, Generic_Actual_Part,
-- Call_Statement_Parameters, Record_Component_Associations, or
-- Function_Call_Parameters.
--
-- Applications that wish to explicitly traverse normalized associations can
-- do so by querying the appropriate locations in order to obtain the
-- normalized list.  The list can then be traversed by recursively calling
-- the traverse instance.  Once that sub-traversal is finished, the Control
-- parameter can be set to Abandon_Children to skip processing of the
-- unnormalized argument list.
--
-- Traversal can be controlled with the Control parameter.
--
-- A call to an instance of Traverse_Element will not result in calls to
-- Pre_Operation or Post_Operation unless Control is set to Continue.
--
-- The subprograms matching Pre_Operation and Post_Operation can set
-- their Control parameter to affect the traverse:
--
-- - Continue
--
--   o Continues the normal depth-first traversal.
--
-- - Abandon_Children
--
--   o Prevents traversal of the current element's children.
--
--   o If set in a Pre_Operation, traversal picks up with the next sibling element
--     of the current element.
--
--   o If set in a Post_Operation, this is the same as Continue, all children will already
--     have been traversed.  Traversal picks up with the Post_Operation of the parent.
--
-- - Abandon_Siblings
--
--   o Prevents traversal of the current element's children and remaining siblings.
--
--   o If set in a Pre_Operation, this abandons the associated Post_Operation for the
--     current element.  Traversal picks up with the Post_Operation of the parent.
--
--   o If set in a Post_Operation, traversal picks up with the Post_Operation of the
--     parent.
--
-- - Terminate_Immediately
--
--   o Does exactly that.
--
-- Raises ASIS_Inappropriate_Element if the element is a Nil_Element
--
---------------------------------------------------------------------------------------

end Asis.Iterator;
```

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

-- 15 package Asis.Declarations

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
package Asis.Declarations is
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- Asis.Declarations encapsulates a set of queries that operate on
-- A_Defining_Name and A_Declaration elements.
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
--|ER A_Declaration - 3.1
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

-- 15.1 function Names

```
-------------------------------------------------------------------------------

    function Names (Declaration : in Asis.Declaration)
                   return Asis.Defining_Name_List;

-------------------------------------------------------------------------------
-- Declaration - Specifies the element to query
--
-- Returns a list of names defined by the declaration, in their order of
-- appearance.  Declarations that define a single name will return a list of
-- length one.
--
-- Returns Nil_Element_List for A_Declaration Elements representing the (implicit)
-- declarations of universal and root numeric type (that is, if Type_Kind
-- (Type_Declaration_View (Declaration) = A_Root_Type_Definition.
--
-- Examples:
--
--     type Foo is (Pooh, Baah);
--        -- Returns a list containing one A_Defining_Name: Foo.
--
--     One, Uno : constant Integer := 1;
--        -- Returns a list of two A_Defining_Name elements: One and Uno.
--
-- Function designators that define operators are A_Defining_Operator_Symbol.
--
-- Results of this query may vary across ASIS implementations.  Some
-- implementations may normalize all multi-name declarations into an
-- equivalent series of corresponding single name declarations.  For those
-- implementations, this query will always return a list containing a single
-- name.  See Reference Manual 3.3.1(7).
--
-- Appropriate Element_Kinds:
--       A_Declaration
--
-- Returns Element_Kinds:
--       A_Defining_Name
--
--|ER---------------------------------------------------------------------------
--|ER A_Defining_Name - 3.1
--|ER---------------------------------------------------------------------------
--|ER A_Defining_Identifier      - 3.1 - no child elements
--|ER A_Defining_Operator_Symbol - 6.1 - no child elements
--|ER
--|ER A string image returned by:
--|ER    function Defining_Name_Image
--
```

```
-------------------------------------------------------------------------------
```

-- 15.2 function Defining_Name_Image

```
-------------------------------------------------------------------------------

    function Defining_Name_Image
            (Defining_Name : in Asis.Defining_Name) return Program_Text;

-------------------------------------------------------------------------------
-- Defining_Name  - Specifies the element to query
--
-- Returns the program text image of the name.  Embedded quotes (for operator
-- designator strings) are doubled.
--
-- A_Defining_Identifier elements are simple identifier names "Abc" (name Abc).
--
-- A_Defining_Operator_Symbol elements have names with embedded quotes
-- """abs""" (function "abs").
--
-- A_Defining_Character_Literal elements have names with embedded apostrophes
-- "'x'" (literal 'x').
--
-- A_Defining_Enumeration_Literal elements have simple identifier names
-- "Blue" (literal Blue). If A_Defining_Enumeration_Literal element is of type
-- Character or Wide_Character but does not have a graphical presentation, then
-- the result is implementation-dependent.
--
-- A_Defining_Expanded_Name elements are prefix.selector names "A.B.C"
-- (name A.B.C).
--
-- The case of names returned by this query may vary between implementors.
-- Implementors are encouraged, but not required, to return names in the
-- same case as was used in the original compilation text.
--
-- The Defining_Name_Image of a label_statement_identifier does not include
-- the enclosing "<<" and ">>" that form the label syntax.  Similarly, the
-- Defining_Name_Image of an identifier for a loop_statement or block_statement
-- does not include the trailing colon that forms the loop name syntax.
-- Use Asis.Text.Element_Image or Asis.Text.Lines queries to obtain these
-- syntactic constructs and any comments associated with them.
--
-- Appropriate Element_Kinds:
--      A_Defining_Name
--|ER---------------------------------------------------------------------------
--|ER A_Defining_Character_Literal   - 3.5.1 - no child elements
--|ER A_Defining_Enumeration_Literal - 3.5.1 - no child elements
--|ER
--|ER A program text image returned by:
--|ER    function Defining_Name_Image
--|ER
--|ER A program text image of the enumeration literal value returned by:
--|ER    function Position_Number_Image
--|ER    function Representation_Value_Image
--
-------------------------------------------------------------------------------
```

-- 15.3 function Position_Number_Image

```
-------------------------------------------------------------------------------

    function Position_Number_Image
            (Defining_Name : in Asis.Defining_Name) return Wide_String;

-------------------------------------------------------------------------------
-- Expression  - Specifies the literal expression to query
--
-- Returns the program text image of the position number of the value of the
-- enumeration literal.
--
-- The program text returned is the image of the universal_integer value that is
-- returned by the attribute 'Pos if it were applied to the value.
-- For example: Integer'Image(Color'Pos(Blue)).
--
-- Appropriate Defining_Name_Kinds:
--      A_Defining_Character_Literal
--      A_Defining_Enumeration_Literal
```

```
--
---------------------------------------------------------------------------------------
```

-- 15.4 function Representation_Value_Image

```
---------------------------------------------------------------------------------------

    function Representation_Value_Image
            (Defining_Name : in Asis.Defining_Name) return Wide_String;

---------------------------------------------------------------------------------------
-- Expression  - Specifies the literal expression to query
--
-- Returns the string image of the internal code for the enumeration literal.
--
-- If a representation_clause is defined for the enumeration type then the
-- string returned is the Integer'Wide_Image of the corresponding value given in
-- the enumeration_aggregate.  Otherwise, the string returned is the same as
-- the Position_Number_Image.
--
-- Appropriate Defining_Name_Kinds:
--      A_Defining_Character_Literal
--      A_Defining_Enumeration_Literal
--
--|ER-----------------------------------------------------------------------------------
--|ER A_Defining_Expanded_Name - 6.1
--|ER
--|ER A string image returned by:
--|ER    function Defining_Name_Image
--|CR
--|CR Child elements returned by:
--|CR    function Defining_Prefix
--|CR    function Defining_Selector
--
---------------------------------------------------------------------------------------
```

-- 15.5 function Defining_Prefix

```
---------------------------------------------------------------------------------------

    function Defining_Prefix (Defining_Name : in Asis.Defining_Name)
                       return Asis.Name;

---------------------------------------------------------------------------------------
-- Defining_Name  - Specifies the element to query
--
-- Returns the element that forms the prefix of the name.  The prefix is the
-- name to the left of the rightmost 'dot' in the expanded name.
-- The Defining_Prefix of A.B is A, and of A.B.C is A.B.
--
-- Appropriate Defining_Name_Kinds:
--      A_Defining_Expanded_Name
--
-- Returns Expression_Kinds:
--      An_Identifier
--      A_Selected_Component
--
---------------------------------------------------------------------------------------
```

-- 15.6 function Defining_Selector

```
---------------------------------------------------------------------------------------

    function Defining_Selector (Defining_Name : in Asis.Defining_Name)
                         return Asis.Defining_Name;

---------------------------------------------------------------------------------------
-- Defining_Name  - Specifies the element to query
--
-- Returns the element that forms the selector of the name.  The selector is
-- the name to the right of the rightmost 'dot' in the expanded name.
-- The Defining_Selector of A.B is B, and of A.B.C is C.
--
-- Appropriate Defining_Name_Kinds:
--      A_Defining_Expanded_Name
--
-- Returns Defining_Name_Kinds:
--      A_Defining_Identifier
```

```
--
--|ER--------------------------------------------------------------------------
--|ER An_Ordinary_Type_Declaration - 3.2.1
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Discriminant_Part
--|CR    function Type_Declaration_View
--
-------------------------------------------------------------------------------
```

-- 15.7 function Discriminant_Part

```
-------------------------------------------------------------------------------

    function Discriminant_Part (Declaration : in Asis.Declaration)
                           return Asis.Definition;

------------------------------------------------------------------------------
-- Declaration - Specifies the type declaration to query
--
-- Returns the discriminant_part, if any, from the type_declaration or
-- formal_type_declaration.
--
-- Returns a Nil_Element if the Declaration has no explicit discriminant_part.
--
-- Appropriate Declaration_Kinds:
--      An_Ordinary_Type_Declaration
--      A_Task_Type_Declaration
--      A_Protected_Type_Declaration
--      An_Incomplete_Type_Declaration
--      A_Private_Type_Declaration
--      A_Private_Extension_Declaration
--      A_Formal_Type_Declaration
--
-- Returns Definition_Kinds:
--      Not_A_Definition
--      An_Unknown_Discriminant_Part
--      A_Known_Discriminant_Part
--
-------------------------------------------------------------------------------
```

-- 15.8 function Type_Declaration_View

```
-------------------------------------------------------------------------------

    function Type_Declaration_View (Declaration : in Asis.Declaration)
                  return Asis.Definition;

-------------------------------------------------------------------------------
-- Declaration - Specifies the declaration element to query
--
-- Returns the definition characteristics that form the view of the
-- type_declaration.  The view is the remainder of the declaration following
-- the reserved word "is".
--
-- For a full_type_declaration, returns the type_definition, task_definition,
-- or protected_definition following the reserved word "is" in the declaration.
--
-- Returns a Nil_Element for a task_type_declaration that has no explicit
-- task_definition.
--
-- For a private_type_declaration or private_extension_declaration, returns
-- the definition element representing the private declaration view.
--
-- For a subtype_declaration, returns the subtype_indication.
--
-- For a formal_type_declaration, returns the formal_type_definition.
--
-- Appropriate Declaration_Kinds:
--       An_Ordinary_Type_Declaration
--       A_Task_Type_Declaration
--       A_Protected_Type_Declaration
--       A_Private_Type_Declaration
--       A_Private_Extension_Declaration
--       A_Subtype_Declaration
--       A_Formal_Type_Declaration
```

```
--
-- Returns Definition_Kinds:
--       Not_A_Definition
--       A_Type_Definition
--       A_Subtype_Indication
--       A_Private_Type_Definition
--       A_Tagged_Private_Type_Definition
--       A_Private_Extension_Definition
--       A_Task_Definition
--       A_Protected_Definition
--       A_Formal_Type_Definition
--
--|ER----------------------------------------------------------------------------------
--|ER A_Subtype_Declaration - 3.2.2
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Type_Declaration_View
--|ER----------------------------------------------------------------------------------
--|ER A_Variable_Declaration          - 3.3.1
--|CR
--|CR Child elements:
--|CR    function Names
--|CR    function Object_Declaration_View
--|CR    function Initialization_Expression
--
-------------------------------------------------------------------------------------
```

-- 15.9 function Object_Declaration_View

```
-------------------------------------------------------------------------------------

    function Object_Declaration_View (Declaration : in Asis.Declaration)
                  return Asis.Definition;

-------------------------------------------------------------------------------------
-- Declaration - Specifies the declaration element to query
--
-- Returns the definition characteristics that form the view of the
-- object_declaration.  The view is the subtype_indication or full type
-- definition of the object_declaration.  An initial value, if any, is not
-- part of this view.
--
-- For a single_task_declaration or single_protected_declaration, returns
-- the task_definition or protected_definition following the reserved word "is".
--
-- Returns a Nil_Element for a single_task_declaration that has no explicit
-- task_definition.
--
-- For a Component_Declaration, returns the Component_Definition following
-- the colon.
--
-- For all other object_declaration variables or constants, returns the
-- subtype_indication or array_type_definition following the colon.
--
-- Appropriate Declaration_Kinds:
--       A_Variable_Declaration
--       A_Constant_Declaration
--       A_Deferred_Constant_Declaration
--       A_Single_Protected_Declaration
--       A_Single_Task_Declaration
--       A_Component_Declaration
--
-- Returns Definition_Kinds:
--       Not_A_Definition
--       A_Type_Definition
--           Returns Type_Kinds:
--                 A_Constrained_Array_Definition
--       A_Subtype_Indication
--       A_Task_Definition
--       A_Protected_Definition
--       A_Component_Definition
--
```

```
-------------------------------------------------------------------------------
```
-- 15.10 function Initialization_Expression
```
-------------------------------------------------------------------------------

    function Initialization_Expression (Declaration : in Asis.Declaration)
                           return Asis.Expression;

-------------------------------------------------------------------------------
-- Declaration - Specifies the object declaration to query
--
-- Returns the initialization expression [:= expression] of the declaration.
--
-- Returns a Nil_Element if the declaration does not include an explicit
-- initialization.
--
-- Appropriate Declaration_Kinds:
--      A_Variable_Declaration
--      A_Constant_Declaration
--      An_Integer_Number_Declaration
--      A_Real_Number_Declaration
--      A_Discriminant_Specification
--      A_Component_Declaration
--      A_Parameter_Specification
--      A_Formal_Object_Declaration
--
-- Returns Element_Kinds:
--      Not_An_Element
--      An_Expression
--
--|ER---------------------------------------------------------------------------
--|ER A_Constant_Declaration          - 3.3.1
--|CR
--|CR Child elements:
--|CR    function Names
--|CR    function Object_Declaration_View
--|CR    function Initialization_Expression
--|CR
--|CR Element queries that provide semantically related elements:
--|CR    function Corresponding_Constant_Declaration
--
-------------------------------------------------------------------------------
```
-- 15.11 function Corresponding_Constant_Declaration
```
-------------------------------------------------------------------------------

    function Corresponding_Constant_Declaration
                (Name : in Asis.Defining_Name)
                return Asis.Declaration;

-------------------------------------------------------------------------------
-- Name    - Specifies the name of a constant declaration to query
--
-- Returns the corresponding full constant declaration when given the name
-- from a deferred constant declaration.
--
-- Returns the corresponding deferred constant declaration when given the name
-- from a full constant declaration.
--
-- Returns A_Pragma if the deferred constant declaration is completed
-- by pragma Import.
--
-- Returns a Nil_Element if the full constant declaration has no corresponding
-- deferred constant declaration.
--
-- Raises ASIS_Inappropriate_Element with a Status of Value_Error if the
-- argument is not the name of a constant or a deferred constant.
--
-- The name of a constant declaration is available from both the Names and the
-- Corresponding_Name_Definition queries.
--
-- Appropriate Element_Kinds:
--      A_Defining_Name
--
```

```
-- Returns Declaration_Kinds:
--      Not_A_Declaration
--      A_Constant_Declaration
--      A_Deferred_Constant_Declaration
--
-- Returns Element_Kinds:
--      Not_An_Element
--      A_Declaration
--      A_Pragma
--|ER----------------------------------------------------------------------------
--|ER A_Deferred_Constant_Declaration          - 3.3.1
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Object_Declaration_View
--|ER----------------------------------------------------------------------------
--|ER An_Integer_Number_Declaration - 3.3.2
--|ER A_Real_Number_Declaration     - 3.3.2
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Initialization_Expression
--|ER----------------------------------------------------------------------------
--|ER An_Enumeration_Literal_Specification - 3.5.1
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|ER----------------------------------------------------------------------------
--|ER A_Discriminant_Specification         - 3.7
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Declaration_Subtype_Mark
--|CR    function Initialization_Expression
--
-------------------------------------------------------------------------------
```

-- 15.12 function Declaration_Subtype_Mark

```
-------------------------------------------------------------------------------

    function Declaration_Subtype_Mark (Declaration : in Asis.Declaration)
                                  return Asis.Expression;

-------------------------------------------------------------------------------
-- Declaration - Specifies the declaration element to query
--
-- Returns the expression element that names the subtype_mark of the
-- declaration.
--
-- Appropriate Declaration_Kinds:
--      A_Discriminant_Specification
--      A_Parameter_Specification
--      A_Formal_Object_Declaration
--      An_Object_Renaming_Declaration
--
-- Returns Expression_Kinds:
--      An_Identifier
--      A_Selected_Component
--      An_Attribute_Reference
--
--|ER----------------------------------------------------------------------------
--|ER A_Component_Declaration - 3.8
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Object_Declaration_View
--|CR    function Initialization_Expression
--|ER----------------------------------------------------------------------------
--|ER An_Incomplete_Type_Declaration - 3.10.1
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Discriminant_Part
--
```

```
-------------------------------------------------------------------------------
```
-- 15.13 function Corresponding_Type_Declaration
```
-------------------------------------------------------------------------------

    function Corresponding_Type_Declaration
                (Declaration : in Asis.Declaration) return Asis.Declaration;

    function Corresponding_Type_Declaration
                (Declaration : in Asis.Declaration;
                 The_Context : in Asis.Context) return Asis.Declaration;
-------------------------------------------------------------------------------
-- Declaration - Specifies the type declaration to query
-- The_Context - Specifies the program Context to use for obtaining package
--               body information
--
-- Returns the corresponding full type declaration when given a private or
-- incomplete type declaration.  Returns the corresponding private or
-- incomplete type declaration when given a full type declaration.
--
-- These two function calls will always produce identical results:
--
--     Decl2 := Corresponding_Type_Declaration ( Decl1 );
--     Decl2 := Corresponding_Type_Declaration
--              ( Decl1,
--                Enclosing_Context ( Enclosing_Compilation_Unit ( Decl1 )));
--
-- Returns a Nil_Element when a full type declaration is given that has no
-- corresponding private or incomplete type declaration, or when a
-- corresponding type declaration does not exist within The_Context.
--
-- The parameter The_Context is used whenever the corresponding full type of
-- an incomplete type is in a corresponding package body.  See Reference Manual
-- 3.10.1(3). Any non-Nil result will always have the given Context as its
-- Enclosing_Context.
--
-- Appropriate Declaration_Kinds:
--      An_Ordinary_Type_Declaration
--      A_Task_Type_Declaration
--      A_Protected_Type_Declaration
--      An_Incomplete_Type_Declaration
--      A_Private_Type_Declaration
--      A_Private_Extension_Declaration
--
-- Returns Declaration_Kinds:
--      Not_A_Declaration
--      An_Ordinary_Type_Declaration
--      A_Task_Type_Declaration
--      A_Protected_Type_Declaration
--      An_Incomplete_Type_Declaration
--      A_Private_Type_Declaration
--      A_Private_Extension_Declaration
--
-------------------------------------------------------------------------------
```
-- 15.14 function Corresponding_First_Subtype
```
-------------------------------------------------------------------------------

    function Corresponding_First_Subtype
                  (Declaration : in Asis.Declaration)
                      return Asis.Declaration;

-------------------------------------------------------------------------------
-- Declaration - Specifies the subtype_declaration to query
--
-- This function recursively unwinds subtyping to return at a type_declaration
-- that defines the first subtype of the argument.
--
-- Returns a declaration that Is_Identical to the argument if the argument is
-- already the first subtype.
--
-- Appropriate Declaration_Kinds:
--      An_Ordinary_Type_Declaration
--      A_Task_Type_Declaration
--      A_Protected_Type_Declaration
```

```
--      A_Private_Type_Declaration
--      A_Private_Extension_Declaration
--      A_Subtype_Declaration
--      A_Formal_Type_Declaration
--
-- Returns Declaration_Kinds:
--      An_Ordinary_Type_Declaration
--      A_Task_Type_Declaration
--      A_Protected_Type_Declaration
--      A_Private_Type_Declaration
--      A_Private_Extension_Declaration
--      A_Formal_Type_Declaration
--
-----------------------------------------------------------------------------------
```

-- 15.15 function Corresponding_Last_Constraint

```
-----------------------------------------------------------------------------------

    function Corresponding_Last_Constraint
                 (Declaration : in Asis.Declaration)
                     return Asis.Declaration;

-----------------------------------------------------------------------------------
-- Declaration - Specifies the subtype_declaration or type_declaration to query.
--
-- This function recursively unwinds subtyping to return at a declaration
-- that is either a type_declaration or subtype_declaration that imposes
-- an explicit constraint on the argument.
--
-- Unwinds a minimum of one level of subtyping even if an argument declaration
-- itself has a constraint.
--
-- Returns a declaration that Is_Identical to the argument if the argument is
-- a type_declaration, i.e. the first subtype.
--
-- Appropriate Declaration_Kinds:
--      An_Ordinary_Type_Declaration
--      A_Task_Type_Declaration
--      A_Protected_Type_Declaration
--      A_Private_Type_Declaration
--      A_Private_Extension_Declaration
--      A_Subtype_Declaration
--      A_Formal_Type_Declaration
--
-- Returns Declaration_Kinds:
--      An_Ordinary_Type_Declaration
--      A_Task_Type_Declaration
--      A_Protected_Type_Declaration
--      A_Private_Type_Declaration
--      A_Private_Extension_Declaration
--      A_Subtype_Declaration
--      A_Formal_Type_Declaration
--
-----------------------------------------------------------------------------------
```

-- 15.16 function Corresponding_Last_Subtype

```
-----------------------------------------------------------------------------------

    function Corresponding_Last_Subtype
                 (Declaration : in Asis.Declaration)
                     return Asis.Declaration;

-----------------------------------------------------------------------------------
-- Declaration - Specifies the subtype_declaration or type_declaration to query.
--
-- This function unwinds subtyping a single level to arrive at a declaration
-- that is either a type_declaration or subtype_declaration.
--
-- Returns a declaration that Is_Identical to the argument if the argument is
-- a type_declaration (i.e., the first subtype).
--
-- Appropriate Declaration_Kinds:
--      An_Ordinary_Type_Declaration
--      A_Task_Type_Declaration
--      A_Protected_Type_Declaration
```

```
--      A_Private_Type_Declaration
--      A_Private_Extension_Declaration
--      A_Subtype_Declaration
--      A_Formal_Type_Declaration
--
-- Returns Declaration_Kinds:
--      An_Ordinary_Type_Declaration
--      A_Task_Type_Declaration
--      A_Protected_Type_Declaration
--      A_Private_Type_Declaration
--      A_Private_Extension_Declaration
--      A_Subtype_Declaration
--      A_Formal_Type_Declaration
--
---------------------------------------------------------------------------------------
```

-- 15.17 function Corresponding_Representation_Clauses

```
---------------------------------------------------------------------------------------

    function Corresponding_Representation_Clauses
                (Declaration : in Asis.Declaration)
                return Asis.Representation_Clause_List;

---------------------------------------------------------------------------------------
-- Declaration - Specifies the declaration to query
--
-- Returns all representation_clause elements that apply to the declaration.
--
-- Returns a Nil_Element_List if no clauses apply to the declaration.
--
-- The clauses returned may be the clauses applying to a parent type if the
-- type is a derived type with no explicit representation.  These clauses
-- are not Is_Part_Of_Implicit, they are the representation_clause elements
-- specified in conjunction with the declaration of the parent type.
--
-- All Declaration_Kinds are appropriate except Not_A_Declaration.
--
-- Returns Clause_Kinds:
--      A_Representation_Clause
--|ER-----------------------------------------------------------------------------------
--|ER A_Loop_Parameter_Specification         - 5.5
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Specification_Subtype_Definition
--
---------------------------------------------------------------------------------------
```

-- 15.18 function Specification_Subtype_Definition

```
---------------------------------------------------------------------------------------

    function Specification_Subtype_Definition
              (Specification : in Asis.Declaration)
                  return Asis.Discrete_Subtype_Definition;

---------------------------------------------------------------------------------------
-- Specification - Specifies the loop_parameter_specification or
--                 Entry_Index_Specification to query
--
-- Returns the Discrete_Subtype_Definition of the specification.
--
-- Appropriate Declaration_Kinds:
--      A_Loop_Parameter_Specification
--      An_Entry_Index_Specification
--
-- Returns Definition_Kinds:
--      A_Discrete_Subtype_Definition
--
--|ER-----------------------------------------------------------------------------------
--|ER A_Procedure_Declaration           - 6.1
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Parameter_Profile
--
```

```
-------------------------------------------------------------------------------
```
-- 15.19 function Parameter_Profile
```
-------------------------------------------------------------------------------

    function Parameter_Profile (Declaration : in Asis.Declaration)
                           return Asis.Parameter_Specification_List;

-------------------------------------------------------------------------------
-- Declaration - Specifies the subprogram or entry declaration to query
--
-- Returns a list of parameter specifications in the formal part of the
-- subprogram or entry declaration, in their order of appearance.
--
-- Returns a Nil_Element_List if the subprogram or entry has no
-- parameters.
--
-- Results of this query may vary across ASIS implementations.  Some
-- implementations normalize all multiple name parameter specifications into
-- an equivalent sequence of corresponding single name parameter
-- specifications.  See Reference Manual 3.3.1(7).
--
-- Appropriate Declaration_Kinds:
--      A_Procedure_Declaration
--      A_Function_Declaration
--      A_Procedure_Body_Declaration
--      A_Function_Body_Declaration
--      A_Procedure_Renaming_Declaration
--      A_Function_Renaming_Declaration
--      An_Entry_Declaration
--      An_Entry_Body_Declaration
--      A_Procedure_Body_Stub
--      A_Function_Body_Stub
--      A_Generic_Function_Declaration
--      A_Generic_Procedure_Declaration
--      A_Formal_Function_Declaration
--      A_Formal_Procedure_Declaration
--
-- Returns Declaration_Kinds:
--      A_Parameter_Specification
--
--|ER----------------------------------------------------------------------------
--|ER A_Function_Declaration           - 6.1
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Parameter_Profile
--|CR    function Result_Profile
--
-------------------------------------------------------------------------------
```
-- 15.20 function Result_Profile
```
-------------------------------------------------------------------------------

    function Result_Profile (Declaration : in Asis.Declaration)
                     return Asis.Expression;

-------------------------------------------------------------------------------
-- Declaration - Specifies the function declaration to query
--
-- Returns the subtype mark expression for the return type for any function
-- declaration.
--
-- Appropriate Declaration_Kinds:
--      A_Function_Declaration
--      A_Function_Body_Declaration
--      A_Function_Body_Stub
--      A_Function_Renaming_Declaration
--      A_Generic_Function_Declaration
--      A_Formal_Function_Declaration
--
-- Returns Expression_Kinds:
--      An_Identifier
--      A_Selected_Component
--      An_Attribute_Reference
```

```
--
--|ER----------------------------------------------------------------------------
--|ER A_Parameter_Specification         - 6.1
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Declaration_Subtype_Mark
--|CR    function Initialization_Expression
--|ER----------------------------------------------------------------------------
--|ER A_Procedure_Body_Declaration - 6.3
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Parameter_Profile
--|CR    function Body_Declarative_Items
--|CR    function Body_Statements
--|CR    function Body_Exception_Handlers
--|CR    function Body_Block_Statement    - obsolescent, not recommended
--
---------------------------------------------------------------------------------
```

-- 15.21 function Body_Declarative_Items

```
---------------------------------------------------------------------------------

    function Body_Declarative_Items (Declaration : in Asis.Declaration;
                                     Include_Pragmas : in Boolean := False)
                                    return Asis.Element_List;

---------------------------------------------------------------------------------
-- Declaration     - Specifies the body declaration to query
-- Include_Pragmas - Specifies whether pragmas are to be returned
--
-- Returns a list of all basic declarations, representation specifications,
-- use clauses, and pragmas in the declarative part of the body, in their
-- order of appearance.
--
-- Returns a Nil_Element_List if there are no declarative_item or pragma elements.
--
-- Results of this query may vary across ASIS implementations.  Some
-- implementations normalize all multi-name declarations into an
-- equivalent sequence of corresponding single name object declarations.
-- See Reference Manual 3.3.1(7).
--
-- Appropriate Declaration_Kinds:
--      A_Function_Body_Declaration
--      A_Procedure_Body_Declaration
--      A_Package_Body_Declaration
--      A_Task_Body_Declaration
--      An_Entry_Body_Declaration
--
-- Returns Element_Kinds:
--      A_Pragma
--      A_Declaration
--      A_Clause
--
---------------------------------------------------------------------------------
```

-- 15.22 function Body_Statements

```
---------------------------------------------------------------------------------

    function Body_Statements (Declaration : in Asis.Declaration;
                              Include_Pragmas : in Boolean := False)
                             return Asis.Statement_List;

---------------------------------------------------------------------------------
-- Declaration     - Specifies the body declaration to query
-- Include_Pragmas - Specifies whether pragmas are to be returned
--
-- Returns a list of the statements and pragmas for the body, in
-- their order of appearance.
--
-- Returns a Nil_Element_List if there are no statements or pragmas.
--
```

```
-- Appropriate Declaration_Kinds:
--      A_Function_Body_Declaration
--      A_Procedure_Body_Declaration
--      A_Package_Body_Declaration
--      A_Task_Body_Declaration
--      An_Entry_Body_Declaration
--
-- Returns Element_Kinds:
--      A_Pragma
--      A_Statement
--
---------------------------------------------------------------------------------------
```

-- 15.23 function Body_Exception_Handlers

```
---------------------------------------------------------------------------------------

    function Body_Exception_Handlers (Declaration : in Asis.Declaration;
                                      Include_Pragmas : in Boolean := False)
                                     return Asis.Exception_Handler_List;

---------------------------------------------------------------------------------------
-- Declaration - Specifies the body declaration to query
-- Include_Pragmas - Specifies whether pragmas are to be returned
--
-- Returns a list of the exception_handler elements of the body, in their order of
-- appearance.
--
-- The only pragmas returned are those following the reserved word "exception"
-- and preceding the reserved word "when" of first exception handler.
--
-- Returns a Nil_Element_List if there are no exception_handler or pragma elements.
--
-- Appropriate Declaration_Kinds:
--      A_Function_Body_Declaration
--      A_Procedure_Body_Declaration
--      A_Package_Body_Declaration
--      A_Task_Body_Declaration
--      An_Entry_Body_Declaration
--
-- Returns Element_Kinds:
--      An_Exception_Handler
--      A_Pragma
--
--|ER-----------------------------------------------------------------------------------
--|ER A_Function_Body_Declaration  - 6.3
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Parameter_Profile
--|CR    function Result_Profile
--|CR    function Body_Declarative_Items
--|CR    function Body_Statements
--|CR    function Body_Exception_Handlers
--|CR    function Body_Block_Statement    - obsolescent, not recommended
--
---------------------------------------------------------------------------------------
```

-- 15.24 function Body_Block_Statement

```
---------------------------------------------------------------------------------------
-- Function Body_Block_Statement is a new query that supplies the
-- equivalent combined functionality of the replaced queries:
-- Subprogram_Body_Block, Package_Body_Block, and Task_Body_Block.
-- Use of the query Body_Block_Statement is not recommended in new programs.
-- This functionality is redundant with the queries Body_Declarative_Items,
-- Body_Statements, and Body_Exception_Handlers.
---------------------------------------------------------------------------------------

    function Body_Block_Statement (Declaration : in Asis.Declaration)
                                  return Asis.Statement;

---------------------------------------------------------------------------------------
-- Declaration - Specifies the program unit body to query
--
-- Returns a block statement that is the structural equivalent of the body.
-- The block statement is not Is_Part_Of_Implicit.  The block includes
```

```
-- the declarative part, the sequence of statements, and any exception
-- handlers.
--
-- Appropriate Declaration_Kinds:
--      A_Function_Body_Declaration
--      A_Procedure_Body_Declaration
--      A_Package_Body_Declaration
--      A_Task_Body_Declaration
--      An_Entry_Body_Declaration
--
-- Returns Statement_Kinds:
--      A_Block_Statement
--
--|AN Application Note:
--|AN
--|AN This function is an obsolescent feature retained for compatibility with
--|AN ASIS 83.  It is never called by Traverse_Element.  Use of this query is
--|AN not recommended in new programs.
--
---------------------------------------------------------------------------------------
```

-- 15.25 function Is_Name_Repeated

```
---------------------------------------------------------------------------------------

    function Is_Name_Repeated
                (Declaration : in Asis.Declaration) return Boolean;

---------------------------------------------------------------------------------------
-- Declaration - Specifies the declaration to query
--
-- Returns True if the name of the declaration is repeated after the "end"
-- which terminates the declaration.
--
-- Returns False for any unexpected Element.
--
-- Expected Declaration_Kinds:
--      A_Package_Declaration
--      A_Package_Body_Declaration
--      A_Procedure_Body_Declaration
--      A_Function_Body_Declaration
--      A_Generic_Package_Declaration
--      A_Task_Type_Declaration
--      A_Single_Task_Declaration
--      A_Task_Body_Declaration
--      A_Protected_Type_Declaration
--      A_Single_Protected_Declaration
--      A_Protected_Body_Declaration
--      An_Entry_Body_Declaration
--
---------------------------------------------------------------------------------------
```

-- 15.26 function Corresponding_Declaration

```
---------------------------------------------------------------------------------------

    function Corresponding_Declaration
                (Declaration : in Asis.Declaration)
                return Asis.Declaration;

    function Corresponding_Declaration
                (Declaration : in Asis.Declaration;
                 The_Context : in Asis.Context)
                return Asis.Declaration;

---------------------------------------------------------------------------------------
-- Declaration     - Specifies the specification to query
-- The_Context     - Specifies a Context to use
--
-- Returns the corresponding specification of a subprogram, package, or task
-- body declaration.  Returns the expanded generic specification template for
-- generic instantiations.  The argument can be a Unit_Declaration from a
-- Compilation_Unit, or, it can be any appropriate body declaration from any
-- declarative context.
--
```

```
-- These two function calls will always produce identical results:
--
--     Decl2 := Corresponding_Declaration (Decl1);
--     Decl2 := Corresponding_Declaration
--              (Decl1, Enclosing_Context ( Enclosing_Compilation_Unit ( Decl1 )));
--
-- If a specification declaration is given, the same element is returned,
-- unless it is a generic instantiation or an inherited subprogram declaration
-- (see below).
--
-- If a subprogram renaming declaration is given:
--
-- a) in case of renaming-as-declaration, the same element is returned;
--
-- b) in case of renaming-as-body, the subprogram declaration completed
--    by this subprogram renaming declaration is returned.
--    (Reference Manual, 8.5.4(1))
--
-- Returns a Nil_Element if no explicit specification exists, or the
-- declaration is the proper body of a subunit.
--
-- The parameter The_Context is used to locate the corresponding specification
-- within a particular Context.  The_Context need not be the Enclosing_Context
-- of the Declaration.  Any non-Nil result will always have The_Context
-- as its Enclosing_Context.  This implies that while a non-Nil result may be
-- Is_Equal with the argument, it will only be Is_Identical if the
-- Enclosing_Context of the Declaration is the same as the parameter
-- The_Context.
--
-- If a generic instantiation is given, the expanded generic specification
-- template representing the instance is returned and Is_Part_Of_Instance.
-- For example, an argument that is A_Package_Instantiation, results in a
-- value that is A_Package_Declaration that can be analyzed with all
-- appropriate queries.
--
-- Retuns the declaration of the generic child unit corresponding to an
-- implicit generic child unit specification. Reference Manual 10.1.1(19).
--
-- The Enclosing_Element of the expanded specification is the generic
-- instantiation.  The Enclosing_Compilation_Unit of the expanded template is
-- that of the instantiation.
--
-- If an inherited subprogram declaration is given, the specification
-- returned is the one for the user-defined subprogram from which the
-- argument was ultimately inherited.
--
-- Appropriate Declaration_Kinds returning a specification:
--      A_Function_Body_Declaration
--      A_Function_Renaming_Declaration (renaming-as-body)
--      A_Function_Body_Stub
--      A_Function_Instantiation
--      A_Package_Body_Declaration
--      A_Package_Body_Stub
--      A_Package_Instantiation
--      A_Procedure_Body_Declaration
--      A_Procedure_Renaming_Declaration (renaming-as-body)
--      A_Procedure_Body_Stub
--      A_Procedure_Instantiation
--      A_Task_Body_Declaration
--      A_Task_Body_Stub
--      A_Protected_Body_Declaration
--      A_Protected_Body_Stub
--      A_Formal_Package_Declaration
--      A_Formal_Package_Declaration_With_Box
--      A_Generic_Package_Renaming_Declaration
--      A_Generic_Procedure_Renaming_Declaration
--      A_Generic_Function_Renaming_Declaration
--      An_Entry_Body_Declaration
--
-- Appropriate Declaration_Kinds returning the argument Declaration:
--      A_Function_Declaration
--      A_Function_Renaming_Declaration (renaming-as-declaration)
--      A_Generic_Function_Declaration
--      A_Generic_Package_Declaration
```

```
--      A_Generic_Procedure_Declaration
--      A_Package_Declaration
--      A_Package_Renaming_Declaration
--      A_Procedure_Declaration
--      A_Procedure_Renaming_Declaration (renaming-as-declaration)
--      A_Single_Task_Declaration
--      A_Task_Type_Declaration
--      A_Protected_Type_Declaration
--      A_Single_Protected_Declaration
--      A_Generic_Package_Renaming_Declaration
--      A_Generic_Procedure_Renaming_Declaration
--      A_Generic_Function_Renaming_Declaration
--      An_Entry_Declaration
--
-- Returns Declaration_Kinds:
--      Not_A_Declaration
--      A_Function_Declaration
--      A_Function_Renaming_Declaration
--      A_Generic_Function_Declaration
--      A_Generic_Package_Declaration
--      A_Generic_Procedure_Declaration
--      A_Package_Declaration
--      A_Package_Renaming_Declaration
--      A_Procedure_Declaration
--      A_Procedure_Renaming_Declaration
--      A_Single_Task_Declaration
--      A_Task_Type_Declaration
--      A_Protected_Type_Declaration
--      A_Single_Protected_Declaration
--      An_Entry_Declaration
--
------------------------------------------------------------------------------------
```

-- 15.27 function Corresponding_Body

```
------------------------------------------------------------------------------------

    function Corresponding_Body (Declaration : in Asis.Declaration)
                                return Asis.Declaration;

    function Corresponding_Body (Declaration : in Asis.Declaration;
                                 The_Context : in Asis.Context)
                                return Asis.Declaration;

------------------------------------------------------------------------------------
-- Declaration - Specifies the specification to query
-- The_Context - Specifies a Context to use
--
-- Returns the corresponding body for a given subprogram, package, or task
-- specification declaration. Returns the expanded generic body template for
-- generic instantiations.  The argument can be a Unit_Declaration from a
-- Compilation_Unit, or, it can be any appropriate specification declaration
-- from any declarative context.
--
-- These two function calls will always produce identical results:
--
--     Decl2 := Corresponding_Body (Decl1);
--     Decl2 := Corresponding_Body
--              (Decl1, Enclosing_Context ( Enclosing_Compilation_Unit( Decl1 )));
--
-- If a body declaration is given, the same element is returned.
--
-- Returns a Nil_Element if no body exists in The_Context.
--
-- The parameter The_Context is used to locate the corresponding specification
-- within a particular Context.  The_Context need not be the Enclosing_Context
-- of the Declaration.  Any non-Nil result will always have The_Context
-- as its Enclosing_Context.  This implies that while a non-Nil result may be
-- Is_Equal with the argument, it will only be Is_Identical if the
-- Enclosing_Context of the Declaration is the same as the parameter
-- The_Context.
--
-- Implicit predefined operations (e.g., "+", "=", etc.) will not typically
-- have unit bodies.  (Corresponding_Body returns a Nil_Element.)
-- User-defined overloads of the predefined operations will have
```

```
-- Corresponding_Body values once the bodies have inserted into the
-- environment.  The Corresponding_Body of an inherited subprogram is that
-- of the original user-defined subprogram.
--
-- If a generic instantiation is given, the body representing the expanded
-- generic body template is returned.  (i.e., an argument that is
-- A_Package_Instantiation, results in a value that is
-- A_Package_Body_Declaration that can be analyzed with all appropriate ASIS queries).
--
-- Returns a Nil_Element if the body of the generic has not yet been compiled
-- or inserted into the Ada Environment Context.
--
-- The Enclosing_Element of the expanded body is the generic instantiation. The
-- Enclosing_Compilation_Unit of the expanded template is that of the instantiation.
--
-- Returns Nil_Element for an implicit generic child unit specification.
-- Reference Manual 10.1.1(19).
--
-- Returns A_Pragma if the Declaration is completed by pragma Import.
--
-- Appropriate Declaration_Kinds returning a body:
--      A_Function_Declaration
--      A_Function_Instantiation
--      A_Generic_Package_Declaration
--      A_Generic_Procedure_Declaration
--      A_Generic_Function_Declaration
--      A_Package_Declaration
--      A_Package_Instantiation
--      A_Procedure_Declaration
--      A_Procedure_Instantiation
--      A_Single_Task_Declaration
--      A_Task_Type_Declaration
--      A_Protected_Type_Declaration
--      A_Single_Protected_Declaration
--      A_Formal_Package_Declaration
--      A_Formal_Package_Declaration_With_Box
--      An_Entry_Declaration (restricted to protected entry)
--
-- Appropriate Declaration_Kinds returning the argument Declaration:
--      A_Function_Body_Declaration
--      A_Function_Body_Stub
--      A_Function_Renaming_Declaration
--      A_Package_Body_Declaration
--      A_Package_Body_Stub
--      A_Package_Renaming_Declaration
--      A_Procedure_Body_Declaration
--      A_Procedure_Renaming_Declaration
--      A_Procedure_Body_Stub
--      A_Task_Body_Declaration
--      A_Task_Body_Stub
--      A_Protected_Body_Declaration
--      A_Protected_Body_Stub
--      A_Generic_Package_Renaming_Declaration
--      A_Generic_Procedure_Renaming_Declaration
--      A_Generic_Function_Renaming_Declaration
--      An_Entry_Body_Declaration
--
-- Returns Declaration_Kinds:
--      Not_A_Declaration
--      A_Function_Body_Declaration
--      A_Function_Body_Stub
--      A_Function_Renaming_Declaration
--      A_Package_Body_Declaration
--      A_Package_Body_Stub
--      A_Procedure_Body_Declaration
--      A_Procedure_Renaming_Declaration
--      A_Procedure_Body_Stub
--      A_Task_Body_Declaration
--      A_Task_Body_Stub
--      A_Protected_Body_Declaration
--      A_Protected_Body_Stub
--      An_Entry_Body_Declaration
--
```

```
-- Returns Element_Kinds:
--      Not_An_Element
--      A_Declaration
--      A_Pragma
--
-----------------------------------------------------------------------------------------
```

-- 15.28 function Corresponding_Subprogram_Derivation

```
-----------------------------------------------------------------------------------------

    function Corresponding_Subprogram_Derivation
              (Declaration : in Asis.Declaration)
                  return Asis.Declaration;

-----------------------------------------------------------------------------------------
-- Declaration - Specifies an implicit inherited subprogram declaration
--
-- Returns the subprogram declaration from which the given implicit inherited
-- subprogram argument was inherited.  The result can itself be an implicitly
-- inherited subprogram.
--
-- Appropriate Element_Kinds:
--      A_Declaration
--
-- Appropriate Declaration_Kinds:
--      A_Function_Declaration
--      A_Procedure_Declaration
--
-- Returns Element_Kinds:
--      A_Declaration
--
-- Returns Declaration_Kinds:
--      A_Function_Body_Declaration
--      A_Function_Declaration
--      A_Function_Renaming_Declaration
--      A_Procedure_Body_Declaration
--      A_Procedure_Declaration
--      A_Procedure_Renaming_Declaration
--
-- Raises ASIS_Inappropriate_Element for a subprogram declaration that is not
-- Is_Part_Of_Inherited.
--
-----------------------------------------------------------------------------------------
```

-- 15.29 function Corresponding_Type

```
-----------------------------------------------------------------------------------------

    function Corresponding_Type (Declaration : in Asis.Declaration)
                                return Asis.Type_Definition;

-----------------------------------------------------------------------------------------
-- Declaration - Specifies the subprogram_declaration to query
--
-- Returns the type definition for which this entity is an implicit
-- declaration.  The result will often be a derived type.  However, this query
-- also works for declarations of predefined operators such as "+" and "=".
-- Raises ASIS_Inappropriate_Element if the argument is not an implicit
-- declaration resulting from the declaration of a type.
--
-- Appropriate Element_Kinds:
--      A_Declaration
--
-- Appropriate Declaration_Kinds:
--      A_Function_Declaration
--      A_Procedure_Declaration
--
-- Returns Definition_Kinds:
--      A_Type_Definition
--      A_Formal_Type_Definition
--
```

```
-------------------------------------------------------------------------------
```

-- 15.30 function Corresponding_Equality_Operator

```
-------------------------------------------------------------------------------

    function Corresponding_Equality_Operator
             (Declaration : in Asis.Declaration) return Asis.Declaration;

-------------------------------------------------------------------------------
-- Declaration - Specifies an equality or an inequality operator declaration
--
-- If given an explicit Declaration of "=" whose result type is Boolean:
--
-- - Returns the complimentary implicit "/=" operator declaration.
--
-- - Returns a Nil_Element if the Ada implementation has not defined an
--   implicit "/=" for the "=".  Implementations of this sort will transform
--   a A/=B expression into a NOT(A=B) expression.  The function call
--   representing the NOT operation is Is_Part_Of_Implicit in this case.
--
-- If given an implicit Declaration of "/=" whose result type is Boolean:
--
-- - Returns the complimentary explicit "=" operator declaration.
--
-- Returns a Nil_Element for any other function declaration.
--
-- Appropriate Declaration_Kinds:
--      A_Function_Declaration
--
-- Returns Declaration_Kinds:
--      A_Function_Declaration
--
--|ER--------------------------------------------------------------------------
--|ER A_Package_Declaration - 7.1
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Visible_Part_Declarative_Items
--|CR    function Private_Part_Declarative_Items
--
-------------------------------------------------------------------------------
```

-- 15.31 function Visible_Part_Declarative_Items

```
-------------------------------------------------------------------------------

    function Visible_Part_Declarative_Items
                (Declaration     : in Asis.Declaration;
                 Include_Pragmas : in Boolean := False)
                return Asis.Declarative_Item_List;

-------------------------------------------------------------------------------
-- Declaration     - Specifies the package to query
-- Include_Pragmas - Specifies whether pragmas are to be returned
--
-- Returns a list of all basic declarations, representation specifications,
-- use clauses, and pragmas in the visible part of a package, in their order
-- of appearance.
--
-- Results of this query may vary across ASIS implementations.  Some
-- implementations normalize all multi-name object declarations into an
-- equivalent sequence of corresponding single name object declarations.
-- See Reference Manual 3.3.1(7).
--
-- Appropriate Declaration_Kinds:
--      A_Generic_Package_Declaration
--      A_Package_Declaration
--
-- Returns Element_Kinds:
--      A_Declaration
--      A_Pragma
--      A_Clause
--
```

```
-----------------------------------------------------------------------------
```
-- 15.32 function Is_Private_Present
```
-----------------------------------------------------------------------------

    function Is_Private_Present
                (Declaration : in Asis.Declaration) return Boolean;

-----------------------------------------------------------------------------
-- Declaration - Specifies the declaration to query
--
-- Returns True if the argument is a package specification which has a reserved
-- word "private" which marks the beginning of a (possibly empty) private part.
--
-- Returns False for any package specification without a private part.
-- Returns False for any unexpected Element.
--
-- Expected Element_Kinds:
--      A_Declaration
--
-- Expected Declaration_Kinds:
--      A_Generic_Package_Declaration
--      A_Package_Declaration
--
-----------------------------------------------------------------------------
```
-- 15.33 function Private_Part_Declarative_Items
```
-----------------------------------------------------------------------------

    function Private_Part_Declarative_Items
                (Declaration     : in Asis.Declaration;
                 Include_Pragmas : in Boolean := False)
                return Asis.Declarative_Item_List;

-----------------------------------------------------------------------------
-- Declaration     - Specifies the package to query
-- Include_Pragmas - Specifies whether pragmas are to be returned
--
-- Returns a list of all basic declarations, representation specifications,
-- use clauses, and pragmas in the private part of a package in their order of
-- appearance.
--
-- Results of this query may vary across ASIS implementations.  Some
-- implementations normalize all multi-name object declarations into an
-- equivalent sequence of corresponding single name object declarations.
-- See Reference Manual 3.3.1(7).
--
-- Appropriate Declaration_Kinds:
--      A_Generic_Package_Declaration
--      A_Package_Declaration
--
-- Returns Element_Kinds:
--      A_Declaration
--      A_Pragma
--      A_Clause
--
--|ER-------------------------------------------------------------------------
--|ER A_Package_Body_Declaration - 7.2
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Body_Declarative_Items
--|CR    function Body_Statements
--|CR    function Body_Exception_Handlers
--|CR    function Body_Block_Statement    - obsolescent, not recommended
--|ER-------------------------------------------------------------------------
--|ER A_Private_Type_Declaration      - 7.3
--|ER A_Private_Extension_Declaration - 7.3
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Discriminant_Part
--|CR    function Type_Declaration_View
--|ER-------------------------------------------------------------------------
--|ER An_Object_Renaming_Declaration - 8.5.1
```

```
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Declaration_Subtype_Mark
--|CR    function Renamed_Entity
--
-------------------------------------------------------------------------------
```

-- 15.34 function Renamed_Entity

```
-------------------------------------------------------------------------------

    function Renamed_Entity (Declaration : in Asis.Declaration)
                            return Asis.Expression;

-------------------------------------------------------------------------------
-- Declaration - Specifies the rename declaration to query
--
-- Returns the name expression that follows the reserved word "renames" in the
-- renaming declaration.
--
-- Appropriate Declaration_Kinds:
--      An_Exception_Renaming_Declaration
--      A_Function_Renaming_Declaration
--      An_Object_Renaming_Declaration
--      A_Package_Renaming_Declaration
--      A_Procedure_Renaming_Declaration
--      A_Generic_Package_Renaming_Declaration
--      A_Generic_Procedure_Renaming_Declaration
--      A_Generic_Function_Renaming_Declaration
--
-- Returns Element_Kinds:
--      An_Expression
--
--|ER---------------------------------------------------------------------------
--|ER An_Exception_Renaming_Declaration - 8.5.2
--|ER A_Package_Renaming_Declaration    - 8.5.3
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Renamed_Entity
--|ER---------------------------------------------------------------------------
--|ER A_Procedure_Renaming_Declaration - 8.5.4
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Parameter_Profile
--|CR    function Renamed_Entity
--|ER---------------------------------------------------------------------------
--|ER A_Function_Renaming_Declaration  - 8.5.4
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Parameter_Profile
--|CR    function Result_Profile
--|CR    function Renamed_Entity
--|ER---------------------------------------------------------------------------
--|ER A_Generic_Package_Renaming_Declaration   - 8.5.5
--|ER A_Generic_Procedure_Renaming_Declaration - 8.5.5
--|ER A_Generic_Function_Renaming_Declaration  - 8.5.5
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Renamed_Entity
--
-------------------------------------------------------------------------------
```

-- 15.35 function Corresponding_Base_Entity

```
-------------------------------------------------------------------------------

    function Corresponding_Base_Entity (Declaration : in Asis.Declaration)
                                 return Asis.Expression;

-------------------------------------------------------------------------------
-- Declaration - Specifies the rename declaration to query
--
```

```
-- The base entity is defined to be the renamed entity that is not itself
-- defined by another renaming declaration.
--
-- If the name following the reserved word “renames” is itself declared
-- by a previous renaming_declaration, then this query unwinds the renamings
-- by recursively operating on the previous renaming_declaration.
--
-- Otherwise, the name following the reserved word “renames” is returned.
--
-- Appropriate Declaration_Kinds:
--      An_Object_Renaming_Declaration
--      An_Exception_Renaming_Declaration
--      A_Procedure_Renaming_Declaration
--      A_Function_Renaming_Declaration
--      A_Package_Renaming_Declaration
--      A_Generic_Package_Renaming_Declaration
--      A_Generic_Procedure_Renaming_Declaration
--      A_Generic_Function_Renaming_Declaration
--
-- Returns Element_Kinds:
--      An_Expression
--
--|ER----------------------------------------------------------------------------
--|ER A_Task_Type_Declaration - 9.1
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Discriminant_Part
--|CR    function Type_Declaration_View
--|ER----------------------------------------------------------------------------
--|ER A_Single_Task_Declaration - 9.1
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Object_Declaration_View
--|ER----------------------------------------------------------------------------
--|ER A_Task_Body_Declaration - 9.1
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Body_Declarative_Items
--|CR    function Body_Statements
--|CR    function Body_Exception_Handlers
--|CR    function Body_Block_Statement    - obsolescent, not recommended
--|ER----------------------------------------------------------------------------
--|ER A_Protected_Type_Declaration - 9.4
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Discriminant_Part
--|CR    function Type_Declaration_View
--|ER----------------------------------------------------------------------------
--|ER A_Single_Protected_Declaration - 9.4
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Object_Declaration_View
--|ER----------------------------------------------------------------------------
--|ER A_Protected_Body_Declaration - 9.4
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Protected_Operation_Items
--
```

-- 15.36 function Protected_Operation_Items

```
    function Protected_Operation_Items
               (Declaration     : in Asis.Declaration;
                   Include_Pragmas : in Boolean := False)
                   return Asis.Declaration_List;
```

```
-----------------------------------------------------------------------------
-- Declaration     - Specifies the protected_body declaration to query
-- Include_Pragmas - Specifies whether pragmas are to be returned
--
-- Returns a list of protected_operation_item and pragma elements of the
-- protected_body, in order of appearance.
--
-- Returns a Nil_Element_List if there are no items or pragmas.
--
-- Appropriate Declaration_Kinds:
--      A_Protected_Body_Declaration
--
-- Returns Element_Kinds:
--      A_Pragma
--      A_Declaration
--      A_Clause
--
-- Returns Declaration_Kinds:
--      A_Procedure_Declaration
--      A_Function_Declaration
--      A_Procedure_Body_Declaration
--      A_Function_Body_Declaration
--      An_Entry_Body_Declaration
--
-- Returns Clause_Kinds:
--      A_Representation_Clause
--
--|ER-------------------------------------------------------------------------
--|ER An_Entry_Declaration - 9.5.2
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Entry_Family_Definition
--|CR    function Parameter_Profile
--
-----------------------------------------------------------------------------
```

-- 15.37 function Entry_Family_Definition

```
-----------------------------------------------------------------------------

    function Entry_Family_Definition (Declaration : in Asis.Declaration)
                                     return Asis.Discrete_Subtype_Definition;

-----------------------------------------------------------------------------
-- Declaration - Specifies the entry declaration to query
--
-- Returns the Discrete_Subtype_Definition element for the entry family of
-- an entry_declaration.
--
-- Returns a Nil_Element if the entry_declaration does not define a family
-- of entries.
--
-- Appropriate Declaration_Kinds:
--      An_Entry_Declaration
--
-- Returns Definition_Kinds:
--      Not_A_Definition
--      A_Discrete_Subtype_Definition
--
--|ER-------------------------------------------------------------------------
--|ER An_Entry_Body_Declaration - 9.5.2
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Entry_Index_Specification
--|CR    function Parameter_Profile
--|CR    function Entry_Barrier
--|CR    function Body_Declarative_Items
--|CR    function Body_Statements
--|CR    function Body_Exception_Handlers
--|CR    function Body_Block_Statement    - obsolescent, not recommended
--
```

```
-------------------------------------------------------------------------------
```
-- 15.38 function Entry_Index_Specification
```
-------------------------------------------------------------------------------

    function Entry_Index_Specification (Declaration : in Asis.Declaration)
                                   return Asis.Declaration;

-------------------------------------------------------------------------------
-- Declaration - Specifies the entry body declaration to query
--
-- Returns the An_Entry_Index_Specification element of an entry body
-- declaration.
--
-- Returns a Nil_Element if the entry does not declare any
-- An_Entry_Index_Specification element.
--
-- Appropriate Declaration_Kinds:
--      An_Entry_Body_Declaration
--
-- Returns Declaration_Kinds:
--      Not_A_Declaration
--      An_Entry_Index_Specification
--
-------------------------------------------------------------------------------
```
-- 15.39 function Entry_Barrier
```
-------------------------------------------------------------------------------

    function Entry_Barrier (Declaration : in Asis.Declaration)
                       return Asis.Expression;

-------------------------------------------------------------------------------
-- Declaration - Specifies the entry body declaration to query
--
-- Returns the expression following the reserved word "when" in an entry body declaration.
--
-- Appropriate Declaration_Kinds:
--      An_Entry_Body_Declaration
--
-- Returns Element_Kinds:
--      An_Expression
--
--|ER---------------------------------------------------------------------------
--|ER A_Procedure_Body_Stub - 10.1.3
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Parameter_Profile
--|ER---------------------------------------------------------------------------
--|ER A_Function_Body_Stub - 10.1.3
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Parameter_Profile
--|CR    function Result_Profile
--|ER---------------------------------------------------------------------------
--|ER A_Package_Body_Stub   - 10.1.3
--|ER A_Task_Body_Stub      - 10.1.3
--|ER A_Protected_Body_Stub - 10.1.3
--|CR
--|CR Child elements returned by:
--|CR    function Names
--
-------------------------------------------------------------------------------
```
-- 15.40 function Corresponding_Subunit
```
-------------------------------------------------------------------------------

    function Corresponding_Subunit (Body_Stub : in Asis.Declaration)
                 return Asis.Declaration;

    function Corresponding_Subunit (Body_Stub   : in Asis.Declaration;
                                    The_Context : in Asis.Context)
                  return Asis.Declaration;
```

```
-------------------------------------------------------------------------------
-- Body_Stub   - Specifies the stub to query
-- The_Context - Specifies a Context to use to locate the subunit
--
-- Returns the Unit_Declaration of the subunit compilation unit corresponding
-- to the body stub.
--
-- Returns a Nil_Element if the subunit does not exist in The_Context.
--
-- These two function calls will always produce identical results:
--
--     Decl2 := Corresponding_Subunit (Decl1);
--     Decl2 := Corresponding_Subunit
--              (Decl1, Enclosing_Context ( Enclosing_Compilation_Unit( Decl1 )));
--
-- The parameter The_Context is used to locate the corresponding subunit body.
-- Any non-Nil result will always have The_Context as its Enclosing_Context.
--
-- Appropriate Declaration_Kinds:
--      A_Function_Body_Stub
--      A_Package_Body_Stub
--      A_Procedure_Body_Stub
--      A_Task_Body_Stub
--      A_Protected_Body_Stub
--
-- Returns Declaration_Kinds:
--      Not_A_Declaration
--      A_Function_Body_Declaration
--      A_Package_Body_Declaration
--      A_Procedure_Body_Declaration
--      A_Task_Body_Declaration
--      A_Protected_Body_Declaration
--
-------------------------------------------------------------------------------
```

-- 15.41 function Is_Subunit

```
-------------------------------------------------------------------------------

    function Is_Subunit (Declaration : in Asis.Declaration) return Boolean;

-------------------------------------------------------------------------------
-- Declaration - Specifies the declaration to query
--
-- Returns True if the declaration is the proper_body of a subunit.
--
-- Returns False for any unexpected Element.
--
-- Equivalent to:
--
--     Declaration = Unit_Declaration(Enclosing_Compilation_Unit (Declaration))
--     and Unit_Kind(Enclosing_Compilation_Unit (Declaration)) in A_Subunit.
--
-- Expected Declaration_Kinds:
--      A_Procedure_Body_Declaration
--      A_Function_Body_Declaration
--      A_Package_Body_Declaration
--      A_Task_Body_Declaration
--      A_Protected_Body_Declaration
--
-------------------------------------------------------------------------------
```

-- 15.42 function Corresponding_Body_Stub

```
-------------------------------------------------------------------------------

    function Corresponding_Body_Stub (Subunit : in Asis.Declaration)
                      return Asis.Declaration;

    function Corresponding_Body_Stub (Subunit     : in Asis.Declaration;
                                      The_Context : in Asis.Context)
                      return Asis.Declaration;

-------------------------------------------------------------------------------
-- Subunit     - Specifies the Is_Subunit declaration to query
-- The_Context - Specifies a Context to use to locate the parent unit
```

```
--
-- Returns the body stub declaration located in the subunit's parent unit.
--
-- Returns a Nil_Element if the parent unit does not exist in The_Context.
--
-- These two function calls will always produce identical results:
--
--     Decl2 := Corresponding_Body_Stub (Decl1);
--     Decl2 := Corresponding_Body_Stub
--              (Decl1, Enclosing_Context ( Enclosing_Compilation_Unit( Decl1 )));
--
-- The parameter The_Context is used to locate the corresponding parent body.
-- Any non-Nil result will always have The_Context as its Enclosing_Context.
--
-- Appropriate Declaration Kinds:
--      (Is_Subunit(Declaration) shall also be True)
--      A_Function_Body_Declaration
--      A_Package_Body_Declaration
--      A_Procedure_Body_Declaration
--      A_Task_Body_Declaration
--      A_Protected_Body_Declaration
--
-- Returns Declaration_Kinds:
--      Not_A_Declaration
--      A_Function_Body_Stub
--      A_Package_Body_Stub
--      A_Procedure_Body_Stub
--      A_Task_Body_Stub
--      A_Protected_Body_Stub
--
--|ER---------------------------------------------------------------------------------
--|ER An_Exception_Declaration - 11.1
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|ER---------------------------------------------------------------------------------
--|ER A_Choice_Parameter_Specification - 11.2
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|ER---------------------------------------------------------------------------------
--|ER A_Generic_Procedure_Declaration - 12.1
--|CR
--|CR Child elements returned by:
--|CR    function Generic_Formal_Part
--|CR    function Names
--|CR    function Parameter_Profile
--
------------------------------------------------------------------------------------
```

-- 15.43 function Generic_Formal_Part

```
------------------------------------------------------------------------------------

    function Generic_Formal_Part
                (Declaration     : in Asis.Declaration;
                 Include_Pragmas : in Boolean := False)
                return Asis.Element_List;

------------------------------------------------------------------------------------
-- Declaration     - Specifies the generic declaration to query
-- Include_Pragmas - Specifies whether pragmas are to be returned
--
-- Returns a list of generic formal parameter declarations, use clauses,
-- and pragmas, in their order of appearance.
--
-- Results of this query may vary across ASIS implementations.  Some
-- implementations normalize all multi-name object declarations into an
-- equivalent sequence of corresponding single name object declarations.
-- See Reference Manual 3.3.1(7).
--
-- Appropriate Declaration_Kinds:
--      A_Generic_Package_Declaration
--      A_Generic_Procedure_Declaration
--      A_Generic_Function_Declaration
```

```
--
-- Returns Element_Kinds:
--      A_Pragma
--      A_Declaration
--      A_Clause
--
-- Returns Declaration_Kinds:
--      A_Formal_Object_Declaration
--      A_Formal_Type_Declaration
--      A_Formal_Procedure_Declaration
--      A_Formal_Function_Declaration
--      A_Formal_Package_Declaration
--      A_Formal_Package_Declaration_With_Box
--
-- Returns Clause_Kinds:
--      A_Use_Package_Clause
--      A_Use_Type_Clause
--
--|ER----------------------------------------------------------------------------
--|ER A_Generic_Function_Declaration - 12.1
--|CR
--|CR Child elements returned by:
--|CR    function Generic_Formal_Part
--|CR    function Names
--|CR    function Parameter_Profile
--|CR    function Result_Profile
--|ER----------------------------------------------------------------------------
--|ER A_Generic_Package_Declaration - 12.1
--|CR
--|CR Child elements returned by:
--|CR    function Generic_Formal_Part
--|CR    function Names
--|CR    function Visible_Part_Declarative_Items
--|CR    function Private_Part_Declarative_Items
--|ER----------------------------------------------------------------------------
--|ER A_Package_Instantiation   - 12.3
--|ER A_Procedure_Instantiation - 12.3
--|ER A_Function_Instantiation  - 12.3
--|CR
--|CR Child elements returned by:
--|CR    function Names
--|CR    function Generic_Unit_Name
--|CR    function Generic_Actual_Part
-------------------------------------------------------------------------------
--
-- Instantiations can always be analyzed in terms of the generic actual
-- parameters supplied with the instantiation.  A generic instance is a copy
-- of the generic unit, and while there is no explicit (textual) specification
-- in the program text, an implicit specification and body, if there is one,
-- with the generic actual parameters is implied.
--
-- To analyze the implicit instance specification or body of a generic
-- instantiation:
--
-- - Use Corresponding_Declaration to return the implicit expanded
--   specification of an instantiation.
--
-- - Use Corresponding_Body to return the implicit body of an instantiation.
--
-- - Then analyze the specification or body with any appropriate queries.
--
-- To analyze the explicit generic specification or body referenced by a
-- generic instantiation:
--
-- - Use Generic_Unit_Name to obtain the name of the generic unit.
--
-- - Then use Corresponding_Name_Declaration to get to the generic declaration.
--
-- - Then use Corresponding_Body to get to the body of the generic declaration.
--
```

```
-------------------------------------------------------------------------------
```
-- 15.44 function Generic_Unit_Name
```
-------------------------------------------------------------------------------

    function Generic_Unit_Name (Declaration : in Asis.Declaration)
                               return Asis.Expression;

-------------------------------------------------------------------------------
-- Declaration - Specifies the generic instantiation to query
--
-- Returns the name following the reserved word "new” in the generic
-- instantiation.  The name denotes the generic package, generic procedure, or
-- generic function that is the template for this generic instance.
--
-- Appropriate Declaration_Kinds:
--      A_Function_Instantiation
--      A_Package_Instantiation
--      A_Procedure_Instantiation
--      A_Formal_Package_Declaration
--      A_Formal_Package_Declaration_With_Box
--
-- Returns Expression_Kinds:
--      An_Identifier
--      An_Operator_Symbol
--      A_Selected_Component
--
-------------------------------------------------------------------------------
```
-- 15.45 function Generic_Actual_Part
```
-------------------------------------------------------------------------------

    function Generic_Actual_Part (Declaration : in Asis.Declaration;
                                  Normalized  : in Boolean := False)
                                 return Asis.Association_List;

-------------------------------------------------------------------------------
-- Declaration - Specifies the generic_instantiation to query
-- Normalized  - Specifies whether the normalized form is desired
--
-- Returns a list of the generic_association elements of the instantiation.
--
-- Returns a Nil_Element_List if there are no generic_association elements.
--
-- An unnormalized list contains only explicit associations ordered as they
-- appear in the program text.  Each unnormalized association has an optional
-- generic_formal_parameter_selector_name and an
-- explicit_generic_actual_parameter component.
--
-- A normalized list contains artificial associations representing all
-- explicit and default associations.  It has a length equal to the number of
-- generic_formal_parameter_declaration elements of the generic_formal_part of the
-- template.  The order of normalized associations matches the order of the
-- generic_formal_parameter_declaration elements.
--
-- Each normalized association represents a one-on-one mapping of a
-- generic_formal_parameter_declaration to the explicit or default expression
-- or name.  A normalized association has:
--
-- - one A_Defining_Name component that denotes the
--   generic_formal_parameter_declaration, and
--
-- - one An_Expression component that is either:
--     the explicit_generic_actual_parameter,
--     a default_expression, or
--     a default_name from the generic_formal_parameter_declaration or
--        an implicit naming expression which denotes the actual subprogram
--        selected at the place of instantiation for a formal subprogram
--        having A_Box_Default.
--
-- Appropriate Declaration_Kinds:
--      A_Function_Instantiation
--      A_Package_Instantiation
--      A_Procedure_Instantiation
--      A_Formal_Package_Declaration
```

```
--
-- Returns Association_Kinds:
--      A_Generic_Association
--
--|IR Implementation Requirements:
--|IR
--|IR Normalized associations are Is_Normalized and Is_Part_Of_Implicit.
--|IR Normalized associations provided by default are Is_Defaulted_Association.
--|IR Normalized associations are never Is_Equal to unnormalized associations.
--
--|IP Implementation Permissions:
--|IP
--|IP An implementation may choose to always include default parameters in its
--|IP internal representation.
--|IP
--|IP An implementation may also choose to normalize its representation
--|IP to use the defining_identifier element rather than the
--|IP generic_formal_parameter_selector_name elements.
--|IP
--|IP In either case, this query will return Is_Normalized associations even if
--|IP Normalized is False, and the query Generic_Actual_Part_Normalized will
--|IP return True.
--
--|ER-----------------------------------------------------------------------------
--|ER A_Formal_Object_Declaration - 12.4
--|CR
--|CR Child elements returned by:
--|CR    functions Names, Declaration_Subtype_Mark, Initialization_Expression
--|ER-----------------------------------------------------------------------------
--|ER A_Formal_Type_Declaration - 12.5
--|CR
--|CR Child elements returned by:
--|CR    functions Names, Discriminant_Part, Type_Declaration_View
--|ER-----------------------------------------------------------------------------
--|ER A_Formal_Procedure_Declaration - 12.6
--|CR
--|CR Child elements returned by:
--|CR    functions Names, function Parameter_Profile, function Formal_Subprogram_Default
--
---------------------------------------------------------------------------------
```

-- 15.46 function Formal_Subprogram_Default

```
---------------------------------------------------------------------------------

    function Formal_Subprogram_Default
                (Declaration : in Asis.Generic_Formal_Parameter)
                return Asis.Expression;

---------------------------------------------------------------------------------
-- Declaration - Specifies the generic formal subprogram declaration to query
--
-- Returns the name appearing after the reserved word "is" in the given generic
-- formal subprogram declaration.
--
-- Appropriate Declaration_Kinds:
--      A_Formal_Function_Declaration
--      A_Formal_Procedure_Declaration
--
-- Appropriate Subprogram_Default_Kinds:
--      A_Name_Default
--
-- Returns Element_Kinds:
--      An_Expression
--
--|ER-----------------------------------------------------------------------------
--|ER A_Formal_Function_Declaration - 12.6
--|CR
--|CR Child elements returned by:
--|CR    functions Names, function Parameter_Profile, Result_Profile,
--|CR    and Formal_Subprogram_Default
```

```
--|ER----------------------------------------------------------------------------
--|ER A_Formal_Package_Declaration - 12.7
--|CR
--|CR Child elements returned by:
--|CR    functions Names, Generic_Unit_Name, and Generic_Actual_Part
--|ER----------------------------------------------------------------------------
--|ER A_Formal_Package_Declaration_With_Box - 12.7
--|CR
--|CR Child elements returned by:
--|CR    functions Names and Generic_Unit_Name
--
-------------------------------------------------------------------------------
```

-- 15.47 function Corresponding_Generic_Element

```
-------------------------------------------------------------------------------

    function Corresponding_Generic_Element (Reference : in Asis.Element)
                                           return Asis.Defining_Name;

-------------------------------------------------------------------------------
-- Reference   - Specifies an expression that references an entity declared
--               within the implicit specification of a generic instantiation,
--               or, specifies the defining name of such an entity.
--
-- Given a reference to some implicit entity, whose declaration occurs within
-- an implicit generic instance, returns the corresponding entity name
-- definition from the generic template used to create the generic instance.
-- (Reference Manual 12.3 (16))
--
-- Returns the first A_Defining_Name, from the generic template, that
-- corresponds to the entity referenced.
--
-- Returns a Nil_Element if the argument does not refer to an entity declared
-- as a component of a generic package instantiation.  The entity name can
-- refer to an ordinary declaration, an inherited subprogram declaration, or a
-- predefined operator declaration.
--
-- Appropriate Element_Kinds:
--      A_Defining_Name
--      An_Expression
--
-- Appropriate Expression_Kinds:
--      An_Identifier
--      An_Operator_Symbol
--      A_Character_Literal
--      An_Enumeration_Literal
--
-- Returns Element_Kinds:
--      Not_An_Element
--      A_Defining_Name
--
-------------------------------------------------------------------------------
```

-- 15.48 function Is_Dispatching_Operation

```
-------------------------------------------------------------------------------

    function Is_Dispatching_Operation (Declaration : in Asis.Element)
                                      return Boolean;

-------------------------------------------------------------------------------
-- Declaration   -  Specifies the declaration to query.
--
-- Returns True if the declaration is a primitive subprogram of a tagged type.
--
-- Returns False for any unexpected argument.
--
-- Expected Element_Kinds:
--      A_Procedure_Declaration
--      A_Function_Declaration
--      A_Procedure_Renaming_Declaration
--      A_Function_Renaming_Declaration
--
-------------------------------------------------------------------------------

end Asis.Declarations;
```

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

-- 16 package Asis.Definitions

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
package Asis.Definitions is
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- Asis.Definitions encapsulates a set of queries that operate on A_Definition
-- and An_Association elements.
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

-- 16.1 function Corresponding_Type_Operators

```
-------------------------------------------------------------------------------
--|ER A_Type_Definition - 3.2.1
-------------------------------------------------------------------------------

    function Corresponding_Type_Operators
            (Type_Definition : in Asis.Type_Definition)
                return Asis.Declaration_List;

-------------------------------------------------------------------------------
-- Type_Definition - Specifies the type to query
--
-- Returns a list of operators.  These include all predefined operators, and
-- all user-defined operator overloads, that have been implicitly or
-- explicitly declared for the type. (Reference Manual 7.3.1(2))
--
-- This list includes only operators appropriate for the type, from the set:
--      and or xor = /= < <= > >= + - & * / mod rem ** abs not
--
-- Returns a Nil_Element_List if there are no predefined or overloaded
-- operators for the type.
--
-- Returns a Nil_Element_List if the implementation does not provide
-- such implicit declarations.
--
-- The Enclosing_Element for each implicit declaration is the declaration (type
-- or object) that declared the type.
--
-- For limited private types, if a user-defined equality operator has
-- been defined, an Ada implementation has two choices when dealing with an
-- instance of the "/=" operator.  a) treat A/=B as NOT(A=B), b) implicitly
-- create a "/=" operator.  Implementations that take the second alternative
-- will include this implicit inequality operation in their result.
-- Implementations that choose the first alternative are encouraged to hide
-- this choice beneath the ASIS interface and to "fake" an inequality
-- operation.  Failing that, the function call, representing the NOT
-- operation, must have Is_Part_Of_Implicit = True so that an ASIS application
-- can tell the  difference between a user-specified NOT(A=B) and an
-- implementation-specific A/=B transformation.
--
-- Appropriate Definition_Kinds:
--      A_Type_Definition
--      A_Formal_Type_Definition
--
-- Returns Declaration_Kinds:
--      A_Function_Declaration
--      A_Function_Body_Declaration
--      A_Function_Body_Stub
--      A_Function_Renaming_Declaration
--      A_Function_Instantiation
--      A_Formal_Function_Declaration
--
--|IP Implementation Permissions:
--|IP
--|IP The result may or may not include language defined operators that have
--|IP been overridden by user-defined overloads.  Operators that are totally
--|IP hidden, in all contexts, by user-defined operators may be omitted from
--|IP the list.
--|IP
--|IP Some implementations do not represent all forms of implicit
```

```
--|IP declarations such that elements representing them can be easily
--|IP provided.  An implementation can choose whether or not to construct
--|IP and provide artificial declarations for implicitly declared elements.
--
--|ER-------------------------------------------------------------------------
--|ER A_Derived_Type_Definition - 3.4
--|CR
--|CR Child elements returned by:
--|CR    function Parent_Subtype_Indication
--|ER-------------------------------------------------------------------------
--|ER A_Derived_Record_Extension_Definition - 3.4
--|CR
--|CR Child elements returned by:
--|CR    function Parent_Subtype_Indication
--|CR    function Record_Definition
--
-----------------------------------------------------------------------------
```

-- 16.2 function Parent_Subtype_Indication

```
-----------------------------------------------------------------------------

    function Parent_Subtype_Indication
                  (Type_Definition : in Asis.Type_Definition)
                        return Asis.Subtype_Indication;

-----------------------------------------------------------------------------
-- Type_Definition - Specifies the derived_type_definition to query
--
-- Returns the parent_subtype_indication following the reserved word "new".
--
-- Appropriate Type_Kinds:
--      A_Derived_Type_Definition
--      A_Derived_Record_Extension_Definition
--
-- Returns Definition_Kinds:
--      A_Subtype_Indication
--
-----------------------------------------------------------------------------
```

-- 16.3 function Record_Definition

```
-----------------------------------------------------------------------------

    function Record_Definition (Type_Definition : in Asis.Type_Definition)
                               return Asis.Definition;

-----------------------------------------------------------------------------
-- Type_Definition - Specifies the definition to query
--
-- Returns the record definition of the type_definition.
--
-- Appropriate Type_Kinds:
--      A_Derived_Record_Extension_Definition
--      A_Record_Type_Definition
--      A_Tagged_Record_Type_Definition
--
-- Returns Definition_Kinds:
--      A_Record_Definition
--      A_Null_Record_Definition
--
-----------------------------------------------------------------------------
```

-- 16.4 function Implicit_Inherited_Declarations

```
-----------------------------------------------------------------------------

    function Implicit_Inherited_Declarations
                  (Definition : in Asis.Definition)
                      return Asis.Declaration_List;

-----------------------------------------------------------------------------
-- Definition - Specifies the derived type to query
--
-- Returns a list of Is_Part_Of_Implicit inherited enumeration literals,
-- discriminants, components, protected subprograms, or entries of a
-- derived_type_definition whose parent type is an enumeration type, or a
-- composite type other than an array type.  See Reference Manual 3.4(10-14).
```

```
--
-- Returns a Nil_Element_List if the root type of derived_type_definition is
-- not an enumeration, record, task, or protected type.
--
-- Returns a Nil_Element_List if the implementation does not provide
-- such implicit declarations.
--
-- The Enclosing_Element for each of the implicit declarations is the
-- Declaration argument.
--
-- Appropriate Definition_Kinds:
--      A_Type_Definition
--      A_Private_Extension_Definition
--      A_Formal_Type_Definition
--
-- Appropriate Type_Kinds:
--      A_Derived_Type_Definition
--      A_Derived_Record_Extension_Definition
--
-- Appropriate Formal_Type_Kinds:
--      A_Formal_Derived_Type_Definition
--
-- Returns Declaration_Kinds:
--
--      An_Enumeration_Literal_Specification
--      A_Discriminant_Specification
--      A_Component_Declaration
--      A_Procedure_Declaration
--      A_Function_Declaration
--      An_Entry_Declaration
--
--|IP Implementation Permissions:
--|IP
--|IP Some implementations do not represent all forms of implicit
--|IP declarations such that elements representing them can be easily
--|IP provided.  An implementation can choose whether or not to construct
--|IP and provide artificial declarations for implicitly declared elements.
--
--|AN Application Note:
--|AN
--|AN This query returns only implicit inherited entry declarations for
--|AN derived task types.  All representation clauses and pragmas associated
--|AN with the entries of the original task type (the root type of the
--|AN derived task type) apply to the inherited entries.  Those are available
--|AN by examining the original type or by calling Corresponding_Pragmas and
--|AN Corresponding_Representation_Clauses.  These functions will return the
--|AN pragmas and clauses from the original type.
--
-------------------------------------------------------------------------------------
```

-- 16.5 function Implicit_Inherited_Subprograms

```
-------------------------------------------------------------------------------------

    function Implicit_Inherited_Subprograms
                (Definition : in Asis.Definition)
                return Asis.Declaration_List;

-------------------------------------------------------------------------------------
-- Definition - Specifies the derived type to query
--
-- Returns the list of user-defined inherited primitive subprograms that have
-- been implicitly declared for the derived_type_definition.
--
-- The list result does not include hidden inherited subprograms (Reference Manual 8.3).
--
-- Returns a Nil_Element_List if there are no inherited subprograms for the
-- derived type.
--
-- Returns a Nil_Element_List if the implementation does not provide
-- such implicit declarations.
--
-- The Enclosing_Element for each of the subprogram declarations is the
-- Definition argument.
--
```

```
-- Appropriate Definition_Kinds:
--      A_Type_Definition
--      A_Private_Extension_Definition
--      A_Formal_Type_Definition
--
-- Appropriate Type_Kinds:
--      A_Derived_Type_Definition
--      A_Derived_Record_Extension_Definition
--
-- Appropriate Formal_Type_Kinds:
--      A_Formal_Derived_Type_Definition
--
-- Returns Declaration_Kinds:
--      A_Function_Declaration
--      A_Procedure_Declaration
--
--|IP Implementation Permissions:
--|IP
--|IP Some implementations do not represent all forms of implicit
--|IP declarations such that elements representing them can be easily
--|IP provided.  An implementation can choose whether or not to construct
--|IP and provide artificial declarations for implicitly declared elements.
--
-------------------------------------------------------------------------------------
```

-- 16.6 function Corresponding_Parent_Subtype

```
-------------------------------------------------------------------------------------

     function Corresponding_Parent_Subtype
                  (Type_Definition : in Asis.Type_Definition)
                      return Asis.Declaration;

-------------------------------------------------------------------------------------
-- Type_Definition - Specifies the derived_type_definition to query
--
-- Returns the parent subtype declaration of the derived_type_definition.
-- The parent subtype is defined by the parent_subtype_indication.
--
-- Appropriate Type_Kinds:
--      A_Derived_Type_Definition
--      A_Derived_Record_Extension_Definition
--
-- Returns Declaration_Kinds:
--      An_Ordinary_Type_Declaration
--      A_Task_Type_Declaration
--      A_Protected_Type_Declaration
--      A_Subtype_Declaration
--      A_Formal_Type_Declaration
--      An_Incomplete_Type_Declaration
--      A_Private_Type_Declaration
--      A_Private_Extension_Declaration
--
-------------------------------------------------------------------------------------
```

-- 16.7 function Corresponding_Root_Type

```
-------------------------------------------------------------------------------------

    function Corresponding_Root_Type
                     (Type_Definition : in Asis.Type_Definition)
                      return Asis.Declaration;

-------------------------------------------------------------------------------------
-- Type_Definition - Specifies the derived_type_definition to query
--
-- This function recursively unwinds all type derivations and subtyping to
-- arrive at a full_type_declaration that is neither a derived type nor a
-- subtype.
--
-- In case of numeric types, this function always returns some user-defined
-- type, not an implicitly defined root type corresponding to
-- A_Root_Type_Definition. The only ways to get implicitly declared numeric
-- root or universal types are to ask for the type of a universal expression
-- or from the parameter and result profile of a predefined operation working
-- with numeric types.
--
```

```
-- Appropriate Type_Kinds:
--      A_Derived_Type_Definition
--      A_Derived_Record_Extension_Definition
--
-- Returns Declaration_Kinds:
--      An_Ordinary_Type_Declaration
--      A_Task_Type_Declaration
--      A_Protected_Type_Declaration
--      A_Formal_Type_Declaration
--      A_Private_Type_Declaration
--      A_Private_Extension_Declaration
--
---------------------------------------------------------------------------------------
```

-- 16.8 function Corresponding_Type_Structure

```
---------------------------------------------------------------------------------------

    function Corresponding_Type_Structure
                  (Type_Definition : in Asis.Type_Definition)
                      return Asis.Declaration;

---------------------------------------------------------------------------------------
-- Type_Definition - Specifies the derived_type_definition to query
--
-- Returns the type structure from which the specified type definition has
-- been derived.  This function will recursively unwind derivations and
-- subtyping until the type_declaration derives a change of representation or
-- is no longer derived.  See Reference Manual 13.6.
--
-- Appropriate Type_Kinds:
--      A_Derived_Type_Definition
--      A_Derived_Record_Extension_Definition
--
-- Returns Declaration_Kinds:
--      An_Ordinary_Type_Declaration
--      A_Task_Type_Declaration
--      A_Protected_Type_Declaration
--      A_Formal_Type_Declaration
--      A_Private_Type_Declaration
--      A_Private_Extension_Declaration
--
--|ER------------------------------------------------------------------------------------
--|ER An_Enumeration_Type_Definition - 3.5.1
--|CR
--|CR Child elements returned by:
--|CR    function Enumeration_Literal_Declarations
--
---------------------------------------------------------------------------------------
```

-- 16.9 function Enumeration_Literal_Declarations

```
---------------------------------------------------------------------------------------

    function Enumeration_Literal_Declarations
                (Type_Definition : in Asis.Type_Definition)
                return Asis.Declaration_List;

---------------------------------------------------------------------------------------
-- Type_Definition - Specifies the enumeration type definition to query
--
-- Returns a list of the literals declared in an enumeration_type_definition,
-- in their order of appearance.
--
-- Appropriate Type_Kinds:
--      An_Enumeration_Type_Definition
--
-- Returns Declaration_Kinds:
--      An_Enumeration_Literal_Specification
--
--|ER------------------------------------------------------------------------------------
--|ER A_Signed_Integer_Type_Definition - 3.5.4
--|CR
--|CR Child elements returned by:
--|CR    function Integer_Constraint
--
```

```
-------------------------------------------------------------------------------------
```
-- 16.10 function Integer_Constraint
```
-------------------------------------------------------------------------------------

    function Integer_Constraint
                (Type_Definition : in Asis.Type_Definition)
                return Asis.Range_Constraint;

-------------------------------------------------------------------------------------
-- Type_Definition - Specifies the signed_integer_type_definition to query
--
-- Returns the range_constraint of the signed_integer_type_definition.
--
-- Appropriate Type_Kinds:
--      A_Signed_Integer_Type_Definition
--
-- Returns Constraint_Kinds:
--      A_Simple_Expression_Range
--
--|ER----------------------------------------------------------------------------------
--|ER A_Modular_Type_Definition - 3.5.4
--|CR
--|CR Child elements returned by:
--|CR    function Mod_Static_Expression
--
-------------------------------------------------------------------------------------
```
-- 16.11 function Mod_Static_Expression
```
-------------------------------------------------------------------------------------

    function Mod_Static_Expression
                (Type_Definition : in Asis.Type_Definition)
                return Asis.Expression;

-------------------------------------------------------------------------------------
-- Type_Definition - Specifies the modular_type_definition to query
--
-- Returns the static_expression following the reserved word "mod".
--
-- Appropriate Type_Kinds:
--      A_Modular_Type_Definition
--
-- Returns Element_Kinds:
--      An_Expression
--
--|ER----------------------------------------------------------------------------------
--|ER A_Floating_Point_Definition - 3.5.7
--|CR
--|CR Child elements returned by:
--|CR    functions Digits_Expression and Real_Range_Constraint
--
--|ER----------------------------------------------------------------------------------
--|ER A_Decimal_Fixed_Point_Definition - 3.5.9
--|CR
--|CR Child elements returned by:
--|CR    functions Digits_Expression, Delta_Expression, and Real_Range_Constraint
--
-------------------------------------------------------------------------------------
```
-- 16.12 function Digits_Expression
```
-------------------------------------------------------------------------------------

    function Digits_Expression (Definition : in Asis.Definition)
                               return Asis.Expression;

-------------------------------------------------------------------------------------
-- Definition  - Specifies the definition to query
--
-- Returns the static_expression following the reserved word "digits".
--
-- Appropriate Type_Kinds:
--      A_Floating_Point_Definition
--      A_Decimal_Fixed_Point_Definition
--
```

```
-- Appropriate Definition_Kinds:
--      A_Constraint
--          Appropriate Constraint_Kinds:
--               A_Digits_Constraint
--
-- Returns Element_Kinds:
--      An_Expression
--
--|ER---------------------------------------------------------------------------------
--|ER An_Ordinary_Fixed_Point_Definition - 3.5.9
--|CR
--|CR Child elements returned by:
--|CR    function Delta_Expression
--
------------------------------------------------------------------------------------
```

-- 16.13 function Delta_Expression

```
------------------------------------------------------------------------------------

    function Delta_Expression (Definition : in Asis.Definition)
                              return Asis.Expression;

------------------------------------------------------------------------------------
-- Definition  - Specifies the definition to query
--
-- Returns the static_expression following the reserved word "delta".
--
-- Appropriate Type_Kinds:
--      An_Ordinary_Fixed_Point_Definition
--      A_Decimal_Fixed_Point_Definition
--
-- Appropriate Definition_Kinds:
--      A_Constraint
--          Appropriate Constraint_Kinds:
--               A_Delta_Constraint
--
-- Returns Element_Kinds:
--      An_Expression
--
------------------------------------------------------------------------------------
```

-- 16.14 function Real_Range_Constraint

```
------------------------------------------------------------------------------------

    function Real_Range_Constraint
             (Definition : in Asis.Definition) return Asis.Range_Constraint;

------------------------------------------------------------------------------------
-- Definition  - Specifies the definition to query
--
-- Returns the real_range_specification range_constraint of the definition.
--
-- Returns a Nil_Element if there is no explicit range_constraint.
--
-- Appropriate Type_Kinds:
--      A_Floating_Point_Definition
--      An_Ordinary_Fixed_Point_Definition
--      A_Decimal_Fixed_Point_Definition
--
-- Appropriate Definition_Kinds:
--      A_Constraint
--          Appropriate Constraint_Kinds:
--               A_Digits_Constraint
--               A_Delta_Constraint
--
-- Returns Constraint_Kinds:
--      Not_A_Constraint
--      A_Simple_Expression_Range
--
--|ER---------------------------------------------------------------------------------
--|ER An_Unconstrained_Array_Definition 3.6
--|CR
--|CR Child elements returned by:
--|CR    functions Index_Subtype_Definitions and Array_Component_Definition
--
```

```
-----------------------------------------------------------------------------------
```
-- 16.15 function Index_Subtype_Definitions
```
-----------------------------------------------------------------------------------

    function Index_Subtype_Definitions
                (Type_Definition : in Asis.Type_Definition)
                return Asis.Expression_List;

-----------------------------------------------------------------------------------
-- Type_Definition - Specifies the array_type_definition to query
--
-- Returns a list of the index_subtype_definition subtype mark names for
-- an unconstrained_array_definition, in their order of appearance.
--
-- Appropriate Type_Kinds:
--      An_Unconstrained_Array_Definition
--
-- Appropriate Formal_Type_Kinds:
--      A_Formal_Unconstrained_Array_Definition
--
-- Returns Expression_Kinds:
--      An_Identifier
--      A_Selected_Component
--
--|ER------------------------------------------------------------------------------
--|ER A_Constrained_Array_Definition 3.6
--|CR
--|CR Child elements returned by:
--|CR    function Discrete_Subtype_Definitions
--|CR    function Array_Component_Definition
--
-----------------------------------------------------------------------------------
```
-- 16.16 function Discrete_Subtype_Definitions
```
-----------------------------------------------------------------------------------

    function Discrete_Subtype_Definitions
                        (Type_Definition : in Asis.Type_Definition)
                              return Asis.Definition_List;

-----------------------------------------------------------------------------------
-- Type_Definition - Specifies the array_type_definition to query
--
-- Returns the list of Discrete_Subtype_Definition elements of a
-- constrained_array_definition, in their order of appearance.
--
-- Appropriate Type_Kinds:
--      A_Constrained_Array_Definition
--
-- Appropriate Formal_Type_Kinds:
--      A_Formal_Constrained_Array_Definition
--
-- Returns Definition_Kinds:
--      A_Discrete_Subtype_Definition
--
-----------------------------------------------------------------------------------
```
-- 16.17 function Array_Component_Definition
```
-----------------------------------------------------------------------------------

    function Array_Component_Definition
                (Type_Definition : in Asis.Type_Definition)
                return Asis.Component_Definition;

-----------------------------------------------------------------------------------
-- Type_Definition - Specifies the array_type_definition to query
--
-- Returns the Component_Definition of the array_type_definition.
--
-- Appropriate Type_Kinds:
--      An_Unconstrained_Array_Definition
--      A_Constrained_Array_Definition
--
-- Appropriate Formal_Type_Kinds:
```

```
--      A_Formal_Unconstrained_Array_Definition
--      A_Formal_Constrained_Array_Definition
--
-- Returns Definition_Kinds:
--      A_Component_Definition
--
--|ER----------------------------------------------------------------------------
--|ER A_Record_Type_Definition - 3.8
--|ER A_Tagged_Record_Type_Definition - 3.8
--|CR
--|CR Child elements returned by:
--|CR    function Record_Definition
--|ER----------------------------------------------------------------------------
--|ER An_Access_Type_Definition - 3.10
--|CR
--|CR Child elements returned by:
--|CR    function Access_To_Object_Definition
--|CR    function Access_To_Subprogram_Parameter_Profile
--|CR    function Access_To_Function_Result_Profile
--
-------------------------------------------------------------------------------
```

-- 16.18 function Access_To_Object_Definition

```
-------------------------------------------------------------------------------

    function Access_To_Object_Definition
               (Type_Definition : in Asis.Type_Definition)
                     return Asis.Subtype_Indication;

-------------------------------------------------------------------------------
-- Type_Definition - Specifies the Access_Type_Definition to query
--
-- Returns the subtype_indication following the reserved word "access".
--
-- Appropriate Type_Kinds:
--      An_Access_Type_Definition.
--
-- Appropriate Formal_Type_Kinds:
--      A_Formal_Access_Type_Definition
--
-- Appropriate Access_Type_Kinds:
--      A_Pool_Specific_Access_To_Variable
--      An_Access_To_Variable
--      An_Access_To_Constant
--
-- Returns Element_Kinds:
--      A_Subtype_Indication
--
-------------------------------------------------------------------------------
```

-- 16.19 function Access_To_Subprogram_Parameter_Profile

```
-------------------------------------------------------------------------------

    function Access_To_Subprogram_Parameter_Profile
               (Type_Definition : in Asis.Type_Definition)
                     return Asis.Parameter_Specification_List;

-------------------------------------------------------------------------------
-- Type_Definition - Specifies the Access_Type_Definition to query
--
-- Returns a list of parameter_specification elements in the formal part of the
-- parameter_profile in the access_to_subprogram_definition.
--
-- Returns a Nil_Element_List if the parameter_profile has no formal part.
--
-- Results of this query may vary across ASIS implementations.  Some
-- implementations normalize all multiple name parameter_specification elements into
-- an equivalent sequence of corresponding single name
-- parameter_specification elements.  See Reference Manual 3.3.1(7).
--
-- Appropriate Type_Kinds:
--      An_Access_Type_Definition.
--      A_Formal_Access_Type_Definition.
--
```

```
-- Appropriate Access_Type_Kinds:
--      An_Access_To_Procedure
--      An_Access_To_Protected_Procedure
--      An_Access_To_Function
--      An_Access_To_Protected_Function
--
-- Returns Declaration_Kinds:
--      A_Parameter_Specification
--
---------------------------------------------------------------------------------------
```

-- 16.20 function Access_To_Function_Result_Profile

```
---------------------------------------------------------------------------------------

    function Access_To_Function_Result_Profile
                   (Type_Definition : in Asis.Type_Definition)
                         return Asis.Expression;

---------------------------------------------------------------------------------------
-- Type_Definition - Specifies the Access_Type_Definition to query
--
-- Returns the subtype_mark expression for the return type for the access
-- function.
--
-- Appropriate Type_Kinds:
--      An_Access_Type_Definition
--      A_Formal_Access_Type_Definition
--
-- Appropriate Access_Type_Kinds:
--      An_Access_To_Function
--      An_Access_To_Protected_Function
--
-- Returns Expression_Kinds:
--      An_Identifier
--      A_Selected_Component
--
--|ER------------------------------------------------------------------------------------
--|ER A_Root_Type_Definition - 3.5.4(9), 3.5.6(2) - No child elements
--|ER------------------------------------------------------------------------------------
--|ER A_Subtype_Indication - 3.3.2
--|CR
--|CR Child elements returned by:
--|CR    function Subtype_Mark
--|CR    function Subtype_Constraint
--
---------------------------------------------------------------------------------------
```

-- 16.21 function Subtype_Mark

```
---------------------------------------------------------------------------------------

    function Subtype_Mark (Definition : in Asis.Definition)
                          return Asis.Expression;

---------------------------------------------------------------------------------------
-- Definition  - Specifies the definition to query
--
-- Returns the subtype_mark expression of the definition.
--
-- Appropriate Definition_Kinds:
--      A_Subtype_Indication
--      A_Discrete_Subtype_Definition
--          Appropriate Discrete_Range_Kinds:
--               A_Discrete_Subtype_Indication
--      A_Discrete_Range
--          Appropriate Discrete_Range_Kinds:
--               A_Discrete_Subtype_Indication
--      A_Formal_Type_Definition
--          Appropriate Formal_Type_Kinds:
--               A_Formal_Derived_Type_Definition
--
-- Returns Expression_Kinds:
--      An_Identifier
--      A_Selected_Component
--      An_Attribute_Reference
--
```

-- 16.22 function Subtype_Constraint

```
    function Subtype_Constraint (Definition : in Asis.Definition)
                                 return Asis.Constraint;

-------------------------------------------------------------------------------------
-- Definition  - Specifies the definition to query
--
-- Returns the constraint of the subtype_indication.
--
-- Returns a Nil_Element if no explicit constraint is present.
--
-- Appropriate Definition_Kinds:
--      A_Subtype_Indication
--      A_Discrete_Subtype_Definition
--          Appropriate Discrete_Range_Kinds:
--               A_Discrete_Subtype_Indication
--      A_Discrete_Range
--          Appropriate Discrete_Range_Kinds:
--               A_Discrete_Subtype_Indication
--
-- Returns Definition_Kinds:
--      Not_A_Definition
--      A_Constraint
--
--|AN Application Note:
--|AN
--|AN When an unconstrained subtype indication for a type having
--|AN discriminants with default values is used, a Nil_Element is
--|AN returned by this function.  Use the queries Subtype_Mark, and
--|AN Corresponding_Name_Declaration [, and Corresponding_First_Subtype]
--|AN to obtain the declaration defining the defaults.
--
--|ER-------------------------------------------------------------------------------
--|ER A_Constraint - 3.2.2
--|ER
--|ER A_Simple_Expression_Range - 3.5
--|CR
--|CR Child elements returned by:
--|CR    function Lower_Bound
--|CR    function Upper_Bound
--
```

-- 16.23 function Lower_Bound

```
    function Lower_Bound (Constraint : in Asis.Range_Constraint)
                          return Asis.Expression;

-------------------------------------------------------------------------------------
-- Constraint  - Specifies the range_constraint or discrete_range to query
--
-- Returns the simple_expression for the lower bound of the range.
--
-- Appropriate Constraint_Kinds:
--      A_Simple_Expression_Range
--
-- Appropriate Discrete_Range_Kinds:
--      A_Discrete_Simple_Expression_Range
--
-- Returns Element_Kinds:
--      An_Expression
--
```

```
-------------------------------------------------------------------------------------
```
-- 16.24 function Upper_Bound
```
-------------------------------------------------------------------------------------

    function Upper_Bound (Constraint : in Asis.Range_Constraint)
                         return Asis.Expression;

-------------------------------------------------------------------------------------
-- Constraint  - Specifies the range_constraint or discrete_range to query
--
-- Returns the simple_expression for the upper bound of the range.
--
-- Appropriate Constraint_Kinds:
--      A_Simple_Expression_Range
--
-- Appropriate Discrete_Range_Kinds:
--      A_Discrete_Simple_Expression_Range
--
-- Returns Element_Kinds:
--      An_Expression
--
--|ER---------------------------------------------------------------------------------
--|ER A_Range_Attribute_Reference - 3.5
--|CR
--|CR Child elements returned by:
--|CR    function Range_Attribute
--
-------------------------------------------------------------------------------------
```
-- 16.25 function Range_Attribute
```
-------------------------------------------------------------------------------------

    function Range_Attribute (Constraint : in Asis.Range_Constraint)
                             return Asis.Expression;

-------------------------------------------------------------------------------------
-- Constraint  - Specifies the range_attribute_reference or
--               discrete_range attribute_reference to query
--
-- Returns the range_attribute_reference expression of the range.
--
-- Appropriate Constraint_Kinds:
--      A_Range_Attribute_Reference
--
-- Appropriate Discrete_Range_Kinds:
--      A_Discrete_Range_Attribute_Reference
--
-- Returns Expression_Kinds:
--      An_Attribute_Reference
--
--|ER---------------------------------------------------------------------------------
--|ER A_Digits_Constraint - 3.5.9
--|CR
--|CR Child elements returned by:
--|CR    function Digits_Expression
--|CR    function Real_Range_Constraint
--|ER---------------------------------------------------------------------------------
--|ER A_Delta_Constraint - J.3
--|CR
--|CR Child elements returned by:
--|CR    function Delta_Expression
--|CR    function Real_Range_Constraint
--|CR-----------------------------------------------------------------------------------
--|ER An_Index_Constraint - 3.6.1
--|CR
--|CR Child elements returned by:
--|CR    function Discrete_Ranges
--
```

```
-------------------------------------------------------------------------------
```
-- 16.26 function Discrete_Ranges
```
-------------------------------------------------------------------------------

    function Discrete_Ranges (Constraint : in Asis.Constraint)
                              return Asis.Discrete_Range_List;

-------------------------------------------------------------------------------
-- Constraint  - Specifies the array index_constraint to query
--
-- Returns the list of discrete_range components for an index_constraint,
-- in their order of appearance.
--
-- Appropriate Constraint_Kinds:
--      An_Index_Constraint
--
-- Returns Definition_Kinds:
--      A_Discrete_Range
--
--|ER---------------------------------------------------------------------------
--|ER A_Discriminant_Constraint - 3.7.1
--|CR
--|CR Child elements returned by:
--|CR    function Discriminant_Associations
--
-------------------------------------------------------------------------------
```
-- 16.27 function Discriminant_Associations
```
-------------------------------------------------------------------------------

    function Discriminant_Associations
                (Constraint : in Asis.Constraint;
                 Normalized : in Boolean := False)
                return Asis.Discriminant_Association_List;

-------------------------------------------------------------------------------
-- Constraint  - Specifies the discriminant_constraint to query
-- Normalized  - Specifies whether the normalized form is desired
--
-- Returns a list of the discriminant_association elements of the
-- discriminant_constraint.
--
-- Returns a Nil_Element_List if there are no discriminant_association elements.
--
-- An unnormalized list contains only explicit associations ordered as they
-- appear in the program text.  Each unnormalized association has a list of
-- discriminant_selector_name elements and an explicit expression.
--
-- A normalized list contains artificial associations representing all
-- explicit associations.  It has a length equal to the number of
-- discriminant_specification elements of the known_discriminant_part.  The order
-- of normalized associations matches the order of discriminant_specification elements.
--
-- Each normalized association represents a one on one mapping of a
-- discriminant_specification to the explicit expression.  A normalized
-- association has one A_Defining_Name component that denotes the
-- discriminant_specification, and one An_Expression component that is the
-- explicit expression.
--
-- Appropriate Constraint_Kinds:
--      A_Discriminant_Constraint
--
-- Returns Association_Kinds:
--      A_Discriminant_Association
--
--|IR Implementation Requirements:
--|IR
--|IR Normalized associations are Is_Normalized and Is_Part_Of_Implicit.
--|IR Normalized associations are never Is_Equal to unnormalized associations.
--
```

```
--|IP Implementation Permissions:
--|IP
--|IP An implementation may choose to normalize its internal representation
--|IP to use the defining_identifier element instead of the
--|IP discriminant_selector_name element.
--|IP
--|IP If so, this query will return Is_Normalized associations even if
--|IP Normalized is False, and the query Discriminant_Associations_Normalized
--|IP will return True.
--
--|AN Application Note:
--|AN
--|AN It is not possible to obtain either a normalized or unnormalized
--|AN Discriminant_Association list for an unconstrained record or derived
--|AN subtype_indication where the discriminant_association elements are supplied
--|AN by default; there is no constraint to query, and a Nil_Element is
--|AN returned from the query Subtype_Constraint.
--
--|ER---------------------------------------------------------------------------------
--|ER A_Component_Definition - 3.6
--|CR
--|CR Child elements returned by:
--|CR    function Component_Subtype_Indication
--
------------------------------------------------------------------------------------
```

-- 16.28 function Component_Subtype_Indication

```
------------------------------------------------------------------------------------

    function Component_Subtype_Indication
            (Component_Definition : in Asis.Component_Definition)
                return Asis.Subtype_Indication;

------------------------------------------------------------------------------------
-- Component_Definition - Specifies the Component_Definition to query
--
-- Returns the subtype_indication of the Component_Definition.
--
-- Appropriate Definition_Kinds:
--      A_Component_Definition
--
-- Returns Definition_Kinds:
--      A_Subtype_Indication
--
--|ER---------------------------------------------------------------------------------
--|ER A_Discrete_Subtype_Definition - 3.6
--|ER A_Discrete_Range - 3.6.1
--|ER
--|ER A_Discrete_Subtype_Indication
--|CR
--|CR Child elements returned by:
--|CR    function Subtype_Mark
--|CR    function Subtype_Constraint
--|CR
--|CR A_Discrete_Simple_Expression_Range
--|CR
--|CR Child elements returned by:
--|CR    function Lower_Bound
--|CR    function Upper_Bound
--|ER
--|ER---------------------------------------------------------------------------------
--|ER A_Discrete_Range_Attribute_Reference - 3.5
--|CR
--|CR Child elements returned by:
--|CR    function Range_Attribute
--|ER---------------------------------------------------------------------------------
--|ER An_Unknown_Discriminant_Part - 3.7 - No child elements
--|ER---------------------------------------------------------------------------------
--|ER A_Known_Discriminant_Part - 3.7
--|CR
--|CR Child elements returned by:
--|CR    function Discriminants
--
```

```
-------------------------------------------------------------------------------------
```
-- 16.29 function Discriminants
```
-------------------------------------------------------------------------------------

    function Discriminants (Definition : in Asis.Definition)
                           return Asis.Discriminant_Specification_List;

------------------------------------------------------------------------------------
-- Definition - Specifies the known_discriminant_part to query
--
-- Returns a list of discriminant_specification elements, in their order of appearance.
--
-- Results of this query may vary across ASIS implementations.  Some
-- implementations normalize all multi-name discriminant_specification elements into
-- an equivalent sequence of single name discriminant_specification elements.
-- See Reference Manual 3.3.1(7).
--
-- Appropriate Definition_Kinds:
--      A_Known_Discriminant_Part
--
-- Returns Declaration_Kinds:
--      A_Discriminant_Specification
--
--|ER----------------------------------------------------------------------------------
--|ER A_Record_Definition - 3.8
--|CR
--|CR Child elements returned by:
--|CR    function Record_Components
--|CR    function Implicit_Components
--
-------------------------------------------------------------------------------------
```
-- 16.30 function Record_Components
```
-------------------------------------------------------------------------------------

function Record_Components (Definition : in Asis.Definition;
                                Include_Pragmas : in Boolean := False)
                                return Asis.Record_Component_List;

-------------------------------------------------------------------------------------
-- Definition - Specifies the record_definition or variant to query
-- Include_Pragmas - Specifies whether pragmas are to be returned
--
-- Returns a list of the components and pragmas of the record_definition or
-- variant, in their order of appearance.
--
-- Declarations are not returned for implementation-defined components of the
-- record_definition.  See Reference Manual 13.5.1 (15).  These components are not
-- normally visible to the ASIS application.  However, they can be obtained
-- with the query Implicit_Components.
--
-- Appropriate Definition_Kinds:
--      A_Record_Definition
--      A_Variant
--
-- Returns Element_Kinds:
--      A_Pragma
--      A_Declaration
--      A_Definition
--      A_Clause
--
-- Returns Declaration_Kinds:
--      A_Component_Declaration
--
-- Returns Definition_Kinds:
--      A_Null_Component
--      A_Variant_Part
--
-- Returns Representation_Clause_Kinds:
--      An_Attribute_Definition_Clause
--
```

```
-------------------------------------------------------------------------------
```
-- 16.31 function Implicit_Components
```
-------------------------------------------------------------------------------

    function Implicit_Components
                    (Definition : in Asis.Definition)
                        return Asis.Record_Component_List;

-------------------------------------------------------------------------------
-- Definition - Specifies the record_definition or variant to query
--
-- Returns a list of all implicit implementation-defined components of the
-- record_definition or variant.  The Enclosing_Element of each component is
-- the Definition argument.  Each component is Is_Part_Of_Implicit.
--
-- Returns a Nil_Element_List if there are no implicit implementation-defined
-- components or if the ASIS implementation does not support such
-- implicit declarations.
--
-- Appropriate Definition_Kinds:
--      A_Record_Definition
--      A_Variant
--
-- Returns Element_Kinds:
--      A_Declaration
--
-- Returns Declaration_Kinds:
--      A_Component_Declaration
--
--|IP Implementation Permissions:
--|IP
--|IP Some implementations do not represent all forms of implicit
--|IP declarations such that elements representing them can be easily
--|IP provided.  An implementation can choose whether or not to construct
--|IP and provide artificial declarations for implicitly declared elements.
--|IP
--|IP Use the query Implicit_Components_Supported to determine if the
--|IP implementation provides implicit record components.
--
--|ER-----------------------------------------------------------------------------
--|ER A_Null_Record_Definition - 3.8 - No child elements
--|ER-----------------------------------------------------------------------------
--|ER A_Variant_Part - 3.8.1
--|CR
--|CR Child elements returned by:
--|CR    function Discriminant_Direct_Name
--|CR    function Variants
--
-------------------------------------------------------------------------------
```
-- 16.32 function Discriminant_Direct_Name
```
-------------------------------------------------------------------------------

    function Discriminant_Direct_Name
                (Variant_Part : in Asis.Record_Component)
                return Asis.Name;

-------------------------------------------------------------------------------
-- Variant_Part    - Specifies the variant_part to query
--
-- Returns the Discriminant_Direct_Name of the variant_part.
--
-- Appropriate Definition_Kinds:
--      A_Variant_Part
--
-- Returns Expression_Kinds:
--      An_Identifier
--
```

```
-------------------------------------------------------------------------------------
```
-- 16.33 function Variants
```
-------------------------------------------------------------------------------------

    function Variants (Variant_Part    : in Asis.Record_Component;
                       Include_Pragmas : in Boolean := False)
                       return Asis.Variant_List;

-------------------------------------------------------------------------------------
-- Variant_Part    - Specifies the variant_part to query
-- Include_Pragmas - Specifies whether pragmas are to be returned
--
-- Returns a list of variants that make up the record component, in their
-- order of appearance.
--
-- The only pragmas returned are those following the reserved word "is"
-- and preceding the reserved word "when" of first variant, and those between
-- following variants.
--
-- Appropriate Definition_Kinds:
--      A_Variant_Part
--
-- Returns Element_Kinds:
--      A_Pragma
--      A_Definition
--
-- Returns Definition_Kinds:
--      A_Variant
--
--|ER--------------------------------------------------------------------------------
--|ER A_Variant - 3.8.1
--|CR
--|CR Child elements returned by:
--|CR    function Variant_Choices
--|CR    function Record_Components
--|CR    function Implicit_Components
--
-------------------------------------------------------------------------------------
```
-- 16.34 function Variant_Choices
```
-------------------------------------------------------------------------------------

    function Variant_Choices (Variant : in Asis.Variant)
                             return Asis.Element_List;

-------------------------------------------------------------------------------------
-- Variant - Specifies the variant to query
--
-- Returns the discrete_choice_list elements, in their order of appearance.
-- Choices are either an expression, a discrete range, or an others choice.
--
-- Appropriate Definition_Kinds:
--      A_Variant
--
-- Returns Element_Kinds:
--      An_Expression
--      A_Definition
--
-- Returns Definition_Kinds:
--      A_Discrete_Range
--      An_Others_Choice
--
--|ER--------------------------------------------------------------------------------
--|ER A_Private_Type_Definition - 7.3 - No child elements
--|ER A_Tagged_Private_Type_Definition - 7.3 - No child elements
--|ER--------------------------------------------------------------------------------
--|ER A_Private_Extension_Definition - 7.3
--|CR
--|CR Child elements returned by:
--|CR    function Ancestor_Subtype_Indication
--
```

```
-------------------------------------------------------------------------------
```
-- 16.35 function Ancestor_Subtype_Indication
```
-------------------------------------------------------------------------------

    function Ancestor_Subtype_Indication
                   (Definition : in Asis.Definition)
                         return Asis.Subtype_Indication;

-------------------------------------------------------------------------------
-- Definition - Specifies the definition to query
--
-- Returns the ancestor_subtype_indication following the reserved word "new"
-- in the private_extension_declaration.
--
-- Appropriate Definition_Kinds:
--      A_Private_Extension_Definition
--
-- Returns Definition_Kinds:
--      A_Subtype_Indication
--
--|ER----------------------------------------------------------------------------------
--|ER A_Task_Definition - 9.1
--|ER A_Protected_Definition - 9.4
--|CR
--|CR Child elements returned by:
--|CR    functions Visible_Part_Items and Private_Part_Items
--
-------------------------------------------------------------------------------
```
-- 16.36 function Visible_Part_Items
```
-------------------------------------------------------------------------------

    function Visible_Part_Items
                (Definition : in Asis.Definition;
                 Include_Pragmas : in Boolean := False)
                return Asis.Declarative_Item_List;

-------------------------------------------------------------------------------
-- Type_Definition - Specifies the type_definition to query
-- Include_Pragmas - Specifies whether pragmas are to be returned
--
-- Returns a list of declarations, representation clauses, and pragmas
-- in the visible part of the task or protected definition, in their order
-- of appearance.  The list does not include discriminant_specification elements of
-- the known_discriminant_part, if any, of the protected type or task type
-- declaration.
--
-- Returns a Nil_Element_List if there are no items.
--
-- Appropriate Definition_Kinds:
--      A_Task_Definition
--      A_Protected_Definition
--
-- Returns Element_Kinds:
--      A_Pragma
--      A_Declaration
--      A_Clause
--
-------------------------------------------------------------------------------
```
-- 16.37 function Private_Part_Items
```
-------------------------------------------------------------------------------

    function Private_Part_Items
                (Definition : in Asis.Definition;
                 Include_Pragmas : in Boolean := False)
                 return Asis.Declarative_Item_List;

-------------------------------------------------------------------------------
-- Type_Definition - Specifies the task type definition to query
-- Include_Pragmas - Specifies whether pragmas are to be returned
--
-- Returns a list of declarations, representation clauses, and pragmas in the private
-- part of the task or protected definition, in their order of appearance.
```

```
--
-- Returns a Nil_Element_List if there are no items.
--
-- Appropriate Definition_Kinds:
--      A_Task_Definition
--      A_Protected_Definition
--
-- Returns Element_Kinds:
--      A_Pragma
--      A_Declaration
--      A_Clause
--
---------------------------------------------------------------------------------------
```

-- 16.38 function Is_Private_Present

```
---------------------------------------------------------------------------------------

    function Is_Private_Present
                (Definition : in Asis.Definition) return Boolean;

---------------------------------------------------------------------------------------
-- Definition - Specifies the definition to query
--
-- Returns True if the argument is a task_definition or a protected_definition
-- that has a reserved word "private" marking the beginning of a (possibly empty)
-- private part.
--
-- Returns False for any definition without a private part.
-- Returns False for any unexpected Element.
--
-- Expected Definition_Kinds:
--      A_Task_Definition
--      A_Protected_Definition
--
--|ER------------------------------------------------------------------------------------
--|ER A_Formal_Type_Definition - 12.5
--|ER
--|ER A_Formal_Private_Type_Definition         - 12.5.1 - No child elements
--|ER A_Formal_Tagged_Private_Type_Definition  - 12.5.1 - No child elements
--|ER
--|ER A_Formal_Derived_Type_Definition
--|CR Child elements returned by:
--|CR    function Subtype_Mark
--
--|ER------------------------------------------------------------------------------------
--|ER A_Formal_Discrete_Type_Definition        - 12.5.2 - No child elements
--|ER A_Formal_Signed_Integer_Type_Definition  - 12.5.2 - No child elements
--|ER A_Formal_Modular_Type_Definition         - 12.5.2 - No child elements
--|ER A_Formal_Floating_Point_Definition       - 12.5.2 - No child elements
--|ER A_Formal_Ordinary_Fixed_Point_Definition - 12.5.2 - No child elements
--|ER A_Formal_Decimal_Fixed_Point_Definition  - 12.5.2 - No child elements
--|ER------------------------------------------------------------------------------------
--|ER A_Formal_Unconstrained_Array_Definition  - 12.5.3
--|CR
--|CR Child elements returned by:
--|CR    function Index_Subtype_Definitions
--|CR    function Array_Component_Definition
--|ER------------------------------------------------------------------------------------
--|ER A_Formal_Constrained_Array_Definition    - 12.5.3
--|CR
--|CR Child elements returned by:
--|CR    function Discrete_Subtype_Definitions
--|CR    function Array_Component_Definition
--|ER------------------------------------------------------------------------------------
--|ER A_Formal_Access_Type_Definition          - 12.5.4
--|CR
--|CR Child elements returned by:
--|CR    function Access_To_Object_Definition
--|CR    function Access_To_Subprogram_Parameter_Profile
--|CR    function Access_To_Function_Result_Profile
--
---------------------------------------------------------------------------------------

end Asis.Definitions;
```

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

-- 17 package Asis.Expressions

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
package Asis.Expressions is
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- Asis.Expressions encapsulates a set of queries that operate on
-- An_Expression and An_Association elements.
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

-- 17.1 function Corresponding_Expression_Type

```
-------------------------------------------------------------------------------

    function Corresponding_Expression_Type (Expression : in Asis.Expression)
                         return Asis.Declaration;

-------------------------------------------------------------------------------
-- Expression  - Specifies the expression to query
--
-- Returns the type declaration for the type or subtype of the expression.
-- This query does not "unwind" subtypes or derived types to get to the
-- Corresponding_First_Subtype or Corresponding_Parent_Subtype declarations.
-- For example, for the following program text:
--
--     type Int is range -5_000 .. 5_000;
--     type My_Int is new Int;
--     type Good_Int is new My_Int;
--     Var: Good_Int;
--
-- The type declaration for Good_Int should be returned. The "unwinding" should
-- not occur. The type declaration for either My_Int or Int should not be returned.
--
-- Returns a Nil_Element if the argument Expression does not represent an Ada
-- expression having an Ada type, including the following classes:
--
-- - Naming expressions that name packages, subprograms, tasks, etc.  These
--   expressions do have a Corresponding_Name_Definition and a
--   Corresponding_Name_Declaration. Although task objects do have
--   a type, this query is limited, on purpose.  Thus, when a naming
--   expression is given to this query (for packages, subprograms,
--   tasks, etc.), this query will return Nil_Element.  As the
--   Application Note below indicates, if any further information
--   is needed, the element should be queried by
--   Corresponding_Name_Definition or Corresponding_Name_Declaration,
--   which should eventually return an A_Task_Type_Declaration element.
--
-- - When An_Identifier Element representing an attribute designator is passed
--   as the actual to this query.
--
-- - The Actual_Parameter Expression from A_Pragma_Argument_Association for a
--   Pragma may or may not have a Corresponding_Expression_Type.
--
-- - An_Attribute_Reference Element also may or may not have a
--   Corresponding_Expression_Type;
--
-- - An enumeration_aggregate which is a part of enumeration_representation_clause.
--
-- Returns a Nil_Element, if the statically determinable type of Expression is a
-- class-wide type, or the Expression corresponds to an inner sub-aggregate in
-- multi-dimensional array aggregates.
--
--|AN Application Note:
--|AN
--|AN If the returned declaration is Nil, an application should make its own
--|AN analysis based on Corresponding_Name_Definition or
--|AN Corresponding_Name_Declaration to get more information about the argument,
--|AN including the static type resolution for class-wide expressions, if needed.
--|AN Use Enclosing_Element to determine if Expression is from pragma argument
--|AN association. If for such an expression, Corresponding_Name_Definition
--|AN raises ASIS_Failed (with a Status of Value_Error), this An_Expression
```

```
--|AN element does not represent a normal Ada expression at all and does not
--|AN follow normal Ada semantic rules.
--|AN For example, "pragma Private_Part (Open => Yes);", the "Yes" expression
--|AN may simply be a "keyword" that is specially recognized by the
--|AN implementor’s compilation system and may not refer to any
--|AN declared object.
--
-- Appropriate Element_Kinds:
--      An_Expression
--
-- Returns Element_Kinds:
--      Not_An_Element
--      A_Declaration
--
--|ER An_Integer_Literal - 2.4 - No child elements
--|ER A_Real_Literal     - 2.4 - No child elements
--|ER A_String_Literal   - 2.6 - No child elements
--|ER
--|ER A string image returned by:
--|ER    function Value_Image
--
---------------------------------------------------------------------------------------
```

-- 17.2 function Value_Image

```
---------------------------------------------------------------------------------------

    function Value_Image (Expression : in Asis.Expression) return Wide_String;

---------------------------------------------------------------------------------------
-- Expression  - Specifies the expression to query
--
-- Returns the string image of the value of the string, integer, or real
-- literal.
--
-- For string literals, Value will return the quotes around the string
-- literal, these quotes are doubled, just as any quote appearing embedded in
-- the string literal in the program text.
--
-- The form of numbers returned by this query may vary between implementors.
-- Implementors are encouraged, but not required, to return numeric literals
-- using the same based or exponent form used in the original compilation text.
--
-- Appropriate Expression_Kinds:
--      An_Integer_Literal
--      A_Real_Literal
--      A_String_Literal
--
---------------------------------------------------------------------------------------
--|ER An_Identifier          - 4.1 - No child elements
--|ER An_Operator_Symbol     - 4.1 - No child elements
--|ER A_Character_Literal    - 4.1 - No child elements
--|ER An_Enumeration_Literal - 4.1 - No child elements
--|ER
--|ER A string image returned by:
--|ER    function Name_Image
--|ER
--|ER Semantic elements returned by:
--|ER    function Corresponding_Name_Definition
--|ER    function Corresponding_Name_Definition_List
--|ER    function Corresponding_Name_Declaration
--
---------------------------------------------------------------------------------------
```

-- 17.3 function Name_Image

```
---------------------------------------------------------------------------------------

    function Name_Image (Expression : in Asis.Expression) return Program_Text;

---------------------------------------------------------------------------------------
-- Name  - Specifies the name to query
--
-- Returns the program text image of the name.
--
-- An_Operator_Symbol elements have names with embedded quotes """abs"""
--   (function "abs").
```

```
--
-- A_Character_Literal elements have names with embedded apostrophes "'x'"
--   (literal 'x').
--
-- An_Enumeration_Literal and An_Identifier elements have identifier names
--   "Blue" (literal Blue) "Abc" (identifier Abc).
--
-- Note: Implicit subtypes that can be encountered while traversing the
-- semantic information embedded in implicit inherited subprogram declarations
-- (Reference Manual 3.4 (17-22)) could have names that are unique in a
-- particular scope.  This is because these subtypes are Is_Part_Of_Implicit
-- declarations that do not form part of the physical text of the original
-- compilation units. Some applications may wish to carefully separate the names
-- of declarations from the names of Is_Part_Of_Implicit declaration when
-- creating symbol tables and other name-specific lookup mechanisms.
--
-- The case of names returned by this query may vary between implementors.
-- Implementors are encouraged, but not required, to return names in the
-- same case as was used in the original compilation text.
--
-- Appropriate Expression_Kinds:
--      An_Identifier
--      An_Operator_Symbol
--      A_Character_Literal
--      An_Enumeration_Literal
--
-------------------------------------------------------------------------------------
```

-- 17.4 function References

```
-------------------------------------------------------------------------------------

    function References (Name           : in Asis.Element;
                         Within_Element : in Asis.Element;
                         Implicitly     : in Boolean := False)
                        return Asis.Name_List;

-------------------------------------------------------------------------------------
-- Name    - Specifies the entity to query
-- Within_Element - Specifies the limits for the query which is limited
--                  to the Element and its children.
--
-- If the Implicitly argument is True:
--   Returns all usage references of the given entity made by both explicit
--   and implicit elements within the given limits.
--
-- If the Implicitly argument is False:
--   Returns all usage references of the given entity made only by explicit
--   elements within the given limits.
--
-- Returned references are in their order of appearance.
--
-- Appropriate Element_Kinds:
--      A_Defining_Name
-- Returns Element_Kinds:
--      An_Expression
--
-- May raise ASIS_Failed with a Status of Obsolete_Reference_Error if the
-- argument is part of an inconsistent compilation unit.
--
-------------------------------------------------------------------------------------
```

-- 17.5 function Is_Referenced

```
-------------------------------------------------------------------------------------

    function Is_Referenced (Name           : in Asis.Element;
                            Within_Element : in Asis.Element;
                            Implicitly     : in Boolean := False)
                           return Boolean;

-------------------------------------------------------------------------------------
-- Name    - Specifies the entity to query
-- Within_Element - Specifies the limits for the query which is limited
--                  to the Element and its children.
--
```

```
-- If the Implicitly argument is True:
--   Returns True if the Name is referenced by either implicit or explicit
--   elements within the given limits.
--
-- If the Implicitly argument is False:
--   Returns True only if the Name is referenced by explicit elements.
--
-- Returns False for any unexpected Element.
--
-- Expected Element_Kinds:
--      A_Defining_Name
--
-- May raise ASIS_Failed with a Status of Obsolete_Reference_Error if the
-- argument is part of an inconsistent compilation unit.
--
-------------------------------------------------------------------------------------
```

-- 17.6 function Corresponding_Name_Definition

```
-------------------------------------------------------------------------------------

    function Corresponding_Name_Definition (Reference : in Asis.Expression)
                             return Asis.Defining_Name;

-------------------------------------------------------------------------------------
-- Reference   - Specifies an expression to query
--
-- Returns the defining_identifier, defining_character_literal,
-- defining_operator_symbol, or defining_program_unit_name from the
-- declaration of the referenced entity.
--
-- - Record component references return the defining name of the
--   record discriminant or component_declaration.  For references to inherited
--   declarations of derived types, the Corresponding_Name_Definition returns
--   the defining name of the implicit inherited declaration.
--
-- - References to implicit operators and inherited subprograms will return
--   an Is_Part_Of_Implicit defining name for the operation.  The
--   Enclosing_Element of the name is an implicit declaration for the
--   operation.  The Enclosing_Element of the declaration is the associated
--   derived_type_definition.
--
-- - References to formal parameters given in calls to inherited subprograms
--   will return an Is_Part_Of_Implicit defining name for the
--   Parameter_Specification from the inherited subprogram specification.
--
-- - References to visible components of instantiated generic packages will
--   return a name from the expanded generic specification instance.
--
-- - References, within expanded generic instances, that refer to other
--   components of the same, or an enclosing, expanded generic instance,
--   return a name from the appropriate expanded specification or body
--   instance.
--
-- In case of renaming, the function returns the new name for the entity.
--
-- Returns a Nil_Element if the reference is to an implicitly declared
-- element for which the implementation does not provide declarations and
-- defining name elements.
--
-- Returns a Nil_Element if the argument is a dispatching call.
--
-- The Enclosing_Element of a non-Nil result is either a Declaration or a
-- Statement.
--
-- Appropriate Expression_Kinds:
--      An_Identifier
--      An_Operator_Symbol
--      A_Character_Literal
--      An_Enumeration_Literal
--
-- Returns Element_Kinds:
--      Not_An_Element
--      A_Defining_Name
--
```

```
--|IP Implementation Permissions:
--|IP
--|IP An implementation may choose to return any part of multi-part
--|IP declarations and definitions. Multi-part declaration/definitions
--|IP can occur for:
--|IP
--|IP - Subprogram specification in package specification, package body,
--|IP   and subunits (is separate);
--|IP
--|IP - Entries in package specification, package body, and subunits (is separate);
--|IP
--|IP - Private type and full type declarations;
--|IP
--|IP - Incomplete type and full type declarations; and
--|IP
--|IP - Deferred constant and full constant declarations.
--|IP
--|IP No guarantee is made that the element will be the first part or
--|IP that the determination will be made due to any visibility rules.
--|IP An application should make its own analysis for each case based
--|IP on which part is returned.
--|IP
--|IP Some implementations do not represent all forms of implicit
--|IP declarations such that elements representing them can be easily
--|IP provided.  An implementation can choose whether or not to construct
--|IP and provide artificial declarations for implicitly declared elements.
--
--|IR Implementation Requirements:
--|IR
--|IR Raises ASIS_Inappropriate_Element, with a Status of Value_Error, if passed
--|IR a reference that does not have a declaration:
--|IR
--|IR - a reference to an attribute_designator.  Attributes are defined, but
--|IR   have no implicit or explicit declarations;
--|IR
--|IR - an identifier which syntactically is placed before "=>" in a
--|IR   pragma_argument_association which has the form of a named association;
--|IR   such an identifier can never have a declaration;
--|IR
--|IR - an identifier specific to a pragma (Reference Manual, 2.8(10));
--|IR
--|IR     pragma Should_I_Check ( Really => Yes );
--|IR
--|IR In this example, both the names Really and Yes have no declaration.
--|IR
--|IR Raises ASIS_Inappropriate_Element, with a Status of Value_Error, if passed
--|IR a portion of a pragma that was "ignored" by the compiler and which does
--|IR not have (sufficient) semantic information for a proper return result
--|IR to be computed.  For example,
--|IR
--|IR     pragma I_Am_Ignored (Foof);
--|IR
--|IR The "Foof" expression is An_Identifier but raises this exception
--|IR if passed to Corresponding_Name_Definition if the pragma was ignored
--|IR or unprocessed.
--|IR
--|IR Raises ASIS_Inappropriate_Element, with a Status of Value_Error, if passed
--|IR a portion of a pragma that is an ambiguous reference to more than one
--|IR entity.  For example,
--|IR
--|IR     pragma Inline ("+");        -- Inlines all "+" operators
--|IR
--|IR The "+" expression is An_Operator_Symbol but raises this
--|IR exception if it referenced more than one "+" operator.  In this
--|IR case, the Corresponding_Name_Definition_List query can be used to obtain a
--|IR list of referenced entities.
--
```

```
-------------------------------------------------------------------------------------
```
-- 17.7 function Corresponding_Name_Definition_List
```
-------------------------------------------------------------------------------------

    function Corresponding_Name_Definition_List (Reference : in Asis.Element)
                               return Asis.Defining_Name_List;

-------------------------------------------------------------------------------------
-- Reference   - Specifies an entity reference to query
--
-- Exactly like Corresponding_Name_Definition except it returns a list.
-- The list will almost always have a length of one.  The exception to this
-- is the case where an expression in a pragma is ambiguous and reference
-- more than one entity.  For example,
--
--      pragma Inline ("+");        -- Inlines all "+" operators
--
-- The "+" expression is An_Operator_Symbol but could reference more than one "+"
-- operator.  In this case, the resulting list includes all referenced entities.
--
-- Appropriate Expression_Kinds:
--      An_Identifier
--      An_Operator_Symbol
--      A_Character_Literal
--      An_Enumeration_Literal
--
-- Returns Element_Kinds:
--      A_Defining_Name
--
-------------------------------------------------------------------------------------
```
-- 17.8 function Corresponding_Name_Declaration
```
-------------------------------------------------------------------------------------

    function Corresponding_Name_Declaration (Reference : in Asis.Expression)
                          return Asis.Element;

-------------------------------------------------------------------------------------
-- Reference   - Specifies the entity reference to query
--
-- Returns the declaration that declared the entity named by the given
-- reference.  The result is exactly the same as:
--
--      Result := Corresponding_Name_Definition (Reference);
--      if not Is_Nil (Result) then
--          Result := Enclosing_Element (Result);
--      end if;
--      return Result;
--
-- See the comments for Corresponding_Name_Definition for details.
-- The result is either a Declaration or a Statement.  Statements result
-- from references to statement labels, loop identifiers, and block identifiers.
--
-- Appropriate Element_Kinds:
--      An_Expression
--
-- Appropriate Expression_Kinds:
--      An_Identifier
--      An_Operator_Symbol
--      A_Character_Literal
--      An_Enumeration_Literal
--
-- Returns Element_Kinds:
--      A_Declaration
--      A_Statement
--
-- Predefined types, exceptions, operators in package Standard can be
-- checked by testing that the enclosing Compilation_Unit is standard.
--
--|ER---------------------------------------------------------------------------------
--|ER An_Explicit_Dereference - 4.1
--|CR
--|CR Child elements returned by: function Prefix
--
```

```
------------------------------------------------------------------------------
```

-- 17.9 function Prefix

```
------------------------------------------------------------------------------

    function Prefix (Expression : in Asis.Expression) return Asis.Expression;

------------------------------------------------------------------------------
-- Expression  - Specifies the name expression to query
--
-- Returns the prefix (the construct to the left of: the rightmost unnested
-- left parenthesis in function_call elements and indexed_component elements or slice
-- elements, the rightmost 'dot' for selected_component elements, or the rightmost
-- tick for attribute_reference elements).
--
-- Returns the operator_symbol for infix operator function calls.  The infix
-- form A + B is equivalent to the prefix form "+"(A, B).
--
-- Appropriate Expression_Kinds:
--      An_Explicit_Dereference       P.ALL
--      An_Attribute_Reference        Priv'Base'First
--      A_Function_Call               Abc(...) or Integer'Image(...)
--      An_Indexed_Component          An_Array(3)
--      A_Selected_Component          A.B.C
--      A_Slice                       An_Array(3 .. 5)
--
-- Returns Expression_Kinds:
--      An_Expression
--
--|ER--------------------------------------------------------------------------------
--|ER An_Indexed_Component - 4.1.1
--|ER
--|CR
--|CR Child elements returned by:
--|CR    function Prefix
--|CR    function Index_Expressions
--|CR
--
------------------------------------------------------------------------------
```

-- 17.10 function Index_Expressions

```
------------------------------------------------------------------------------

    function Index_Expressions (Expression : in Asis.Expression)
                               return Asis.Expression_List;

------------------------------------------------------------------------------
-- Expression  - Specifies an indexed_component to query
--
-- Returns the list of expressions (possibly only one) within the parenthesis,
-- in their order of appearance.
--
-- Appropriate Expression_Kinds:
--      An_Indexed_Component
--
-- Returns Element_Kinds:
--      An_Expression
--
--|ER--------------------------------------------------------------------------------
--|ER A_Slice - 4.1.2
--|CR
--|CR Child elements returned by:
--|CR    function Prefix
--|CR    function Slice_Range
--|CR
--
```

```
------------------------------------------------------------------------------
```
-- 17.11 function Slice_Range
```
------------------------------------------------------------------------------

    function Slice_Range (Expression : in Asis.Expression)
                         return Asis.Discrete_Range;

------------------------------------------------------------------------------
-- Expression  - Specifies the slice to query
--
-- Returns the discrete range of the slice.
--
-- Appropriate Expression_Kinds:
--      A_Slice
--
-- Returns Definition_Kinds:
--      A_Discrete_Range
--
------------------------------------------------------------------------------
```
-- 17.12 function Selector
```
------------------------------------------------------------------------------

    function Selector (Expression : in Asis.Expression)
                      return Asis.Expression;

------------------------------------------------------------------------------
-- Expression  - Specifies the selected_component to query
--
-- Returns the selector (the construct to the right of the rightmost ’dot’ in
-- the selected_component).
--
-- Appropriate Expression_Kinds:
--      A_Selected_Component
--
-- Returns Expression_Kinds:
--      An_Identifier
--      An_Operator_Symbol
--      A_Character_Literal
--      An_Enumeration_Literal
--
------------------------------------------------------------------------------
```
-- 17.13 function Attribute_Designator_Identifier
```
------------------------------------------------------------------------------

    function Attribute_Designator_Identifier
            (Expression : in Asis.Expression)
                return Asis.Expression;

------------------------------------------------------------------------------
-- Expression  - Specifies an attribute_reference expression to query
--
-- Returns the identifier of the attribute_designator (the construct to the
-- right of the rightmost tick of the attribute_reference).  The Prefix of
-- the attribute_reference can itself be an attribute_reference as in
-- T’BASE’FIRST where the prefix is T’BASE and the attribute_designator name
-- is FIRST.
--
-- Attribute_designator reserved words "access", "delta", and "digits" are treated
-- as An_Identifier.
--
-- Appropriate Expression_Kinds:
--      An_Attribute_Reference
--
-- Returns Expression_Kinds:
--      An_Identifier
--
```

```
-------------------------------------------------------------------------------
```

-- 17.14 function Attribute_Designator_Expressions

```
-------------------------------------------------------------------------------

    function Attribute_Designator_Expressions
            (Expression : in Asis.Expression)
                return Asis.Expression_List;

-------------------------------------------------------------------------------
-- Expression  - Specifies an attribute expression to query
--
-- Returns the static expressions associated with the optional argument of the
-- attribute_designator.  Expected predefined attributes are A'First(N),
-- A'Last(N), A'Length(N), and A'Range(N).
--
-- Returns a Nil_Element_List if there are no arguments.
--
-- Appropriate Expression_Kinds:
--      An_Attribute_Reference
--          Appropriate Attribute_Kinds:
--               A_First_Attribute
--               A_Last_Attribute
--               A_Length_Attribute
--               A_Range_Attribute
--               An_Implementation_Defined_Attribute
--               An_Unknown_Attribute
--
-- Returns Element_Kinds:
--      An_Expression
--
--|IP Implementation Permissions:
--|IP
--|IP This query returns a list to support implementation-defined attributes
--|IP that may have more than one static_expression.
--
-------------------------------------------------------------------------------
```

-- 17.15 function Record_Component_Associations

```
-------------------------------------------------------------------------------

    function Record_Component_Associations
                  (Expression : in Asis.Expression;
                         Normalized : in Boolean := False)
                        return Asis.Association_List;

-------------------------------------------------------------------------------
-- Expression  - Specifies an aggregate expression to query
-- Normalized  - Specifies whether the normalized form is desired
--
-- Returns a list of the record_component_association elements of a record_aggregate
-- or an extension_aggregate.
--
-- Returns a Nil_Element_List if the aggregate is of the form (null record).
--
-- An unnormalized list contains all needed associations ordered as they
-- appear in the program text.  Each unnormalized association has an optional
-- list of discriminant_selector_name elements, and an explicit expression.
--
-- A normalized list contains artificial associations representing all
-- needed components in an order matching the declaration order of the
-- needed components.
--
-- Each normalized association represents a one on one mapping of a
-- component to the explicit expression.  A normalized association has one
-- A_Defining_Name component that denotes the discriminant_specification or
-- component_declaration, and one An_Expression component that is the
-- expression.
--
-- Appropriate Expression_Kinds:
--      A_Record_Aggregate
--      An_Extension_Aggregate
--
-- Returns Association_Kinds:
--      A_Record_Component_Association
```

```
--
--|IR Implementation Requirements:
--|IR
--|IR Normalized associations are Is_Normalized and Is_Part_Of_Implicit.
--|IR Normalized associations are never Is_Equal to unnormalized associations.
--
--|IP Implementation Permissions:
--|IP
--|IP An implementation may choose to normalize its internal representation
--|IP to use the defining_identifier element instead of the component_selector_name
--|IP element.
--|IP
--|IP If so, this query will return Is_Normalized associations even if
--|IP Normalized is False, and the query
--|IP Record_Component_Associations_Normalized will return True.
--
-------------------------------------------------------------------------------------
```

-- 17.16 function Extension_Aggregate_Expression

```
-------------------------------------------------------------------------------------

    function Extension_Aggregate_Expression
                              (Expression : in Asis.Expression)
                                       return Asis.Expression;

-------------------------------------------------------------------------------------
-- Expression  - Specifies an extension_aggregate expression to query
--
-- Returns the ancestor_part expression preceding the reserved word with in
-- the extension_aggregate.
--
-- Appropriate Expression_Kinds:
--      An_Extension_Aggregate
--
-- Returns Element_Kinds:
--      An_Expression
--
-------------------------------------------------------------------------------------
```

-- 17.17 function Array_Component_Associations

```
-------------------------------------------------------------------------------------

    function Array_Component_Associations
                 (Expression : in Asis.Expression)
                       return Asis.Association_List;

-------------------------------------------------------------------------------------
-- Expression  - Specifies an array aggregate expression to query
--
-- Returns a list of the Array_Component_Associations in an array aggregate.
--
-- Appropriate Expression_Kinds:
--      A_Positional_Array_Aggregate
--      A_Named_Array_Aggregate
--
-- Returns Association_Kinds:
--      An_Array_Component_Association
--
--|AN Application Note:
--|AN
--|AN While positional_array_aggregate elements do not have
--|AN array_component_association elements defined by Ada syntax, ASIS treats
--|AN A_Positional_Array_Aggregate as if it were A_Named_Array_Aggregate.
--|AN The An_Array_Component_Association elements returned will have
--|AN Array_Component_Choices that are a Nil_Element_List for all positional
--|AN expressions except an others choice.
--
```

-- 17.18 function Array_Component_Choices

```
    function Array_Component_Choices
            (Association : in Asis.Association)
                return Asis.Expression_List;

---------------------------------------------------------------------------
-- Association - Specifies the component association to query
--
-- If the Association is from a named_array_aggregate:
--
-- - Returns the discrete_choice_list order of appearance.  The choices are
--   either An_Expression or A_Discrete_Range elements, or a single
--   An_Others_Choice element.
--
-- If the Association is from a positional_array_aggregate:
--
-- - Returns a single An_Others_Choice if the association is an others
--   choice (others => expression).
--
-- - Returns a Nil_Element_List otherwise.
--
-- Appropriate Association_Kinds:
--      An_Array_Component_Association
--
-- Returns Element_Kinds:
--      A_Definition
--      An_Expression
--
-- Returns Definition_Kinds:
--      A_Discrete_Range
--      An_Others_Choice
--
---------------------------------------------------------------------------
```

-- 17.19 function Record_Component_Choices

```
    function Record_Component_Choices
            (Association : in Asis.Association)
                return Asis.Expression_List;

---------------------------------------------------------------------------
-- Association - Specifies the component association to query
--
-- If the Association argument is from an unnormalized list:
--
-- - If the Association is a named component association:
--
--   o Returns the component_choice_list order of appearance.  The choices are
--     either An_Identifier elements representing component_selector_name elements, or
--     a single An_Others_Choice element.
--
--   o The Enclosing_Element of the choices is the Association argument.
--
-- - If the Association is a positional component association:
--
--   o Returns a Nil_Element_List.
--
-- If the Association argument is from a Normalized list:
--
-- - Returns a list containing a single choice:
--
--   o A_Defining_Name element representing the defining_identifier of
--     the component_declaration.
--
--   o The Enclosing_Element of the A_Defining_Name is the component_declaration.
--
-- Normalized lists contain artificial ASIS An_Association elements that
-- provide one formal A_Defining_Name => An_Expression pair per
-- association.  These artificial associations are Is_Normalized.  Their
-- component A_Defining_Name is not Is_Normalized.
```

```
--
-- Appropriate Association_Kinds:
--      A_Record_Component_Association
--
-- Returns Element_Kinds:
--      A_Defining_Name             -- Is_Normalized(Association)
--      An_Expression               -- not Is_Normalized(Association)
--           Returns Expression_Kinds:
--                An_Identifier
--      A_Definition
--           Returns Definition_Kinds:
--                An_Others_Choice
--
-----------------------------------------------------------------------------------------
```

-- 17.20 function Component_Expression

```
-----------------------------------------------------------------------------------------

    function Component_Expression (Association : in Asis.Association)
                                  return Asis.Expression;

-----------------------------------------------------------------------------------------
-- Association - Specifies the component association to query
--
-- Returns the expression of the record_component_association or
-- array_component_association.
--
-- The Enclosing_Element of the expression is the Association argument.
--
-- Normalized lists contain artificial ASIS An_Association elements that
-- provide one formal A_Defining_Name => An_Expression pair per
-- association.  These artificial associations are Is_Normalized.  Their
-- component An_Expression elements are not Is_Normalized.
--
-- Appropriate Association_Kinds:
--      A_Record_Component_Association
--      An_Array_Component_Association
--
-- Returns Element_Kinds:
--      An_Expression
--
-----------------------------------------------------------------------------------------
```

-- 17.21 function Formal_Parameter

```
-----------------------------------------------------------------------------------------

    function Formal_Parameter (Association : in Asis.Association)
                              return Asis.Element;

-----------------------------------------------------------------------------------------
-- Association - Specifies the association to query
--
-- If the Association argument is from an unnormalized list:
--
-- - If the Association is given in named notation:
--
--   Returns An_Identifier representing the formal_parameter_selector_name,
--   generic_formal_parameter_selector_name, or pragma_argument_identifier.
--
--   The Enclosing_Element of the An_Identifier element is the Association
--   argument.
--
-- - If the Association is given in positional notation:
--
--   Returns a Nil_Element.
--
-- If the Association argument is from a Normalized list:
--
-- - Returns A_Defining_Name representing the defining_identifier of the
--   parameter_specification or generic_formal_parameter_declaration.
--   Pragma_argument_association elements are not available in normalized form.
--
-- - The Enclosing_Element of the A_Defining_Name is the
--   parameter_specification or generic_formal_parameter_declaration element.
--
```

```
-- Normalized lists contain artificial ASIS An_Association elements that
-- provide one formal A_Defining_Name => An_Expression pair per
-- association.  These artificial associations are Is_Normalized.  Their
-- component A_Defining_Name elements are not Is_Normalized.
--
-- Appropriate Association_Kinds:
--      A_Parameter_Association
--      A_Generic_Association
--      A_Pragma_Argument_Association
--
-- Returns Element_Kinds:
--      Not_An_Element
--      An_Operator_Symbol
--      A_Defining_Name             -- Is_Normalized(Association)
--      An_Expression               -- not Is_Normalized(Association)
--           Returns Expression_Kinds:
--                 An_Identifier
--
-------------------------------------------------------------------------------------
```

-- 17.22 function Actual_Parameter

```
-------------------------------------------------------------------------------------

    function Actual_Parameter (Association : in Asis.Association)
                          return Asis.Expression;

-------------------------------------------------------------------------------------
-- Association   - Specifies the association to query
--
-- If the Association argument is from an unnormalized list:
--
-- - Returns An_Expression representing:
--
--   o the explicit_actual_parameter of a parameter_association.
--
--   o the explicit_generic_actual_parameter of a generic_association.
--
--   o the name or expression of a pragma_argument_association.
--
-- - The Enclosing_Element of An_Expression is the Association argument.
--
-- If the Association argument is from a Normalized list:
--
-- - If the Association is given explicitly:
--
--   o Returns An_Expression representing:
--
--     + the explicit_actual_parameter of a parameter_association.
--
--     + the explicit_generic_actual_parameter of a generic_association.
--
--   o The Enclosing_Element of An_Expression is the Association argument.
--
-- - If the Association is given by default:
--
--   o Returns An_Expression representing:
--
--     + the corresponding default_expression of the Is_Normalized
--       A_Parameter_Association.
--
--     + the corresponding default_expression or default_name of the
--       Is_Normalized A_Generic_Association.
--
--   o The Enclosing_Element of the An_Expression element is the
--     parameter_specification or generic_formal_parameter_declaration that
--     contains the default_expression or default_name, except for the case when
--     this An_Expression element is an implicit naming expression
--     representing the actual subprogram selected at the place of the
--     instantiation for A_Box_Default.  In the latter case, the Enclosing_Element
--     for such An_Expression is the instantiation.
--
--   o Normalized lists contain artificial ASIS An_Association elements that
--     provide one formal A_Defining_Name => An_Expression pair per
--     association.  These artificial associations are Is_Normalized.
```

```
--     Artificial associations of default associations are
--     Is_Defaulted_Association.  Their component An_Expression elements are
--     not Is_Normalized and are not Is_Defaulted_Association.
--
-- If the argument is A_Pragma_Argument_Association, then this function may
-- return any expression to support implementation-defined pragmas.
--
-- Appropriate Association_Kinds:
--      A_Parameter_Association
--      A_Generic_Association
--      A_Pragma_Argument_Association
--
-- Returns Element_Kinds:
--      An_Expression
--
-------------------------------------------------------------------------------------
```

-- 17.23 function Discriminant_Selector_Names

```
-------------------------------------------------------------------------------------

    function Discriminant_Selector_Names
                (Association : in Asis.Discriminant_Association)
                return Asis.Expression_List;

-------------------------------------------------------------------------------------
-- Association - Specifies the discriminant association to query
--
-- If the Association argument is from an unnormalized list:
--
-- - If the Association is a named discriminant_association:
--
--   o Returns a list of the An_Identifier discriminant_selector_name elements in order
--     of appearance.
--
--   o The Enclosing_Element of the names is the Association argument.
--
-- - If the Association is a positional discriminant_association:
--
--   o Returns a Nil_Element_List.
--
-- If the Association argument is from a Normalized list:
--
-- - Returns a list containing a single A_Defining_Name element representing
--   the defining_identifier of the discriminant_specification.
--
-- - The Enclosing_Element of the A_Defining_Name is the
--   discriminant_specification.
--
-- - Normalized lists contain artificial ASIS An_Association elements that
--   provide one formal A_Defining_Name => An_Expression pair per
--   association.  These artificial associations are Is_Normalized.  Their
--   component A_Defining_Name elements are not Is_Normalized.
--
-- Appropriate Association_Kinds:
--      A_Discriminant_Association
--
-- Returns Element_Kinds:
--      A_Defining_Name             -- Is_Normalized(Association)
--      An_Expression               -- not Is_Normalized(Association)
--           Returns Expression_Kinds:
--                An_Identifier
--
```

```
-------------------------------------------------------------------------------
```
-- 17.24 function Discriminant_Expression
```
-------------------------------------------------------------------------------

    function Discriminant_Expression
                (Association : in Asis.Discriminant_Association)
                return Asis.Expression;

-------------------------------------------------------------------------------
-- Association - Specifies the discriminant_association to query
--
-- If the Association argument is from an unnormalized list:
--
-- - Returns An_Expression representing the expression of the
--   discriminant_association.
--
-- - The Enclosing_Element of An_Expression is the Association argument.
--
-- If the Association argument is from a Normalized list:
--
-- - If the Association is given explicitly:
--
--   o Returns An_Expression representing the expression of the
--     discriminant_association.
--
--   o The Enclosing_Element of An_Expression is the Association argument.
--
-- - If the Association is given by default:
--
--   o Returns An_Expression representing:
--
--     + the corresponding default_expression of the Is_Normalized
--       A_Discriminant_Association.
--
--   o The Enclosing_Element of the An_Expression element is the
--     discriminant_specification that contains the default_expression.
--
-- - Normalized lists contain artificial ASIS An_Association elements that
--   provide one formal A_Defining_Name => An_Expression pair per
--   association.  These artificial associations are Is_Normalized.
--   Artificial associations of default associations are
--   Is_Defaulted_Association.  Their component An_Expression elements are
--   not Is_Normalized and are not Is_Defaulted_Association.
--
-- Appropriate Association_Kinds:
--      A_Discriminant_Association
--
-- Returns Element_Kinds:
--      An_Expression
--
-------------------------------------------------------------------------------
```
-- 17.25 function Is_Normalized
```
-------------------------------------------------------------------------------

    function Is_Normalized (Association : in Asis.Association) return Boolean;

-------------------------------------------------------------------------------
-- Association - Specifies the association to query
--
-- Returns True if the association is a normalized, artificially created
-- association returned by the queries Discriminant_Associations,
-- Generic_Actual_Part, Call_Statement_Parameters,
-- Record_Component_Associations, or Function_Call_Parameters where
-- Normalized => True (or the operation returns Is_Normalized associations
-- even if Normalized => False).  See the Implementation Permissions for
-- these queries.
--
-- Returns False for any unexpected Element.
--
```

```
-- Expected Association_Kinds:
--      A_Discriminant_Association
--      A_Record_Component_Association
--      A_Parameter_Association
--      A_Generic_Association
--
-------------------------------------------------------------------------------------
```

-- 17.26 function Is_Defaulted_Association

```
-------------------------------------------------------------------------------------

    function Is_Defaulted_Association
        (Association : in Asis.Association) return Boolean;

-------------------------------------------------------------------------------------
-- Association - Specifies the association to query
--
-- Returns True if the association is a normalized, artificially created
-- association returned by the queries Discriminant_Associations,
-- Generic_Actual_Part, Record_Component_Associations,
-- Call_Statement_Parameters, or Function_Call_Parameters where
-- Normalized => True (or the operation returns default associations even if
-- Normalized => False) and the association contains a default expression.
-- A default expression is one that is implicitly supplied by the language
-- semantics and that was not explicitly supplied (typed) by the user.
--
-- Returns False for any unexpected Element.
--
-- Expected Association_Kinds:
--      A_Parameter_Association
--      A_Generic_Association
--
--|AN Application Note:
--|AN
--|AN Always returns False for discriminant associations.  Defaulted
--|AN discriminant associations occur only when the discriminant constraint is
--|AN completely missing from a subtype indication.  Consequently, it is not
--|AN possible to obtain a (normalized) discriminant constraint list for such
--|AN subtype indications.  Always returns False for component associations.
--|AN Aggregates cannot have defaulted components.
--
-------------------------------------------------------------------------------------
```

-- 17.27 function Expression_Parenthesized

```
-------------------------------------------------------------------------------------

    function Expression_Parenthesized (Expression : in Asis.Expression)
                                  return Asis.Expression;

-------------------------------------------------------------------------------------
-- Expression  - Specifies the parenthesized expression to query
--
-- Returns the expression within the parenthesis.  This operation unwinds only
-- one set of parenthesis at a time, so the result may itself be
-- A_Parenthesized_Expression.
--
-- A_Parenthesized_Expression kind corresponds only to the (expression)
-- alternative in the syntax notion of primary in Reference Manual 4.4.  For
-- example, an expression of a type_conversion is A_Parenthesized_Expression only
-- if it is similar to the form subtype_mark((expression)) where it has at least
-- one set of its own parenthesis.
--
-- Appropriate Expression_Kinds:
--      A_Parenthesized_Expression
--
-- Returns Element_Kinds:
--      An_Expression
--
```

```
-------------------------------------------------------------------------------------
```
-- 17.28 function Is_Prefix_Call
```
-------------------------------------------------------------------------------------

    function Is_Prefix_Call (Expression : in Asis.Expression) return Boolean;

-------------------------------------------------------------------------------------
-- Expression  - Specifies the function call expression to query
--
-- Returns True if the function call is in prefix form.
--
-- Returns False for any unexpected Element.
--
-- For example,
--
--       Foo (A, B);   -- Returns TRUE
--       "<" (A, B);   -- Returns TRUE
--       ... A < B ... -- Returns FALSE
--
-- Expected Expression_Kinds:
--      A_Function_Call
--
-------------------------------------------------------------------------------------
```
-- 17.29 function Corresponding_Called_Function
```
-------------------------------------------------------------------------------------

    function Corresponding_Called_Function
            (Expression : in Asis.Expression)
                return Asis.Declaration;

-------------------------------------------------------------------------------------
-- Expression  - Specifies the function_call to query
--
-- Returns the declaration of the called function.
--
-- Returns a Nil_Element if the:
--
-- - function_prefix denotes a predefined operator for which the implementation
--   does not provide an artificial function declaration,
--
-- - prefix of the call denotes an access to a function implicit or explicit dereference,
--
-- - argument is a dispatching call.
--
-- If function_prefix denotes an attribute_reference, and if the corresponding
-- attribute is (re)defined by an attribute definition clause, an implementation
-- is encouraged, but not required, to return the definition of the corresponding
-- subprogram whose name is used after "use" in this attribute definition
-- clause. If an implementation cannot return such a subprogram definition, a
-- Nil_Element should be returned. For an attribute reference which is not
-- (re)defined by an attribute definition clause, a Nil_Element should be returned.
--
-- Appropriate Expression_Kinds:
--      A_Function_Call
--
-- Returns Declaration_Kinds:
--      Not_A_Declaration
--      A_Function_Declaration
--      A_Function_Body_Declaration
--      A_Function_Body_Stub
--      A_Function_Renaming_Declaration
--      A_Function_Instantiation
--      A_Formal_Function_Declaration
--      A_Generic_Function_Declaration
--
--|IP Implementation Permissions:
--|IP
--|IP An implementation may choose to return any part of multi-part declarations
--|IP and definitions. Multi-part declaration/definitions can occur for:
--|IP
--|IP    - Subprogram specification in package specification, package body,
--|IP      and subunits (is separate);
--|IP    - Entries in package specification, package body, and subunits
```

```
--|IP      (is separate);
--|IP    - Private type and full type declarations;
--|IP    - Incomplete type and full type declarations; and
--|IP    - Deferred constant and full constant declarations.
--|IP
--|IP No guarantee is made that the element will be the first part or
--|IP that the determination will be made due to any visibility rules.
--|IP An application should make its own analysis for each case based
--|IP on which part is returned.
--|IP
--|IP An implementation can choose whether or not to construct and provide
--|IP artificial implicit declarations for predefined operators.
--
------------------------------------------------------------------------------------
```

-- 17.30 function Function_Call_Parameters

```
------------------------------------------------------------------------------------

    function Function_Call_Parameters (Expression : in Asis.Expression;
                                       Normalized : in Boolean := False)
                                      return Asis.Association_List;

------------------------------------------------------------------------------------
-- Expression  - Specifies the function call expression to query
-- Normalized  - Specifies whether the normalized form is desired
--
-- Returns a list of parameter_association elements of the call.
--
-- Returns a Nil_Element_List if there are no parameter_association elements.
--
-- An unnormalized list contains only explicit associations ordered as they
-- appear in the program text.  Each unnormalized association has an optional
-- formal_parameter_selector_name and an explicit_actual_parameter component.
--
-- A normalized list contains artificial associations representing all
-- explicit and default associations.  It has a length equal to the number of
-- parameter_specification elements of the formal_part of the
-- parameter_and_result_profile.  The order of normalized associations matches
-- the order of parameter_specification elements.
--
-- Each normalized association represents a one on one mapping of a
-- parameter_specification elements to the explicit or default expression.
-- A normalized association has one A_Defining_Name component that denotes the
-- parameter_specification, and one An_Expression component that is either the
-- explicit_actual_parameter or a default_expression.
--
-- If the prefix of the call denotes an access to a function implicit or
-- explicit deference, normalized associations are constructed on the basis
-- of the formal_part of the parameter_and_result_profile from the
-- corresponding access_to_subprogram definition.

-- Returns Nil_Element for normalized associations in the case where
-- the called function can be determined only dynamically (dispatching
-- calls). ASIS cannot produce any meaningful result in this case.

-- The exception ASIS_Inappropriate_Element is raised when the function
-- call is an attribute reference and Is_Normalized is True.
--
-- Appropriate Expression_Kinds:
--      A_Function_Call
--
-- Returns Element_Kinds:
--      A_Parameter_Association
--
--|IR Implementation Requirements:
--|IR
--|IR Normalized associations are Is_Normalized and Is_Part_Of_Implicit.
--|IR Normalized associations provided by default are Is_Defaulted_Association.
--|IR Normalized associations are never Is_Equal to unnormalized associations.
--
--|IP Implementation Permissions:
--|IP
--|IP An implementation may choose to always include default parameters in its
--|IP internal representation.
```

```
--|IP
--|IP An implementation may also choose to normalize its representation
--|IP to use defining_identifier elements rather than formal_parameter_selector_name
--|IP elements.
--|IP
--|IP In either case, this query will return Is_Normalized associations even if
--|IP Normalized is False, and the query Function_Call_Parameters_Normalized
--|IP will return True.
--
------------------------------------------------------------------------------------
```

-- 17.31 function Short_Circuit_Operation_Left_Expression

```
------------------------------------------------------------------------------------

    function Short_Circuit_Operation_Left_Expression
                (Expression : in Asis.Expression)
                return Asis.Expression;

------------------------------------------------------------------------------------
-- Expression  - Specifies the short circuit operation to query
--
-- Returns the expression preceding the reserved words “and then” or “or else”
-- in the short circuit expression.
--
-- Appropriate Expression_Kinds:
--      An_And_Then_Short_Circuit
--      An_Or_Else_Short_Circuit
--
-- Returns Element_Kinds:
--      An_Expression
--
------------------------------------------------------------------------------------
```

-- 17.32 function Short_Circuit_Operation_Right_Expression

```
------------------------------------------------------------------------------------

    function Short_Circuit_Operation_Right_Expression
                (Expression : in Asis.Expression)
                return Asis.Expression;

------------------------------------------------------------------------------------
-- Expression  - Specifies the short circuit operation to query
--
-- Returns the expression following the reserved words "or else" or "and then"
-- in the short circuit expression.
--
-- Appropriate Expression_Kinds:
--      An_And_Then_Short_Circuit
--      An_Or_Else_Short_Circuit
--
-- Returns Element_Kinds:
--      An_Expression
--
------------------------------------------------------------------------------------
```

-- 17.33 function Membership_Test_Expression

```
------------------------------------------------------------------------------------

    function Membership_Test_Expression (Expression : in Asis.Expression)
                                   return Asis.Expression;

------------------------------------------------------------------------------------
-- Expression  - Specifies the membership test operation to query
--
-- Returns the expression on the left hand side of the membership test.
--
-- Appropriate Expression_Kinds:
--      An_In_Range_Membership_Test
--      A_Not_In_Range_Membership_Test
--      An_In_Type_Membership_Test
--      A_Not_In_Type_Membership_Test
--
-- Returns Element_Kinds:
--      An_Expression
--
```

```
------------------------------------------------------------------------------------------
```
-- 17.34 function Membership_Test_Range
```
------------------------------------------------------------------------------------------

    function Membership_Test_Range
                (Expression : in Asis.Expression)
                return Asis.Range_Constraint;

------------------------------------------------------------------------------------------
-- Expression  - Specifies the membership test operation to query
--
-- Returns the range following the reserved words "in" or "not in" from the
-- membership test.
--
-- Appropriate Expression_Kinds:
--      An_In_Range_Membership_Test
--      A_Not_In_Range_Membership_Test
--
-- Returns Constraint_Kinds:
--      A_Range_Attribute_Reference
--      A_Simple_Expression_Range
--
------------------------------------------------------------------------------------------
```
-- 17.35 function Membership_Test_Subtype_Mark
```
------------------------------------------------------------------------------------------

    function Membership_Test_Subtype_Mark
                (Expression : in Asis.Expression)
                return Asis.Expression;

------------------------------------------------------------------------------------------
-- Expression  - Specifies the membership test operation to query
--
-- Returns the subtype_mark expression following the reserved words "in" or
-- "not in" from the membership test.
--
-- Appropriate Expression_Kinds:
--      An_In_Type_Membership_Test
--      A_Not_In_Type_Membership_Test
--
-- Returns Expression_Kinds:
--      An_Identifier
--      A_Selected_Component
--      An_Attribute_Reference
--
------------------------------------------------------------------------------------------
```
-- 17.36 function Converted_Or_Qualified_Subtype_Mark
```
------------------------------------------------------------------------------------------

    function Converted_Or_Qualified_Subtype_Mark
                   (Expression : in Asis.Expression)
                       return Asis.Expression;

------------------------------------------------------------------------------------------
-- Expression  - Specifies the type conversion or qualified expression to query.
--
-- Returns the subtype_mark expression that converts or qualifies the
-- expression.
--
-- Appropriate Expression_Kinds:
--      A_Type_Conversion
--      A_Qualified_Expression
--
-- Returns Expression_Kinds:
--      An_Identifier
--      A_Selected_Component
--      An_Attribute_Reference
--
```

```
-------------------------------------------------------------------------------------
```
-- 17.37 function Converted_Or_Qualified_Expression
```
-------------------------------------------------------------------------------------

    function Converted_Or_Qualified_Expression
                   (Expression : in Asis.Expression)
                       return Asis.Expression;

-------------------------------------------------------------------------------------
-- Expression  - Specifies the type conversion or qualified expression to query
--
-- Returns the expression being converted or qualified.
--
-- Appropriate Expression_Kinds:
--      A_Type_Conversion
--      A_Qualified_Expression
--
-- Returns Element_Kinds:
--      An_Expression
--
-------------------------------------------------------------------------------------
```
-- 17.38 function Allocator_Subtype_Indication
```
-------------------------------------------------------------------------------------

    function Allocator_Subtype_Indication (Expression : in Asis.Expression)
                                           return Asis.Subtype_Indication;

-------------------------------------------------------------------------------------
-- Expression  - Specifies the allocator expression to query
--
-- Returns the subtype indication for the object being allocated.
--
-- Appropriate Expression_Kinds:
--      An_Allocation_From_Subtype
--
-- Returns Definition_Kinds:
--      A_Subtype_Indication
--
-------------------------------------------------------------------------------------
```
-- 17.39 function Allocator_Qualified_Expression
```
-------------------------------------------------------------------------------------

    function Allocator_Qualified_Expression (Expression : in Asis.Expression)
                                             return Asis.Expression;

-------------------------------------------------------------------------------------
-- Expression  - Specifies the allocator expression to query
--
-- Returns the qualified expression for the object being allocated.
--
-- Appropriate Expression_Kinds:
--      An_Allocation_From_Qualified_Expression
--
-- Returns Expression_Kinds:
--      A_Qualified_Expression
--
```

```
-----------------------------------------------------------------------------------------
end Asis.Expressions;
```

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

-- 18 package Asis.Statements

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
package Asis.Statements is
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- Asis.Statements encapsulates a set of queries that operate on A_Statement,
-- A_Path, and An_Exception_Handler elements.
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

-- 18.1 function Label_Names

```
-------------------------------------------------------------------------------

    function Label_Names (Statement : in Asis.Statement)
                         return Asis.Defining_Name_List;

-------------------------------------------------------------------------------
-- Statement   - Specifies the statement to query
--
-- Returns label_statement_identifier elements (A_Defining_Name elements) that
-- define the labels attached to the statement, in their order of appearance.
--
-- Returns a Nil_Element_List if there are no labels attached to the statement.
--
-- The Enclosing_Element of the A_Defining_Name elements is the statement.
--
-- Appropriate Element_Kinds:
--      A_Statement
--
-- Returns Defining_Name_Kinds:
--      A_Defining_Identifier
--
-------------------------------------------------------------------------------
```

-- 18.2 function Assignment_Variable_Name

```
-------------------------------------------------------------------------------

    function Assignment_Variable_Name (Statement : in Asis.Statement)
                                      return Asis.Expression;

-------------------------------------------------------------------------------
-- Statement   - Specifies the assignment statement to query
--
-- Returns the expression that names the left hand side of the assignment.
--
-- Appropriate Element_Kinds:
--      A_Statement
--
-- Appropriate Statement_Kinds:
--      An_Assignment_Statement
--
-- Returns Element_Kinds:
--      An_Expression
--
-------------------------------------------------------------------------------
```

-- 18.3 function Assignment_Expression

```
-------------------------------------------------------------------------------

    function Assignment_Expression (Statement : in Asis.Statement)
                                   return Asis.Expression;

-------------------------------------------------------------------------------
-- Statement   - Specifies the assignment statement to query
--
-- Returns the expression from the right hand side of the assignment.
--
-- Appropriate Element_Kinds:
--      A_Statement
--
```

```
-- Appropriate Statement_Kinds:
--      An_Assignment_Statement
--
-- Returns Element_Kinds:
--      An_Expression
```

-- 18.4 function Statement_Paths

```
    function Statement_Paths (Statement : in Asis.Statement;
                              Include_Pragmas : in Boolean := False)
                              return Asis.Path_List;

-------------------------------------------------------------------------------------
-- Statement       - Specifies the statement to query
-- Include_Pragmas - Specifies whether pragmas are to be returned
--
-- Returns a list of the execution paths of the statement, in
-- their order of appearance.
--
-- The only pragmas returned are those preceding the first alternative in
-- a case statement.
--
-- Appropriate Statement_Kinds:
--      An_If_Statement
--      A_Case_Statement
--      A_Selective_Accept_Statement
--      A_Timed_Entry_Call_Statement
--      A_Conditional_Entry_Call_Statement
--      An_Asynchronous_Select_Statement
--
-- Returns Element_Kinds:
--      A_Path
--      A_Pragma
--
```

-- 18.5 function Condition_Expression

```
    function Condition_Expression (Path : in Asis.Path)
                                   return Asis.Expression;

-------------------------------------------------------------------------------------
-- Path - Specifies the execution path to query
--
-- Returns the condition expression for an IF path or an ELSIF path.
--
-- Appropriate Path_Kinds:
--      An_If_Path
--      An_Elsif_Path
--
-- Returns Element_Kinds:
--      An_Expression
--
```

-- 18.6 function Sequence_Of_Statements

```
    function Sequence_Of_Statements (Path            : in Asis.Path;
                                     Include_Pragmas : in Boolean := False)
                                     return Asis.Statement_List;

-------------------------------------------------------------------------------------
-- Path            - Specifies the execution path to query
-- Include_Pragmas - Specifies whether pragmas are to be returned
--
-- Returns a list of the statements and pragmas from an execution path,
-- in their order of appearance.
--
-- Appropriate Element_Kinds:
--      A_Path
```

```
--
-- Returns Element_Kinds:
--      A_Statement
--      A_Pragma
--
-------------------------------------------------------------------------------
```

-- 18.7 function Case_Expression

```
-------------------------------------------------------------------------------

    function Case_Expression (Statement : in Asis.Statement)
                              return Asis.Expression;

-------------------------------------------------------------------------------
-- Statement   - Specifies the case statement to query
--
-- Returns the expression of the case statement that determines which
-- execution path is taken.
--
-- Appropriate Element_Kinds:
--      A_Statement
--
-- Appropriate Statement_Kinds:
--      A_Case_Statement
--
-- Returns Element_Kinds:
--      An_Expression
--
-------------------------------------------------------------------------------
```

-- 18.8 function Case_Statement_Alternative_Choices

```
-------------------------------------------------------------------------------

    function Case_Statement_Alternative_Choices (Path : in Asis.Path)
                                                 return Asis.Element_List;

-------------------------------------------------------------------------------
-- Path - Specifies the case_statement_alternative execution path to query
--
-- Returns a list of the ‘when <choice> | <choice>' elements, in their
-- order of appearance.
--
-- Appropriate Path_Kinds:
--      A_Case_Path
--
-- Returns Element_Kinds:
--      An_Expression
--      A_Definition
--
-- Returns Definition_Kinds:
--      A_Discrete_Range
--      An_Others_Choice
--
-------------------------------------------------------------------------------
```

-- 18.9 function Statement_Identifier

```
-------------------------------------------------------------------------------

    function Statement_Identifier (Statement : in Asis.Statement)
                                   return Asis.Defining_Name;

-------------------------------------------------------------------------------
-- Statement   - Specifies the statement to query
--
-- Returns the identifier for the loop_statement or block_statement.
--
-- Returns a Nil_Element if the loop has no identifier.
--
-- The Enclosing_Element of the name is the statement.
--
-- Appropriate Statement_Kinds:
--      A_Loop_Statement
--      A_While_Loop_Statement
--      A_For_Loop_Statement
--      A_Block_Statement
```

```
--
-- Returns Defining_Name_Kinds:
--      Not_A_Defining_Name
--      A_Defining_Identifier
--
-------------------------------------------------------------------------------------
```

-- 18.10 function Is_Name_Repeated

```
-------------------------------------------------------------------------------------

    function Is_Name_Repeated (Statement : in Asis.Statement) return Boolean;

-------------------------------------------------------------------------------------
-- Statement   - Specifies the statement to query
--
-- Returns True if the name of the accept, loop, or block is repeated after
-- the end of the statement.  Always returns True for loop or block
-- statements since the name is required.
--
-- Returns False for any unexpected Element.
--
-- Expected Statement_Kinds:
--      A_Block_Statement
--      A_Loop_Statement
--      An_Accept_Statement
--
-------------------------------------------------------------------------------------
```

-- 18.11 function While_Condition

```
-------------------------------------------------------------------------------------

    function While_Condition (Statement : in Asis.Statement)
                             return Asis.Expression;

-------------------------------------------------------------------------------------
-- Statement   - Specifies the loop statement to query
--
-- Returns the condition expression associated with the while loop.
--
-- Appropriate Element_Kinds:
--      A_Statement
--
-- Appropriate Statement_Kinds:
--      A_While_Loop_Statement
--
-- Returns Element_Kinds:
--      An_Expression
--
-------------------------------------------------------------------------------------
```

-- 18.12 function For_Loop_Parameter_Specification

```
-------------------------------------------------------------------------------------

    function For_Loop_Parameter_Specification (Statement : in Asis.Statement)
                                              return Asis.Declaration;

-------------------------------------------------------------------------------------
-- Statement   - Specifies the loop statement to query
--
-- Returns the declaration of the A_Loop_Parameter_Specification.
--
-- Appropriate Statement_Kinds:
--      A_For_Loop_Statement
--
-- Returns Declaration_Kinds:
--      A_Loop_Parameter_Specification
--
```

```
-------------------------------------------------------------------------------
```
-- 18.13 function Loop_Statements
```
-------------------------------------------------------------------------------

    function Loop_Statements (Statement       : in Asis.Statement;
                              Include_Pragmas : in Boolean := False)
                              return Asis.Statement_List;

-------------------------------------------------------------------------------
-- Statement       - Specifies the loop statement to query
-- Include_Pragmas - Specifies whether pragmas are to be returned
--
-- Returns the sequence_of_statements and any pragmas from the loop_statement,
-- in their order of appearance.
--
-- Appropriate Statement_Kinds:
--      A_Loop_Statement
--      A_While_Loop_Statement
--      A_For_Loop_Statement
--
-- Returns Element_Kinds:
--      A_Pragma
--      A_Statement
--
-------------------------------------------------------------------------------
```
-- 18.14 function Is_Declare_Block
```
-------------------------------------------------------------------------------

    function Is_Declare_Block (Statement : in Asis.Statement) return Boolean;

-------------------------------------------------------------------------------
-- Statement   - Specifies the statement to query
--
-- Returns True if the statement is a block_statement and it was created with
-- the use of the "declare" reserved word.  The presence or absence of any
-- declarative_item elements is not relevant.
--
-- Returns False if the "declare" reserved word does not appear in the
-- block_statement, or for any unexpected Element.
--
-- Expected Statement_Kinds:
--      A_Block_Statement
--
-------------------------------------------------------------------------------
```
-- 18.15 function Block_Declarative_Items
```
-------------------------------------------------------------------------------

    function Block_Declarative_Items
            (Statement       : in Asis.Statement;
                Include_Pragmas : in Boolean := False)
                return Asis.Declarative_Item_List;

-------------------------------------------------------------------------------
-- Statement       - Specifies the block statement to query
-- Include_Pragmas - Specifies whether pragmas are to be returned
--
-- Returns a list of the declarations, representation_clause elements, pragmas,
-- and use_clause elements in the declarative_part of the block_statement, in their
-- order of appearance.
--
-- Returns a Nil_Element_List if there are no declarative items.
--
-- Appropriate Statement_Kinds:
--      A_Block_Statement
--
-- Returns Element_Kinds:
--      A_Declaration
--      A_Pragma
--      A_Clause
--
```

```
-----------------------------------------------------------------------------------------
```

-- 18.16 function Block_Statements

```
-----------------------------------------------------------------------------------------

    function Block_Statements (Statement       : in Asis.Statement;
                               Include_Pragmas : in Boolean := False)
                               return Asis.Statement_List;

-----------------------------------------------------------------------------------------
-- Statement       - Specifies the block statement to query
-- Include_Pragmas - Specifies whether pragmas are to be returned
--
-- Returns a list of the statements and pragmas for the block_statement, in
-- their order of appearance.
--
-- Returns a Nil_Element_List if there are no statements or pragmas.  This
-- can only occur for a block_statement obtained from the obsolescent query
-- Body_Block_Statement when its argument is a package_body
-- that has no sequence_of_statements.
--
-- Appropriate Statement_Kinds:
--      A_Block_Statement
--
-- Returns Element_Kinds:
--      A_Pragma
--      A_Statement
--
-----------------------------------------------------------------------------------------
```

-- 18.17 function Block_Exception_Handlers

```
-----------------------------------------------------------------------------------------

    function Block_Exception_Handlers (Statement : in Asis.Statement;
                                       Include_Pragmas : in Boolean := False)
                                       return Asis.Exception_Handler_List;

-----------------------------------------------------------------------------------------
-- Statement       - Specifies the block statement to query
-- Include_Pragmas - Specifies whether pragmas are to be returned
--
-- Returns a list of the exception_handler elements of the block_statement, in their
-- order of appearance.
--
-- The only pragmas returned are those following the reserved word "exception"
-- and preceding the reserved word “when” of first exception handler.
--
-- Returns a Nil_Element_List if there are no exception_handler elements.
--
-- Appropriate Statement_Kinds:
--      A_Block_Statement
--
-- Returns Element_Kinds:
--      An_Exception_Handler
--      A_Pragma
--
-----------------------------------------------------------------------------------------
```

-- 18.18 function Exit_Loop_Name

```
-----------------------------------------------------------------------------------------

    function Exit_Loop_Name (Statement : in Asis.Statement)
                             return Asis.Expression;

-----------------------------------------------------------------------------------------
-- Statement   - Specifies the exit statement to query
--
-- Returns the name of the exited loop.
--
-- Returns a Nil_Element if no loop name is present.
--
-- Appropriate Statement_Kinds:
--      An_Exit_Statement
--
```

```
-- Returns Expression_Kinds:
--      Not_An_Expression
--      An_Identifier
--      A_Selected_Component
--
-----------------------------------------------------------------------------------------
```

-- 18.19 function Exit_Condition

```
-----------------------------------------------------------------------------------------

    function Exit_Condition (Statement : in Asis.Statement)
                             return Asis.Expression;

-----------------------------------------------------------------------------------------
-- Statement   - Specifies the exit statement to query
--
-- Returns the "when" condition of the exit statement.
--
-- Returns a Nil_Element if no condition is present.
--
-- Appropriate Statement_Kinds:
--      An_Exit_Statement
--
-- Returns Element_Kinds:
--      Not_An_Element
--      An_Expression
--
-----------------------------------------------------------------------------------------
```

-- 18.20 function Corresponding_Loop_Exited

```
-----------------------------------------------------------------------------------------

    function Corresponding_Loop_Exited (Statement : in Asis.Statement)
                                        return Asis.Statement;

-----------------------------------------------------------------------------------------
-- Statement   - Specifies the exit statement to query
--
-- Returns the loop statement exited by the exit statement.
--
-- Appropriate Statement_Kinds:
--      An_Exit_Statement
--
-- Returns Element_Kinds:
--      A_Loop_Statement
--      A_While_Loop_Statement
--      A_For_Loop_Statement
--
-----------------------------------------------------------------------------------------
```

-- 18.21 function Return_Expression

```
-----------------------------------------------------------------------------------------
    function Return_Expression (Statement : in Asis.Statement)
                                return Asis.Expression;

-----------------------------------------------------------------------------------------
-- Statement   - Specifies the return statement to query
--
-- Returns the expression in the return statement.
--
-- Returns a Nil_Element if no expression is present.
--
-- Appropriate Statement_Kinds:
--      A_Return_Statement
--
-- Returns Element_Kinds:
--      Not_An_Element
--      An_Expression
--
```

```
-------------------------------------------------------------------------------
```

-- 18.22 function Goto_Label

```
-------------------------------------------------------------------------------

    function Goto_Label (Statement : in Asis.Statement)
                         return Asis.Expression;

-------------------------------------------------------------------------------
-- Statement   - Specifies the goto statement to query
--
-- Returns the expression reference for the label, as specified by the goto
-- statement.
--
-- Appropriate Statement_Kinds:
--      A_Goto_Statement
--
-- Returns Expression_Kinds:
--      An_Identifier
--
-------------------------------------------------------------------------------
```

-- 18.23 function Corresponding_Destination_Statement

```
-------------------------------------------------------------------------------

    function Corresponding_Destination_Statement
       (Statement : in Asis.Statement)
        return Asis.Statement;

-------------------------------------------------------------------------------
-- Statement  - Specifies the goto statement to query
--
-- Returns the target statement specified by the goto statement.
--
-- Appropriate Statement_Kinds:
--      A_Goto_Statement
--
-- Returns Element_Kinds:
--      A_Statement
--
--|AN Application Note:
--|AN
--|AN The Reference Manual allows a pragma between a statement and a label attached
--|AN to it. If so, when the label is passed as an actual parameter to
--|AN this query, the query returns the statement, but not the label. The only way
--|AN for an application to know that there are any pragmas between a statement
--|AN and its label is to get the spans of these program elements and analyze the
--|AN corresponding positions in the source text.
--
-------------------------------------------------------------------------------
```

-- 18.24 function Called_Name

```
-------------------------------------------------------------------------------

    function Called_Name (Statement : in Asis.Statement)
                          return Asis.Expression;

-------------------------------------------------------------------------------
-- Statement   - Specifies the procedure call or entry call statement to query
--
-- Returns the name of the called procedure or entry.  The name of an entry
-- family takes the form of An_Indexed_Component.
--
-- Appropriate Statement_Kinds:
--      An_Entry_Call_Statement
--      A_Procedure_Call_Statement
--
-- Returns Element_Kinds:
--      An_Expression
--
```

```
-------------------------------------------------------------------------------
```

-- 18.25 function Corresponding_Called_Entity

```
-------------------------------------------------------------------------------

    function Corresponding_Called_Entity (Statement : in Asis.Statement)
                                          return Asis.Declaration;

-------------------------------------------------------------------------------
-- Statement   - Specifies the procedure_call_statement or
--               entry_call_statement to query
--
-- Returns the declaration of the procedure or entry denoted by the call.
--
-- Returns a Nil_Element if the:
--
-- - prefix of the call denotes an access to a procedure implicit
--   or explicit dereference,
--
-- - argument is a dispatching call,
--
-- - argument is a call to a dispatching operation of a tagged type which
--   is not statically determined.
--
-- If procedure_prefix denotes an attribute_reference, and if the corresponding
-- attribute is (re)defined by an attribute definition clause, an implementation
-- is encouraged, but not required, to return the definition of the corresponding
-- subprogram whose name is used after "use" in this attribute definition
-- clause. If an implementation cannot return such a subprogram definition, a
-- Nil_Element should be returned. For an attribute reference which is not
-- (re)defined by an attribute definition clause, a Nil_Element should be returned.
--
-- Appropriate Statement_Kinds:
--      An_Entry_Call_Statement
--      A_Procedure_Call_Statement
--
-- Returns Declaration_Kinds:
--      Not_A_Declaration
--      A_Procedure_Declaration
--      A_Procedure_Body_Declaration
--      A_Procedure_Body_Stub
--      A_Procedure_Renaming_Declaration
--      A_Procedure_Instantiation
--      A_Formal_Procedure_Declaration
--      An_Entry_Declaration
--      A_Generic_Procedure_Declaration
--
--|IP Implementation Permissions
--|IP
--|IP An implementation may choose to return any part of multi-part
--|IP declarations and definitions. Multi-part declaration/definitions
--|IP can occur for:
--|IP
--|IP    - Subprogram specification in package specification, package body,
--|IP      and subunits (is separate);
--|IP    - Entries in package specification, package body, and subunits
--|IP      (is separate);
--|IP    - Private type and full type declarations;
--|IP    - Incomplete type and full type declarations; and
--|IP    - Deferred constant and full constant declarations.
--|IP
--|IP No guarantee is made that the element will be the first part or
--|IP that the determination will be made due to any visibility rules.
--|IP An application should make its own analysis for each case based
--|IP on which part is returned.
--
```

-- 18.26 function Call_Statement_Parameters

```
    function Call_Statement_Parameters (Statement  : in Asis.Statement;
                                        Normalized : in Boolean := False)
                                        return Asis.Association_List;

-------------------------------------------------------------------------------------
-- Statement   - Specifies the procedure_call_statement or
--               entry_call_statement to query
-- Normalized  - Specifies whether the normalized form is desired
--
-- Returns a list of parameter_association elements of the call.
--
-- Returns a Nil_Element_List if there are no parameter_association elements.
--
-- An unnormalized list contains only explicit associations ordered as they
-- appear in the program text.  Each unnormalized association has an optional
-- formal_parameter_selector_name and an explicit_actual_parameter component.
--
-- A normalized list contains artificial associations representing all
-- explicit and default associations.  It has a length equal to the number of
-- parameter_specification elements of the formal_part of the
-- parameter_and_result_profile.  The order of normalized associations matches
-- the order of parameter_specification elements.
--
-- Each normalized association represents a one on one mapping of a
-- parameter_specification elements to the explicit or default expression.
-- A normalized association has one A_Defining_Name component that denotes the
-- parameter_specification, and one An_Expression component that is either the
-- explicit_actual_parameter or a default_expression.
--
-- If the prefix of the call denotes an access to a procedure implicit or
-- explicit deference, normalized associations are constructed on the basis
-- of the formal_part of the parameter_profile from the corresponding
-- access_to_subprogram definition.
--
-- Returns Nil_Element for normalized associations in the case where
-- the called procedure can be determined only dynamically (dispatching
-- calls). ASIS cannot produce any meaningful result in this case.
--
-- The exception ASIS_Inappropriate_Element is raised when the procedure
-- call is an attribute reference and Is_Normalized is True.
--
-- Appropriate Statement_Kinds:
--      An_Entry_Call_Statement
--      A_Procedure_Call_Statement
--
-- Returns Element_Kinds:
--      A_Parameter_Association
--
--|IR Implementation Requirements:
--|IR
--|IR Normalized associations are Is_Normalized and Is_Part_Of_Implicit.
--|IR Normalized associations provided by default are Is_Defaulted_Association.
--|IR Normalized associations are never Is_Equal to unnormalized associations.
--
--|IP Implementation Permissions:
--|IP
--|IP An implementation may choose to always include default parameters in its
--|IP internal representation.
--|IP
--|IP An implementation may also choose to normalize its representation
--|IP to use defining_identifier elements rather than formal_parameter_selector_name
--|IP elements.
--|IP
--|IP In either case, this query will return Is_Normalized associations even if
--|IP Normalized is False, and the query Call_Statement_Parameters_Normalized
--|IP will return True.
--
```

```
-------------------------------------------------------------------------------
```
-- 18.27 function Accept_Entry_Index
```
-------------------------------------------------------------------------------

    function Accept_Entry_Index (Statement : in Asis.Statement)
                                 return Asis.Expression;

-------------------------------------------------------------------------------
-- Statement   - Specifies the accept statement to query
--
-- Returns the entry index expression in the accept statement.
--
-- Returns a Nil_Element if the statement has no explicit entry index,
--
-- Appropriate Statement_Kinds:
--      An_Accept_Statement
--
-- Returns Element_Kinds:
--      Not_An_Element
--      An_Expression
--
-------------------------------------------------------------------------------
```
-- 18.28 function Accept_Entry_Direct_Name
```
-------------------------------------------------------------------------------

    function Accept_Entry_Direct_Name (Statement : in Asis.Statement)
                                       return Asis.Name;

-------------------------------------------------------------------------------
-- Statement   - Specifies the accept statement to query
--
-- Returns the direct name of the entry.  The name follows the reserved word "accept".
--
-- Appropriate Statement_Kinds:
--      An_Accept_Statement
--
-- Returns Expression_Kinds:
--      An_Identifier
--
-------------------------------------------------------------------------------
```
-- 18.29 function Accept_Parameters
```
-------------------------------------------------------------------------------

    function Accept_Parameters (Statement : in Asis.Statement)
                                return Asis.Parameter_Specification_List;

-------------------------------------------------------------------------------
-- Statement   - Specifies the accept statement to query
--
-- Returns a list of parameter specifications in the formal part of the accept
-- statement, in their order of appearance.
--
-- Returns a Nil_Element_List if the accept_statement has no parameters.
--
-- Results of this query may vary across ASIS implementations.  Some
-- implementations normalize all multiple name parameter specifications into an
-- equivalent sequence of corresponding single name parameter specifications.
-- See Reference Manual 3.3.1(7).
--
-- Appropriate Statement_Kinds:
--      An_Accept_Statement
--
-- Returns Declaration_Kinds:
--      A_Parameter_Specification
--
```

```
-------------------------------------------------------------------------------
```
-- 18.30 function Accept_Body_Statements
```
-------------------------------------------------------------------------------
    function Accept_Body_Statements (Statement       : in Asis.Statement;
                                     Include_Pragmas : in Boolean := False)
                                     return Asis.Statement_List;

-------------------------------------------------------------------------------
-- Statement       - Specifies the accept statement to query
-- Include_Pragmas - Specifies whether pragmas are to be returned
--
-- Returns the list of statements and pragmas from the body of the accept
-- statement, in their order of appearance.
--
-- Appropriate Statement_Kinds:
--      An_Accept_Statement
--
-- Returns Element_Kinds:
--      A_Pragma
--      A_Statement
--
-------------------------------------------------------------------------------
```
-- 18.31 function Accept_Body_Exception_Handlers
```
-------------------------------------------------------------------------------

    function Accept_Body_Exception_Handlers
                 (Statement       : in Asis.Statement;
                  Include_Pragmas : in Boolean := False)
                  return Asis.Statement_List;

-------------------------------------------------------------------------------
-- Statement       - Specifies the accept statement to query
-- Include_Pragmas - Specifies whether pragmas are to be returned
--
-- Returns the list of exception handlers and pragmas from the body of the
-- accept statement, in their order of appearance.
--
-- Appropriate Statement_Kinds:
--      An_Accept_Statement
--
-- Returns Element_Kinds:
--      A_Pragma
--      An_Exception_Handler
--
-------------------------------------------------------------------------------
```
-- 18.32 function Corresponding_Entry
```
-------------------------------------------------------------------------------

    function Corresponding_Entry (Statement : in Asis.Statement)
                                  return Asis.Declaration;

-------------------------------------------------------------------------------
-- Statement   - Specifies the accept statement to query
--
-- Returns the declaration of the entry accepted in this statement.
--
-- Appropriate Statement_Kinds:
--      An_Accept_Statement
--
-- Returns Declaration_Kinds:
--      An_Entry_Declaration
--
```

```
-------------------------------------------------------------------------------------
```

-- 18.33 function Requeue_Entry_Name

```
-------------------------------------------------------------------------------------

    function Requeue_Entry_Name (Statement : in Asis.Statement)
                                 return Asis.Name;

-------------------------------------------------------------------------------------
-- Statement   - Specifies the requeue statement to query
--
-- Returns the name of the entry requeued by the statement.
-- The name follows the reserved word "requeue".
--
-- Appropriate Statement_Kinds:
--      A_Requeue_Statement
--      A_Requeue_Statement_With_Abort
--
-- Returns Element_Kinds:
--      An_Expression
--
-------------------------------------------------------------------------------------
```

-- 18.34 function Delay_Expression

```
-------------------------------------------------------------------------------------

    function Delay_Expression (Statement : in Asis.Statement)
                               return Asis.Expression;

-------------------------------------------------------------------------------------
-- Statement   - Specifies the delay statement to query
--
-- Returns the expression for the duration of the delay.
--
-- Appropriate Statement_Kinds:
--      A_Delay_Until_Statement
--      A_Delay_Relative_Statement
--
-- Returns Element_Kinds:
--      An_Expression
--
-------------------------------------------------------------------------------------
```

-- 18.35 function Guard

```
-------------------------------------------------------------------------------------

    function Guard (Path : in Asis.Path)
                    return Asis.Expression;

-------------------------------------------------------------------------------------
-- Path - Specifies the select statement execution path to query
--
-- Returns the conditional expression guard for the path.
--
-- Returns a Nil_Element if there is no guard, or if the path is from a
-- timed_entry_call, a conditional_entry_call, or an asynchronous_select
-- statement where a guard is not legal.
--
-- Appropriate Path_Kinds:
--      A_Select_Path
--      An_Or_Path
--
-- Returns Element_Kinds:
--      Not_An_Element
--      An_Expression
--
```

```
---------------------------------------------------------------------------------------
```

-- 18.36 function Aborted_Tasks

```
---------------------------------------------------------------------------------------

    function Aborted_Tasks (Statement : in Asis.Statement)
                            return Asis.Expression_List;

---------------------------------------------------------------------------------------
-- Statement    - Specifies the abort statement to query
--
-- Returns a list of the task names from the ABORT statement, in their order
-- of appearance.
--
-- Appropriate Statement_Kinds:
--      An_Abort_Statement
--
-- Returns Element_Kinds:
--      An_Expression
--
---------------------------------------------------------------------------------------
```

-- 18.37 function Choice_Parameter_Specification

```
---------------------------------------------------------------------------------------

    function Choice_Parameter_Specification
       (Handler : in Asis.Exception_Handler)
        return Asis.Declaration;

---------------------------------------------------------------------------------------
-- Handler - Specifies the exception handler to query
--
-- Returns the choice parameter specification following the reserved word "when"
-- in the exception handler.
--
-- Returns a Nil_Element if there is no explicit choice parameter.
--
-- Appropriate Element_Kinds:
--    An_Exception_Handler
--
-- Returns Declaration_Kinds:
--      Not_A_Declaration
--      A_Choice_Parameter_Specification
--
---------------------------------------------------------------------------------------
```

-- 18.38 function Exception_Choices

```
---------------------------------------------------------------------------------------

    function Exception_Choices (Handler : in Asis.Exception_Handler)
                                return Asis.Element_List;

---------------------------------------------------------------------------------------
-- Handler - Specifies the exception handler to query
--
-- Returns a list of the 'when <choice> | <choice>' elements, in their
-- order of appearance.  Choices are either the exception name expression or
-- an others choice.
--
-- Appropriate Element_Kinds:
--    An_Exception_Handler
--
-- Returns Expression_Kinds:
--      An_Identifier
--      A_Selected_Component
--
-- Returns Definition_Kinds:
--      An_Others_Choice
--
```

```
-------------------------------------------------------------------------------------
```
-- 18.39 function Handler_Statements
```
-------------------------------------------------------------------------------------

    function Handler_Statements (Handler         : in Asis.Exception_Handler;
                                 Include_Pragmas : in Boolean := False)
                                 return Asis.Statement_List;

-------------------------------------------------------------------------------------
-- Handler         - Specifies the exception handler to query
-- Include_Pragmas - Specifies whether pragmas are to be returned
--
-- Returns the list of statements and pragmas from the body of the
-- exception handler, in their order of appearance.
--
-- Appropriate Element_Kinds:
--    An_Exception_Handler
--
-- Returns Element_Kinds:
--      A_Pragma
--      A_Statement
--
-------------------------------------------------------------------------------------
```
-- 18.40 function Raised_Exception
```
-------------------------------------------------------------------------------------

    function Raised_Exception (Statement : in Asis.Statement)
                               return Asis.Expression;

-------------------------------------------------------------------------------------
-- Statement - Specifies the raise statement to query
--
-- Returns the expression that names the raised exception.
--
-- Returns a Nil_Element if there is no explicitly named exception.
--
-- Appropriate Statement_Kinds:
--      A_Raise_Statement
--
-- Returns Expression_Kinds:
--      Not_An_Expression
--      An_Identifier
--      A_Selected_Component
--
-------------------------------------------------------------------------------------
```
-- 18.41 function Qualified_Expression
```
-------------------------------------------------------------------------------------

    function Qualified_Expression (Statement : in Asis.Statement)
                                   return Asis.Expression;

-------------------------------------------------------------------------------------
-- Statement  - Specifies the code statement to query
--
-- Returns the qualified aggregate expression representing the code statement.
--
-- Appropriate Statement_Kinds:
--      A_Code_Statement
--
-- Returns Expression_Kinds:
--      A_Qualified_Expression
--
```

```
-----------------------------------------------------------------------------------------
```

-- 18.42 function Is_Dispatching_Call

```
-----------------------------------------------------------------------------------------

    function Is_Dispatching_Call (Call : in Asis.Element) return Boolean;

-----------------------------------------------------------------------------------------
-- Call - Specifies the element to query.
--
-- Returns True if the controlling tag of Call is dynamically determined.
--
-- This function shall always return False when pragma Restrictions(No_Dispatch)
-- applies.
--
-- Returns False for any unexpected Element.
--
-- Expected Expression_Kinds:
--    A_Function_Call
--
-- Expected Statement_Kinds:
--    A_Procedure_Call_Statement
--
-----------------------------------------------------------------------------------------
```

-- 18.43 function Is_Call_On_Dispatching_Operation

```
-----------------------------------------------------------------------------------------

    function Is_Call_On_Dispatching_Operation (Call : in Asis.Element)
                                               return Boolean;

-----------------------------------------------------------------------------------------
-- Call - Specifies the element to query.
--
-- Returns True if the name or prefix of Call denotes the declaration of a
-- primitive operation of a tagged type.
--
-- Returns False for any unexpected Element.
--
-- Expected Element_Kinds:
--    A_Function_Call
--    A_Procedure_Call_Statement
--
-----------------------------------------------------------------------------------------

end Asis.Statements;
```

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

-- 19 package Asis.Clauses

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
package Asis.Clauses is
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- Asis.Clauses
--
-- This package encapsulates a set of queries that operate on A_Clause
-- elements.
--
--|ER---------------------------------------------------------------------------
--|ER A_Use_Package_Clause - 8.4
--|ER A_Use_Type_Clause    - 8.4
--|ER A_With_Clause        - 10.1.2
--|CR
--|CR Child elements returned by:
--|CR    function Clause_Names
--|CR
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

-- 19.1 function Clause_Names

```
-------------------------------------------------------------------------------

    function Clause_Names (Clause : in Asis.Element)
                           return Asis.Name_List;

-------------------------------------------------------------------------------
-- Clause  - Specifies the with_clause or use_clause to query
--
-- Returns a list of the names that appear in the given clause.
-- The names in the list should be in their order of appearance in the
-- original clauses from the compilation text.
--
-- Results of this query may vary across ASIS implementations.  Some
-- implementations normalize all clauses containing multiple names
-- into an equivalent sequence of corresponding single clauses.
-- Similarly, an implementation may keep a name only once even though that
-- name can appear more than once in a clause.
--
-- Appropriate Element_Kinds:
--      A_Use_Package_Clause
--      A_Use_Type_Clause
--      A_With_Clause
--
-- Returns Expression_Kinds:
--      An_Identifier
--      A_Selected_Component
--      An_Attribute_Reference
--
--|ER---------------------------------------------------------------------------
--|ER A_Representation_Clause - 13.1
--|ER---------------------------------------------------------------------------
--|ER An_Attribute_Definition_Clause - 13.3
--|ER An_Enumeration_Representation_Clause - 13.4
--|ER An_At_Clause - J.7
--|CR
--|CR Child elements returned by:
--|CR    function Representation_Clause_Name
--|CR    function Representation_Clause_Expression
--
-------------------------------------------------------------------------------
```

-- 19.2 function Representation_Clause_Name

```
-------------------------------------------------------------------------------

    function Representation_Clause_Name (Clause : in Asis.Clause)
                                         return Asis.Name;

-------------------------------------------------------------------------------
-- Clause  - Specifies the representation_clause to query
```

```
--
-- Returns the direct_name expression following the reserved word "for".
--
-- Appropriate Clause_Kinds:
--      A_Representation_Clause
--      A_Component_Clause
--
-- Returns Expression_Kinds:
--      An_Identifier
--      An_Attribute_Reference
--
-------------------------------------------------------------------------------------
```

-- 19.3 function Representation_Clause_Expression

```
-------------------------------------------------------------------------------------

    function Representation_Clause_Expression
                (Clause : in Asis.Representation_Clause)
                 return Asis.Expression;

-------------------------------------------------------------------------------------
-- Clause  - Specifies the representation_clause to query
--
-- Returns the expression following the reserved word “use” or the reserved
-- words “use at”.
--
-- Appropriate Representation_Clause_Kinds:
--      An_Attribute_Definition_Clause
--      An_Enumeration_Representation_Clause
--      An_At_Clause
--
-- Returns Element_Kinds:
--      An_Expression
--
--|ER---------------------------------------------------------------------------------
--|ER A_Record_Representation_Clause - 13.5.1
--|CR
--|CR Child elements returned by:
--|CR    function Representation_Clause_Name
--|CR    function Mod_Clause_Expression
--|CR    function Component_Clauses
--
-------------------------------------------------------------------------------------
```

-- 19.4 function Mod_Clause_Expression

```
-------------------------------------------------------------------------------------

    function Mod_Clause_Expression (Clause : in Asis.Representation_Clause)
                             return Asis.Expression;

-------------------------------------------------------------------------------------
-- Clause  - Specifies the record representation clause to query
--
-- Returns the static_expression appearing after the reserved words "at mod".
--
-- Returns a Nil_Element if a mod_clause is not present.
--
-- Appropriate Representation_Clause_Kinds:
--      A_Record_Representation_Clause
--
-- Returns Element_Kinds:
--      Not_An_Element
--      An_Expression
--
-------------------------------------------------------------------------------------
```

-- 19.5 function Component_Clauses

```
-------------------------------------------------------------------------------------

    function Component_Clauses (Clause : in Asis.Representation_Clause;
                            Include_Pragmas : in Boolean := False)
                            return Asis.Component_Clause_List;

-------------------------------------------------------------------------------------
-- Clause          - Specifies the record representation clause to query
```

```
-- Include_Pragmas - Specifies whether pragmas are to be returned
--
-- Returns the component_clause and pragma elements from the
-- record_representation_clause, in their order of appearance.
--
-- Returns a Nil_Element_List if the record_representation_clause has no
-- component_clause or pragma elements.
--
-- Appropriate Representation_Clause_Kinds:
--      A_Record_Representation_Clause
--
-- Returns Element_Kinds:
--      A_Clause
--      A_Pragma
--
-- Returns Clause_Kinds:
--      A_Component_Clause
--
--|ER----------------------------------------------------------------------------------
--|ER A_Component_Clause - 13.5.1
--|CR
--|CR Child elements returned by:
--|CR    function Representation_Clause_Name
--|CR    function Component_Clause_Position
--|CR    function Component_Clause_Range
--
-------------------------------------------------------------------------------------
```

-- 19.6 function Component_Clause_Position

```
-------------------------------------------------------------------------------------

    function Component_Clause_Position (Clause : in Asis.Component_Clause)
                                    return Asis.Expression;

-------------------------------------------------------------------------------------
-- Clause  - Specifies the component_clause to query
--
-- Returns the position expression for the component_clause.
--
-- Appropriate Clause_Kinds:
--      A_Component_Clause
--
-- Returns Element_Kinds:
--      An_Expression
--
-------------------------------------------------------------------------------------
```

-- 19.7 function Component_Clause_Range

```
-------------------------------------------------------------------------------------

    function Component_Clause_Range (Clause : in Asis.Component_Clause)
                                 return Asis.Discrete_Range;

-------------------------------------------------------------------------------------
-- Clause  - Specifies the component_clause to query
--
-- Returns the first_bit .. last_bit range for the component_clause.
--
-- Appropriate Clause_Kinds:
--      A_Component_Clause
--
-- Returns Discrete_Range_Kinds:
--      A_Discrete_Simple_Expression_Range
--
-------------------------------------------------------------------------------------

end Asis.Clauses;
```

```
-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
```

-- 20 package Asis.Text

```
-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
package Asis.Text is
-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
-- Asis.Text
--
-- This package encapsulates a set of operations to access the text of ASIS
-- Elements.  It assumes no knowledge of the existence, location, or form of
-- the program text.
--
-- The text of a program consists of the texts of one or more compilations.
-- The text of each compilation is a sequence of separate lexical elements.
-- Each lexical element is either a delimiter, an identifier (which can be a
-- reserved word), a numeric literal, a character literal, a string literal,
-- blank space, or a comment.
--
-- Each ASIS Element has a text image whose value is the series of characters
-- contained by the text span of the Element.  The text span covers all the
-- characters from the first character of the Element through the last
-- character of the Element over some range of lines.
--
-- General Usage Rules:
--
-- Line lists can be indexed to obtain individual lines.  The bounds of each
-- list correspond to the lines with those same numbers from the compilation
-- text.
--
-- Any Asis.Text query may raise ASIS_Failed with a Status of Text_Error if
-- the program text cannot be located or retrieved for any reason such as
-- renaming, deletion, corruption, or moving of the text.
-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
```

-- 20.1 type Line

```
-------------------------------------------------------------------------------------
-- An Ada text line abstraction (a private type).
--
-- Used to represent text fragments from a compilation.
-- ASIS Lines are representations of the compilation text.
-- This shall be supported by all ASIS implementations.
-------------------------------------------------------------------------------------

    type Line is private;
    Nil_Line  : constant Line;

    function "=" (Left  : in Line;
                  Right : in Line)
                  Return Boolean is abstract;

-- Nil_Line is the value of an uninitialized Line object.
--
-------------------------------------------------------------------------------------
```

-- 20.2 type Line_Number

```
-------------------------------------------------------------------------------------
-- Line_Number is a numeric subtype that allows each ASIS implementation to place
-- constraints on the upper bound for Line_List elements and compilation unit size.
--
-- The upper bound of Line_Number (Maximum_Line_Number) is the only
-- allowed variation for these declarations.
--
-- Line_Number = 0 is reserved to act as an "invalid" Line_Number value.  No
-- unit text line will ever have a Line_Number of zero.
-------------------------------------------------------------------------------------
-- Line shall be an undiscriminated private type, or, shall be derived from an
-- undiscriminated private type.  It can be declared as a new type or as a
-- subtype of an existing type.
-------------------------------------------------------------------------------------
```

```
    Maximum_Line_Number : constant ASIS_Natural :=
        Implementation_Defined_Integer_Constant;

    subtype Line_Number is ASIS_Natural range 0 .. Maximum_Line_Number;

-------------------------------------------------------------------------------
```

-- 20.3 type Line_Number_Positive

```
-------------------------------------------------------------------------------

    subtype Line_Number_Positive is Line_Number range 1 .. Maximum_Line_Number;

-------------------------------------------------------------------------------
```

-- 20.4 type Line_List

```
-------------------------------------------------------------------------------

    type Line_List is array (Line_Number_Positive range <>) of Line;
    Nil_Line_List : constant Line_List;

-------------------------------------------------------------------------------
```

-- 20.5 type Character_Position

```
-------------------------------------------------------------------------------
-- Character_Position is a numeric subtype that allows each ASIS implementation to
-- place constraints on the upper bound for Character_Position and for compilation
-- unit line lengths.
--
-- The upper bound of Character_Position (Maximum_Line_Length) is the
-- only allowed variation for these declarations.
--
-- Character_Position = 0 is reserved to act as an "invalid"
-- Character_Position value.  No unit text line will ever have a character in
-- position zero.
-------------------------------------------------------------------------------

    Maximum_Line_Length : constant ASIS_Natural :=
        Implementation_Defined_Integer_Constant;

    subtype Character_Position is ASIS_Natural range 0 .. Maximum_Line_Length;

-------------------------------------------------------------------------------
```

-- 20.6 type Character_Position_Positive

```
-------------------------------------------------------------------------------

    subtype Character_Position_Positive is
       Character_Position range 1 .. Maximum_Line_Length;

-------------------------------------------------------------------------------
```

-- 20.7 type Span

```
-------------------------------------------------------------------------------
-- Span is a single text position that is identified by a line number and a column
-- number representing the text's position within the compilation unit.
--
-- The text of an element can span one or more lines.  The textual Span of an
-- element identifies the lower and upper bound of a span of text positions.
--
-- Spans and positions give client tools the option of accessing compilation
-- unit text through the queries provided by this package, or, to access
-- the text directly through the original compilation unit text file. Type span
-- facilitates the capture of comments before or after an element.
--
-- Note: The original compilation unit text may or may not have existed in a
-- "file", and any such file may or may not still exist.  Reference Manual 10.1
-- specifies that the text of a compilation unit is submitted to a compiler.  It
-- does not specify that the text is stored in a "file", nor does it specify that
-- the text of a compilation unit has any particular lifetime.
--
-------------------------------------------------------------------------------
```

```
    type Span is                                       -- Default is Nil_Span
        record
            First_Line   : Line_Number_Positive        := 1; -- 1..0 - empty
            First_Column : Character_Position_Positive := 1; -- 1..0 - empty
            Last_Line    : Line_Number                 := 0;
            Last_Column  : Character_Position          := 0;
        end record;

    Nil_Span : constant Span := (First_Line   => 1,
                                 First_Column => 1,
                                 Last_Line    => 0,
                                 Last_Column  => 0);
```

-- 20.8 function First_Line_Number

```
    function First_Line_Number (Element : in Asis.Element)
                               return Line_Number;

---------------------------------------------------------------------------------------
-- Element - Specifies the element to query
--
-- Returns the first line number on which the text of the element resides.
--
-- Returns 0 if not Is_Text_Available(Element).
--
--|AN Application Note:
--|AN
--|AN The line number recorded for a particular element may or may not match the
--|AN "true" line number of the program text for that element if the Ada environment
--|AN and the local text editors do not agree on the definition of "line".  For
--|AN example, the Reference Manual states that any occurrence of an Ascii.Cr character
--|AN is to be treated as one or more end-of-line occurrences.  On most Unix systems,
--|AN the editors do not treat a carriage return as being an end-of-line character.
--|AN
--|AN Ada treats all of the following as end-of-line characters: Ascii.Cr,
--|AN Ascii.Lf, Ascii.Ff, Ascii.Vt.  It is up to the compilation system to
--|AN determine whether sequences of these characters causes one, or more,
--|AN end-of-line occurrences.  Be warned, if the Ada environment and the
--|AN system editor (or any other line-counting program) do not use the same
--|AN end-of-line conventions, then the line numbers reported by ASIS may not
--|AN match those reported by those other programs.
--
```

-- 20.9 function Last_Line_Number

```
    function Last_Line_Number (Element : in Asis.Element)
                               return Line_Number;

---------------------------------------------------------------------------------------
-- Element - Specifies the element to query
--
-- Returns the last line number on which the text of the element resides.
--
-- Returns 0 if not Is_Text_Available(Element).
--
```

-- 20.10 function Element_Span

```
    function Element_Span (Element : in Asis.Element)
                           return Span;

---------------------------------------------------------------------------------------
-- Element - Specifies the element to query
--
-- Returns the span of the given element.
--
-- Returns a Nil_Span if the text of a Compilation_Unit (Compilation) cannot be
-- located for any reason.
```

```
--
--|AN Application Note:
--|AN
--|AN For this query, Element is only a means to access the
--|AN Compilation_Unit (Compilation), the availability of the text of this Element
--|AN itself is irrelevant to the result of the query.
--
-------------------------------------------------------------------------------
```

-- 20.11 function Compilation_Unit_Span

```
-------------------------------------------------------------------------------

    function Compilation_Unit_Span (Element : in Asis.Element)
                                   return Span;

-------------------------------------------------------------------------------
-- Element - Specifies the element to query
--
-- Returns the span of the text comprising the enclosing compilation unit of
-- the given element.
--
-- Returns a Nil_Span if the text of a Compilation_Unit (Compilation) cannot be
-- located for any reason.
--
--|AN Application Note:
--|AN
--|AN For this query, Element is only a means to access the
--|AN Compilation_Unit (Compilation), the availability of the text of this Element
--|AN itself is irrelevant to the result of the query.
--
-------------------------------------------------------------------------------
```

-- 20.12 function Compilation_Span

```
-------------------------------------------------------------------------------

    function Compilation_Span (Element : in Asis.Element)
                              return Span;

-------------------------------------------------------------------------------
-- Element - Specifies the element to query
--
-- Returns the span of the text comprising the compilation to which the
-- element belongs.  The text span may include one or more compilation units.
--
-- Returns a Nil_Span if not Is_Text_Available(Element).
--
-------------------------------------------------------------------------------
```

-- 20.13 function Is_Nil

```
-------------------------------------------------------------------------------

    function Is_Nil (Right : in Line)
                    return Boolean;

-------------------------------------------------------------------------------
-- Right   - Specifies the line to check
--
-- Returns True if the argument is the Nil_Line.
--
-- A Line from a Line_List obtained from any of the Lines functions
-- will not be Is_Nil even if it has a length of zero.
--
-------------------------------------------------------------------------------
```

-- 20.14 function Is_Nil

```
-------------------------------------------------------------------------------

    function Is_Nil (Right : in Line_List)
                    return Boolean;

-------------------------------------------------------------------------------
-- Right   - Specifies the line list to check
--
-- Returns True if the argument has a 'Length of zero.
--
```

-- 20.15 function Is_Nil

```
    function Is_Nil (Right : in Span)
                     return Boolean;

-------------------------------------------------------------------------------------
-- Right   - Specifies the Span to check
--
-- Returns True if the argument has a Nil_Span.
--
-------------------------------------------------------------------------------------
```

-- 20.16 function Is_Equal

```
-------------------------------------------------------------------------------------

    function Is_Equal (Left  : in Line;
                       Right : in Line) return Boolean;

-------------------------------------------------------------------------------------
-- Left    - Specifies the first of the two lines
-- Right   - Specifies the second of the two lines
--
-- Returns True if the two lines encompass the same text (have the same Span
-- and are from the same compilation).
--
-------------------------------------------------------------------------------------
```

-- 20.17 function Is_Identical

```
-------------------------------------------------------------------------------------

    function Is_Identical (Left  : in Line;
                           Right : in Line) return Boolean;

-------------------------------------------------------------------------------------
-- Left    - Specifies the first of the two lines
-- Right   - Specifies the second of the two lines
--
-- Returns True if the two lines encompass the same text (have the same Span
-- and are from the same compilation) and are from the same Context.
--
-------------------------------------------------------------------------------------
```

-- 20.18 function Length

```
-------------------------------------------------------------------------------------

    function Length (The_Line : in Line) return Character_Position;

-------------------------------------------------------------------------------------
-- The_Line    - Specifies the line to query
--
-- Returns the length of the line.
--
-- Raises ASIS_Inappropriate_Line if Is_Nil (The_Line).
--
-------------------------------------------------------------------------------------
```

-- 20.19 function Lines

```
-------------------------------------------------------------------------------------

    function Lines (Element : in Asis.Element) return Line_List;

-------------------------------------------------------------------------------------
-- Element - Specifies the element to query
--
-- Returns a list of lines covering the span of the given program element.
--
-- Returns a Nil_Span if the text of a Compilation containing a given
-- Element cannot be located for any reason.
--
-- Line lists can be indexed to obtain individual lines.  The bounds of each
-- list correspond to the lines with those same numbers in the compilation
-- text.
--
```

```
-- The first Line of the result contains text from the compilation starting at
-- the First_Line/First_Column of Element’s Span.  The last Line of the result
-- contains text from the compilation ending at the Last_Line/Last_Column of
-- the Element’s Span.  Text before or after those limits is not reflected
-- in the returned list.
--
--|AN Application Note:
--|AN
--|AN For this query, Element is only a means to access the
--|AN Compilation_Unit (Compilation), the availability of the text of this Element
--|AN itself is irrelevant to the result of the query.
--
-------------------------------------------------------------------------------------
```

-- 20.20 function Lines

```
-------------------------------------------------------------------------------------

    function Lines (Element  : in Asis.Element;
                    The_Span : in Span) return Line_List;

-------------------------------------------------------------------------------------
-- Element  - Specifies the element to query
-- The_Span - Specifies the textual span to return
--
-- Returns a list of lines covering the given span from the compilation
-- containing the given program element.
--
-- Returns a Nil_Span if the text of a Compilation containing a given
-- Element cannot be located for any reason.
--
-- This operation can be used to access lines from text outside the span of an
-- element, but still within the compilation.  For example, lines containing
-- preceding comments or lines between two elements.
--
-- Line lists can be indexed to obtain individual lines.  The bounds of each
-- list correspond to the lines with those same numbers in the compilation
-- text.
--
-- The first Line of the result contains text from the compilation starting at
-- line Span.First_Line and column Span.First_Column.  The last Line of the
-- result contains text from the compilation ending at line Span.Last_Line and
-- column Span.Last_Column.  Text before or after those limits is not
-- reflected in the returned list.
--
-- Raises ASIS_Inappropriate_Line_Number if Is_Nil (The_Span). If The_Span defines
-- a line whose number is outside the range of text lines that can be accessed
-- through the Element, the implementation is encouraged, but not required to
-- raise ASIS_Inappropriate_Line_Number.
--
--|AN Application Note:
--|AN
--|AN For this query, Element is only a means to access the
--|AN Compilation_Unit (Compilation), the availability of the text of this Element
--|AN itself is irrelevant to the result of the query.
--
```

```
-------------------------------------------------------------------------------
```
-- 20.21 function Lines
```
-------------------------------------------------------------------------------

    function Lines (Element    : in Asis.Element;
                    First_Line : in Line_Number_Positive;
                    Last_Line  : in Line_Number) return Line_List;

-------------------------------------------------------------------------------
-- Element     - Specifies the element to query
-- First_Line  - Specifies the first line to return
-- Last_Line   - Specifies the last line to return
--
-- Returns a list of Lines covering the full text for each of the indicated
-- lines from the compilation containing the given element.  This operation
-- can be used to access lines from text outside the span of an element, but
-- still within the compilation.
--
-- Returns a Nil_Span if the text of a Compilation containing a given
-- Element cannot be located for any reason.
--
-- Line lists can be indexed to obtain individual lines.  The bounds of each
-- list correspond to the lines with those same numbers in the compilation text.
--
-- Raises ASIS_Inappropriate_Line_Number if the span is nil. If the span defines
-- a line whose number is outside the range of text lines that can be accessed
-- through the Element, the implementation is encouraged, but not required to
-- raise ASIS_Inappropriate_Line_Number.
--
--|AN Application Note:
--|AN
--|AN For this query, Element is only a means to access the
--|AN Compilation_Unit (Compilation), the availability of the text of this Element
--|AN itself is irrelevant to the result of the query.
--
-------------------------------------------------------------------------------
```
-- 20.22 function Delimiter_Image
```
-------------------------------------------------------------------------------

    function Delimiter_Image return Wide_String;

-------------------------------------------------------------------------------
-- Returns the string used as the delimiter separating individual lines of
-- text within the program text image of an element. It is also used as the
-- delimiter separating individual lines of strings returned by Debug_Image.
--
-------------------------------------------------------------------------------
```
-- 20.23 function Element_Image
```
-------------------------------------------------------------------------------

    function Element_Image (Element : in Asis.Element) return Program_Text;

-------------------------------------------------------------------------------
-- Element - Specifies the element to query
--
-- Returns a program text image of the element.  The image of an element can
-- span more than one line, in which case the program text returned by the
-- function Delimiter_Image separates the individual lines.  The bounds on
-- the returned program text value are 1..N, N is as large as necessary.
--
-- Returns a null string if not Is_Text_Available(Element).
--
-- If an Element's Span begins at column position P, the returned program text will
-- be padded at the beginning with P-1 white space characters (Ascii.' ' or Ascii.Ht).
-- The first character of the Element's image will thus begin at character P of the
-- returned program text.  Due to the possible presence of Ascii.Ht characters, the
-- "column" position of characters within the image might not be the same as their
-- print-column positions when the image is displayed on a screen or printed.
--
-- NOTE: The image of a large element can exceed the range of Program_Text. In this
-- case, the exception ASIS_Failed is raised with a Status of Capacity_Error.
-- Use the Lines function to operate on the image of large elements.
```

```
--
-------------------------------------------------------------------------------
```

-- 20.24 function Line_Image

```
-------------------------------------------------------------------------------

    function Line_Image (The_Line : in Line) return Program_Text;

-------------------------------------------------------------------------------
-- The_Line    - Specifies the line to query
--
-- Returns a program text image of the line.  The image of a single lexical
-- element can be sliced from the returned value using the first and last
-- column character positions from the Span of the Element.  The bounds on the
-- returned program text are 1 .. Length(Line).
--
-- If the Line is the first line from the Lines result for an Element, it can
-- represent only a portion of a line from the original compilation.  If the
-- span began at character position P, the first Line of it's Lines
-- result is padded at the beginning with P-1 white space characters
-- (Ascii.' ' or Ascii.Ht).  The first character of the image will
-- thus begin at character P of the program text for the first Line.  Due to the
-- possible presence of Ascii.Ht characters, the "column" position of
-- characters within the image may not be the same as their print-column
-- positions when the image is displayed or printed.
--
-- Similarly, if the Line is the last line from the Lines result for an
-- Element, it may represent only a portion of a line from the original
-- compilation.  The program text image of such a Line is shorter than the
-- line from compilation and will contain only the initial portion of
-- that line.
--
-- Raises ASIS_Inappropriate_Line if Is_Nil (The_Line).
--
-------------------------------------------------------------------------------
```

-- 20.25 function Non_Comment_Image

```
-------------------------------------------------------------------------------

    function Non_Comment_Image (The_Line : in Line) return Program_Text;

-------------------------------------------------------------------------------
-- The_Line    - Specifies the line to query
--
-- Returns a program text image of a Line up to, but excluding, any comment
-- appearing in that Line.
--
-- The value returned is the same as that returned by the Image function,
-- except that any hyphens ("--") that start a comment, and any characters
-- that follow those hyphens, are dropped.
--
-- The bounds on the returned program text are 1..N, where N is one less than the
-- column of any hyphens ("--") that start a comment on the line.
--
-- Raises ASIS_Inappropriate_Line if Is_Nil (The_Line).
--
-------------------------------------------------------------------------------
```

-- 20.26 function Comment_Image

```
-------------------------------------------------------------------------------

    function Comment_Image (The_Line : in Line) return Program_Text;

-------------------------------------------------------------------------------
-- The_Line    - Specifies the line to query
--
-- Returns a program text image of any comment on that line, excluding any
-- lexical elements preceding the comment.
--
-- The value returned is the same as that returned by the Image function,
-- except that any program text prior to the two adjacent hyphens ("--") which start
-- a comment is replaced by an equal number of spaces.  If the hyphens began in
-- column P of the Line, they will also begin in character position P of the
-- returned program text.
--
```

```
-- A null string is returned if the line has no comment.
--
-- The bounds of the program text are 1..N, where N is as large as necessary.
--
-- Raises ASIS_Inappropriate_Line if Is_Nil (The_Line).
--
---------------------------------------------------------------------------------------
```

-- 20.27 function Is_Text_Available

```
---------------------------------------------------------------------------------------

    function Is_Text_Available (Element : in Asis.Element) return Boolean;

---------------------------------------------------------------------------------------
-- Element - Specifies the element to query
--
-- Returns True if the implementation can return a valid text image for the
-- given element.
--
-- Returns False for any Element that Is_Nil, Is_Part_Of_Implicit, or
-- Is_Part_Of_Instance.
--
-- Returns False if the text of the element cannot be located for any reason
-- such as renaming, deletion, or moving of text.
--
--|IR Implementation Requirements:
--|IR
--|IR An implementation shall make text available for all explicit elements.
--
---------------------------------------------------------------------------------------
```

-- 20.28 function Debug_Image

```
---------------------------------------------------------------------------------------

    function Debug_Image (The_Line : in Line) return Wide_String;

---------------------------------------------------------------------------------------
-- The_Line    - Specifies the line to convert
--
-- Returns a string value containing implementation-defined debug
-- information associated with the line.
--
-- The return value uses Asis.Text.Delimiter_Image to separate the lines
-- of multi-line results.  The return value does not end with
-- Asis.Text.Delimiter_Image.
--
-- These values are intended for two purposes.  They are suitable for
-- inclusion in problem reports sent to the ASIS implementor.  They can
-- be presumed to contain information useful when debugging the
-- implementation itself. They are also suitable for use by the ASIS
-- application when printing simple application debugging messages during
-- application development.  They are intended to be, to some worthwhile
-- degree, intelligible to the user.
--
---------------------------------------------------------------------------------------

private

    type Line is (Implementation_Defined);
    Nil_Line  : constant Line := Implementation_Defined;
    Nil_Line_List : constant Line_List (1 .. 0) := (others => Nil_Line);

---------------------------------------------------------------------------------------

end Asis.Text;
```

```
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
```

-- 21 package Asis.Ids

```
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
package Asis.Ids is
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-- Asis.Ids provides support for permanent unique Element "Identifiers" (Ids).
-- An Id is an efficient way for a tool to reference an ASIS element.  The Id
-- is permanent from session to session provided that the Ada compilation
-- environment is unchanged.
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-- This package encapsulates a set of operations and queries that implement
-- the ASIS Id abstraction.  An Id is a way of identifying a particular
-- Element, from a particular Compilation_Unit, from a particular Context.
-- Ids can be written to files.  Ids can be read from files and converted into
-- an Element value with the use of a suitable open Context.
--
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
```

-- 21.1 type Id

```
-----------------------------------------------------------------------------------
-- The Ada element Id abstraction (a private type).
--
-- The Id type is a distinct abstract type representing permanent "names"
-- that correspond to specific Element values.
--
-- These names can be written to files, retrieved at a later time, and
-- converted to Element values.
-----------------------------------------------------------------------------------
-- ASIS Ids are a means of identifying particular Element values obtained from
-- a particular physical compilation unit.  Ids are "relative names".  Each Id
-- value is valid (is usable, makes sense, can be interpreted) only in the
-- context of an appropriate open ASIS Context.
--
-- Id shall be an undiscriminated private type, or, shall be derived from an
-- undiscriminated private type.  It shall be declared as a new type or as a
-- subtype of an existing type.
-----------------------------------------------------------------------------------

    type Id is private;
    Nil_Id : constant Id;

    function "=" (Left  : in Id;
                  Right : in Id)
                  Return Boolean is abstract;


-----------------------------------------------------------------------------------
```

-- 21.2 function Hash

```
-----------------------------------------------------------------------------------

    function Hash (The_Id : in Id)              return Asis.ASIS_Integer;

-----------------------------------------------------------------------------------
```

-- 21.3 function "<"

```
-----------------------------------------------------------------------------------

    function "<" (Left  : in Id;
                  Right : in Id) return Boolean;

-----------------------------------------------------------------------------------
```

-- 21.4 function ">"

```
-----------------------------------------------------------------------------------

    function ">" (Left  : in Id;
                  Right : in Id) return Boolean;
```

```
-----------------------------------------------------------------------------------
```
-- 21.5 function Is_Nil
```
-----------------------------------------------------------------------------------

    function Is_Nil (Right : in Id) return Boolean;

-----------------------------------------------------------------------------------
-- Right   - Specifies the Id to check
--
-- Returns True if the Id is the Nil_Id.
--
-----------------------------------------------------------------------------------
```
-- 21.6 function Is_Equal
```
-----------------------------------------------------------------------------------

   function Is_Equal (Left  : in Id;
                      Right : in Id) return Boolean;

-----------------------------------------------------------------------------------
-- Left    - Specifies the left Id to compare
-- Right   - Specifies the right Id to compare
--
-- Returns True if Left and Right represent the same physical Id, from the
-- same physical compilation unit.  The two Ids convert
-- to Is_Identical Elements when converted with the same open ASIS Context.
--
-----------------------------------------------------------------------------------
```
-- 21.7 function Create_Id
```
-----------------------------------------------------------------------------------

    function Create_Id (Element : in Asis.Element) return Id;

-----------------------------------------------------------------------------------
-- Element - Specifies any Element value whose Id is desired
--
-- Returns a unique Id value corresponding to this Element, from the
-- corresponding Enclosing_Compilation_Unit and the corresponding
-- Enclosing_Context.  The Id value will not be equal ("=") to the Id value
-- for any other Element value unless the two Elements are Is_Identical.
--
-- Nil_Id is returned for a Nil_Element.
--
-- All Element_Kinds are appropriate.
--
-----------------------------------------------------------------------------------
```
-- 21.8 function Create_Element
```
-----------------------------------------------------------------------------------

    function Create_Element (The_Id      : in Id;
                             The_Context : in Asis.Context)
                           return Asis.Element;

-----------------------------------------------------------------------------------
-- The_Id      - Specifies the Id to be converted to an Element
-- The_Context - Specifies the Context containing the Element with this Id
--
-- Returns the Element value corresponding to The_Id.  The_Id shall
-- correspond to an Element available from a Compilation_Unit contained by
-- (referencible through) The_Context.
--
-- Raises ASIS_Inappropriate_Element if the Element value is not available
-- though The_Context.  The Status is Value_Error and the Diagnosis
-- string will attempt to indicate the reason for the failure.  (e.g., "Unit is
-- inconsistent", "No such unit", "Element is inconsistent (Unit inconsistent)",
-- etc.)
--
```

```
-------------------------------------------------------------------------------------
```

-- 21.9 function Debug_Image

```
-------------------------------------------------------------------------------------

    function Debug_Image (The_Id : in Id) return Wide_String;

-------------------------------------------------------------------------------------
-- The_Id  - Specifies an Id to convert
--
-- Returns a string value containing implementation-defined debug
-- information associated with the Id.
--
-- The return value uses Asis.Text.Delimiter_Image to separate the lines
-- of multi-line results.  The return value does not end with
-- Asis.Text.Delimiter_Image.
--
-- These values are intended for two purposes.  They are suitable for
-- inclusion in problem reports sent to the ASIS implementor.  They can
-- be presumed to contain information useful when debugging the
-- implementation itself. They are also suitable for use by the ASIS
-- application when printing simple application debugging messages during
-- application development.  They are intended to be, to some worthwhile
-- degree, intelligible to the user.
--
-------------------------------------------------------------------------------------

private

    type Id is (Implementation_Defined);
    Nil_Id : constant Id := Implementation_Defined;

-------------------------------------------------------------------------------------

end Asis.Ids;
```

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

-- 22 package Asis.Data_Decomposition (optional)

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
package Asis.Data_Decomposition is
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- Asis.Data_Decomposition
--
-- This package is optional.
--
-- Operations to decompose data values using the ASIS type information and a
-- Portable_Data stream, representing a data value of that type.
--
-- An application can write data, using the Asis.Data_Decomposition.Portable_Transfer
-- package to an external medium for later retrieval by another application.  The
-- second application reads that data and then uses this package to convert that data
-- into useful information.  Simple discrete scalar types can be converted
-- directly into useful information.  Composite types, such as records and arrays,
-- shall first be broken into their various discriminants and components.
--
-- A data stream representing a record value can be decomposed into a group
-- of discriminant and component data streams by extracting those streams from
-- the record's data stream.  This extraction is performed by applying any of
-- the Record_Components which describe the discriminants and components of
-- the record type.  Each discriminant and each component of a record type is
-- described by a Record_Component value.  Each value encapsulates the
-- information needed, by the implementation, to efficiently extract the
-- associated field from a data stream representing a record value of the
-- correct type.
--
-- A data stream representing an array value can be decomposed into a group of
-- component data streams by extracting those streams from the array's data
-- stream.  This extraction is performed by applying the single
-- Array_Component which describes the components of the array type.  One
-- Array_Component value is used to describe all array components.  The value
-- encapsulates the information needed, by the implementation, to efficiently
-- extract any of the array components.
--
-- Assumptions and Limitations of this Interface:
--
-- a) The data stream is appropriate for the ASIS host machine.  For example,
--    the implementation of this interface will not need to worry about
--    byte flipping or reordering of bits caused by movement of data between
--    machine architectures.
--
-- b) Records, arrays, and their components may be packed.
--
-- c) Records, array components, enumerations, and scalar types may have
--    representation and length clauses applied to them.  This includes scalar
--    types used as record discriminants and array indices.
--
-- d) This specification supports two of the three type models discussed
--    below.  Models A and B are supported.  Model C is not supported.
--
--    1) Simple "static" types contain no variants, have a single fixed 'Size,
--       and all components and attributes are themselves static and/or fully
--       constrained.  The size and position for any component of the type can be
--       determined without regard to constraints.  For example:
--
--           type Static_Record is
--               record
--                   F1, F2 : Natural;
--                   C1     : Wide_Character;
--                   A1     : Wide_String (1..5);
--               end record;
--
--           type Static_Discriminated (X : Boolean) is
--               record
--                   F1, F2 : Natural;
--                   C1     : Wide_Character;
--               end record;
```

```
--
--          type Static_Array   is array (Integer range 1 .. 100) of Boolean;
--          type Static_Enum    is (One, Two, Three);
--          type Static_Integer is range 1 .. 512;
--          type Static_Float   is digits 15 range -100.0 .. 100.0;
--          type Static_Fixed   is delta 0.1 range -100.0 .. 100.0;
--
--    2) Simple "dynamic" types contain one or more components or attributes
--       whose size, position, or value depends on the value of one or more
--       constraints computed at execution time.  This means that the size,
--       position, or number of components within the data type cannot be
--       determined without reference to constraint values.
--
--       Records containing components, whose size depends on discriminants
--       of the record, can be handled because the discriminants for a record
--       value are fully specified by the data stream form of the record value.
--       For example:
--
--          type Dynamic_Length (Length : Natural) is
--              record
--                  S1 : Wide_String (1 .. Length);
--              end record;
--
--          type Dynamic_Variant (When : Boolean) is
--              record
--                  case When is
--                      when True =>
--                          C1 : Wide_Character;
--                      when False =>
--                          null;
--                  end case;
--              end record;
--
--       Arrays with an unconstrained subtype, whose 'Length, 'First, and 'Last
--       depend on dynamic index constraints, can be handled because these
--       attributes can be queried and stored when the data stream is written.
--       For example:
--
--           I : Integer := Some_Function;
--           type Dynamic_Array is
--               array (Integer range I .. I + 10) of Boolean;
--
--           type Heap_Array   is array (Integer range <>) of Boolean;
--           type Access_Array is access Heap_Array;
--           X : Access_Array := new Heap_Array (1 .. 100);
--
--    3) Complex, externally "discriminated" records, contain one or more
--       components whose size or position depends on the value of one or more
--       non-static external values (values not stored within instances of the
--       type) at execution time.  The size for a value of the type cannot be
--       determined without reference to these external values, whose runtime
--       values are not known to the ASIS Context and cannot be automatically
--       recorded by the Asis.Data_Decomposition.Portable_Transfer generics.
--       For example:
--
--          N : Natural := Function_Call();
--          ....
--          declare
--              type Complex is
--                  record
--                      S1 : Wide_String (1 .. N);
--                  end record;
--          begin
--              ....
--          end;
--
--
-- General Usage Rules:
--
-- All operations in this package will attempt to detect the use of invalid
-- data streams.  A data stream is "invalid" if an operation determines that
-- the stream could not possibly represent a value of the expected variety.
-- Possible errors are: stream is of incorrect length, stream contains bit
-- patterns which are illegal, etc.  The exception ASIS_Inappropriate_Element
```

```
-- is raised in these cases.  The Status value is Data_Error.  The
-- Diagnosis string will indicate the kind of error detected.
--
-- All implementations will handle arrays with a minimum of 16 dimensions,
-- or the number of dimensions allowed by their compiler, whichever is
-- smaller.
--
---------------------------------------------------------------------------------------
```

-- 22.1 type Record_Component

```
---------------------------------------------------------------------------------------
-- Type Record_Component describes one discriminant or component of a record type.
--
-- Implementation is highly implementation dependent.  The "=" operator is not
-- meaningful between Record_Component values unless one of them is the
-- Nil_Record_Component value.
--
-- A record type describes composite values which contain zero or more
-- discriminant and component fields.  A_Record_Type_Definition can be queried
-- to obtain a list of Record_Components.  Each Record_Component contains the
-- information necessary to extract one discriminant or component field of the
-- record.
--
-- Record_Components are intended for use with data stream extraction
-- operations.  An extraction operation is performed using a Record_Component,
-- in conjunction with a data stream representing a value of the record type.
-- The record data stream contains data for all fields of the record.  The
-- result is an extracted data stream representing just the value of the one
-- field.  Record_Components are implemented so as to allow for efficient
-- extraction of field values.
--
-- An extracted field data stream is suitable for all uses.  If the field is a
-- scalar type, it can be converted directly into useful information.  If the
-- field is, in turn, another composite value, it can be further decomposed
-- into its own component values.
--
-- There are two ways to obtain the Record_Components or the Array_Component
-- needed to further decompose an embedded composite field.  First, if the
-- type of the field is known, the type definition can be directly queried to
-- obtain the Record_Components or the Array_Component that describe its
-- internal structure.  Second, the Record_Component used to extract the field
-- can be queried to obtain the same Record_Components or the same
-- Array_Component.  Both methods return identical information.
--
-- This kind of nested decomposition can be carried to any required level.
--
-- Record_Components become invalid when the Context, from which they
-- originate, is closed.  All Record_Components are obtained by referencing a)
-- an Element, which has an associated Context, b) another Record_Component,
-- or c) an Array_Component.  Ultimately, all component values originate from
-- a A_Type_Definition Element; that Element determines their Context of
-- origin.
---------------------------------------------------------------------------------------

    type Record_Component is private;
    Nil_Record_Component : constant Record_Component;

    function "=" (Left  : in Record_Component;
                  Right : in Record_Component)
                  Return Boolean is abstract;


---------------------------------------------------------------------------------------
```

-- 22.2 type Record_Component_List

```
---------------------------------------------------------------------------------------

    type Record_Component_List is
       array (Asis.List_Index range <>) of Record_Component;
```

```
-------------------------------------------------------------------------------
```

-- 22.3 type Array_Component

```
-------------------------------------------------------------------------------
-- Type Array_Component describes the components of an array valued field for a record
-- type.
--
-- Implementation is highly implementor dependent.  The "=" operator is not
-- meaningful between Array_Component values unless one of them is the
-- Nil_Array_Component value.
--
-- An array type describes composite values which contain zero or more indexed
-- components. Both An_Unconstrained_Array_Definition or
-- A_Constrained_Array_Definition can be queried to obtain a single
-- Array_Component. The Array_Component contains the information necessary to
-- extract any arbitrary component of the array.
--
-- Array_Components are intended for use with data stream extraction
-- operations.  An extraction operation is performed using an Array_Component,
-- in conjunction with a data stream representing a value of the array type.
-- The array data stream contains data for all components of the array.  The
-- result is an extracted data stream representing just the value of the one
-- component.  Array_Components are implemented so as to allow for efficient
-- extraction of array component values.
--
-- An extracted component data stream is suitable for all uses.  If the
-- component is a scalar type, it can be converted directly into useful
-- information.  If the component is, in turn, another composite value, it can
-- be further decomposed into its own component values.
--
-- There are two ways to obtain the Record_Components or the Array_Component
-- needed to further decompose an embedded composite component.  First, if the
-- type of the component is known, the type definition can be directly queried
-- to obtain the Record_Components or the Array_Component that describe its
-- internal structure.  Second, the Array_Component used to extract the
-- component can be queried to obtain the same Record_Components or the same
-- Array_Component.  Both methods return identical information.
--
-- This kind of nested decomposition can be carried to any required level.
--
-- Array_Components become invalid when the Context, from which they
-- originate, is closed.  All Record_Components are obtained by referencing a)
-- an Element, which has an associated Context, b) a Record_Component, or c)
-- another Array_Component.  Ultimately, all component values originate from a
-- A_Type_Definition Element; that Element determines their Context of origin.
-------------------------------------------------------------------------------

    type Array_Component is private;
    Nil_Array_Component : constant Array_Component;

    function "=" (Left  : in Array_Component;
                  Right : in Array_Component)
                  Return Boolean is abstract;

-------------------------------------------------------------------------------
```

-- 22.4 type Array_Component_List

```
-------------------------------------------------------------------------------

    type Array_Component_List is
       array (Asis.List_Index range <>) of Array_Component;

-------------------------------------------------------------------------------
```

-- 22.5 type Dimension_Indexes

```
-------------------------------------------------------------------------------
-- Dimension_Indexes - an array of index values used to access an array stream
-------------------------------------------------------------------------------

    type Dimension_Indexes is
       array (Asis.ASIS_Positive range <>) of Asis.ASIS_Positive;
```

```
-------------------------------------------------------------------------------
```
-- 22.6 type Array_Component_Iterator
```
-------------------------------------------------------------------------------
-- Type Array_Component_Iterator is used to iterate over successive components of an
-- array.  This can be more efficient than using individual index values when
-- extracting array components from a data stream because it substitutes two
-- subroutine calls (Next and Done) for the multiplications and divisions
-- implicit in indexing an N dimensional array with a single index value.
--
-- Iterators can be copied.  The copies operate independently (have separate
-- state).
--
-- An example:
--
--     declare
--         Component        : Array_Component := ...;
--         Iter             : Array_Component_Iterator;
--         Array_Stream     : Portable_Data (...) := ...;
--         Component_Stream : Portable_Data (...);
--     begin
--         Iter := Array_Iterator (Component);
--         while not Done (Iter) loop
--             Component_Stream := Component_Data_Stream (Iter, Array_Stream);
--             Next (Iter);
--         end loop;
--     end;
-------------------------------------------------------------------------------

    type Array_Component_Iterator is private;

    Nil_Array_Component_Iterator : constant Array_Component_Iterator;

-------------------------------------------------------------------------------
```
-- 22.7 type Portable_Data
```
-------------------------------------------------------------------------------
--
-- Portable_Data represents an ordered "stream" of data values.
--
-- The portable representation for application data is an array of data
-- values.  This portable data representation is guaranteed to be valid when
-- written, and later read, on the same machine architecture, using the same
-- implementor's compiler and runtime system.  Portability of the data
-- values, across implementations and architectures, is not guaranteed.
-- Some implementors may be able to provide data values which are portable
-- across a larger subset of their supported machine architectures.
--
-- Some of the problems encountered when changing architectures are: bit
-- order, byte order, floating point representation, and alignment
-- constraints.  Some of the problems encountered when changing runtime
-- systems or implementations are: type representation, optimization,
-- record padding, and other I/O subsystem implementation variations.
--
-- The nature of these data values is deliberately unspecified.  An
-- implementor will choose a data value type that is suitable for the
-- expected uses of these arrays and data values.  Arrays and data
-- values have these uses:
--
-- a) Array values are used in conjunction with the
--    Asis.Data_Decomposition interface.  The data value type should be
--    readily decomposable, by that package, so that array and record
--    components can be efficiently extracted from a data stream represented
--    by this array type.  The efficiency of that interface is a priority.
--
-- b) The data value type is read and written by applications.  It
--    should have a size that makes efficient I/O possible.  Applications can
--    be expected to perform I/O in any or all of these ways:
--
--    1) Ada.Sequential_Io or Ada.Direct_Io could be used to read or write
--       these values.
--
--    2) Individual values may be placed inside other types and those types
--       may be read or written.
--
```

```
--    3) The 'Address of a data value, plus the 'Size of the data value
--       type, may be used to perform low level system I/O.  Note: This
--       requires the 'Size of the type and the 'Size of a variable of that
--       type to be the same for some implementations.
--
--    4) Individual values may be passed through Unchecked_Conversion in
--       order to obtain a different value type, of the same 'Size, suitable
--       for use with some user I/O facility.  This usage is non-portable
--       across implementations.
--
-- c) Array values are read and written by applications.  The data value
--    type should have a size that makes efficient I/O possible.
--    Applications can be expected to perform I/O in any or all of these ways:
--
--    1) Ada.Sequential_Io or Ada.Direct_Io could be used to read or write a
--       constrained array subtype.
--
--    2) Array values may be placed inside other types and those types may
--       be read and written.
--
--    3) The 'Address of the first array value, plus the 'Length of the
--       array times the 'Size of the values, may be used to perform low
--       level system I/O.  Note: This implies that the array type is
--       unpacked, or, that the packed array type has no "padding" (e.g.,
--       groups of five 6-bit values packed into 32-bit words with 2 bits
--       of padding every 5 elements).
--
--    4) Array values may be passed through Unchecked_Conversion in order to
--       obtain an array value, with a different value type, suitable for
--       use with some user I/O facility.  This usage is non-portable across
--       implementations.
--
-- The data value type should be chosen so that the 'Address of the first
-- array data value is also the 'Address of the first storage unit containing
-- array data.  This is especially necessary for target architectures where
-- the "bit" instructions address bits in the opposite direction as that used
-- by normal machine memory (or array component) indexing.  A recommended
-- 'Size is System.Storage_Unit (or a multiple of that size).
--
-- Implementations that do not support Unchecked_Conversion of array values,
-- or which do not guarantee that Unchecked_Conversion of array values will
-- always "do the right thing" (convert only the data, and not the dope vector
-- information), should provide warnings in their ASIS documentation that
-- detail possible consequences and work-arounds.
--
-- The index range for the Portable_Data type shall be a numeric type whose
-- range is large enough to encompass the Portable_Data representation for all
-- possible runtime data values.
--
-- All conversion interfaces always return Portable_Data array values with a
-- 'First of one (1).
--
-- The Portable_Value type may be implemented in any way
-- whatsoever.  It need not be a numeric type.
---------------------------------------------------------------------------------------

    type Portable_Value is (Implementation_Defined);

    subtype Portable_Positive is Asis.ASIS_Positive
       range 1 .. Implementation_Defined_Integer_Constant;

    type Portable_Data is array (Portable_Positive range <>) of Portable_Value;

    Nil_Portable_Data : Portable_Data (1 .. 0);
```

```
-------------------------------------------------------------------------------
```

-- 22.8 type Type_Model_Kinds

```
-------------------------------------------------------------------------------
-- Type_Model_Kinds
--
-- Each Type_Definition fits into one of three type models.
-------------------------------------------------------------------------------

    type Type_Model_Kinds is (A_Simple_Static_Model,
                              A_Simple_Dynamic_Model,
                              A_Complex_Dynamic_Model,
                              Not_A_Type_Model);           -- Nil arguments

-------------------------------------------------------------------------------
```

-- 22.9 function Type_Model_Kind

```
-------------------------------------------------------------------------------

    function Type_Model_Kind (Type_Definition : in Asis.Type_Definition)
                             return Type_Model_Kinds;

    function Type_Model_Kind (Component : in Record_Component)
                             return Type_Model_Kinds;

    function Type_Model_Kind (Component : in Array_Component)
                             return Type_Model_Kinds;

-------------------------------------------------------------------------------
-- Type_Definition - Specifies the type definition to query
-- Component       - Specifies a record field with a record or array type
--
-- Returns the model that best describes the type indicated by the argument.
-- Returns Not_A_Type_Model for any unexpected argument such as a Nil value.
--
-- Expected Element_Kinds:
--      A_Type_Definition
--
-------------------------------------------------------------------------------
```

-- 22.10 function Is_Nil

```
-------------------------------------------------------------------------------

    function Is_Nil (Right : in Record_Component) return Boolean;

    function Is_Nil (Right : in Array_Component)  return Boolean;

-------------------------------------------------------------------------------
-- Right   - Specifies the component to check
--
-- Returns True if Right is a Nil (or uninitialized) component value.
--
-- Returns False for all other values.
--
-- All component values are appropriate.
--
-------------------------------------------------------------------------------
```

-- 22.11 function Is_Equal

```
-------------------------------------------------------------------------------

    function Is_Equal (Left  : in Record_Component;
                       Right : in Record_Component) return Boolean;

    function Is_Equal (Left  : in Array_Component;
                       Right : in Array_Component)  return Boolean;

-------------------------------------------------------------------------------
-- Left    - Specifies the left component to compare
-- Right   - Specifies the right component to compare
--
-- Returns True if Left and Right represent the same physical component of the
-- same record or array type, from the same physical compilation unit.  The
-- two components may or may not be from the same open ASIS Context variable.
--
```

```
-- Implies:
--
--     Is_Equal (Enclosing_Compilation_Unit (Component_Declaration (Left)),
--               Enclosing_Compilation_Unit (Component_Declaration (Right)))
--     = True
--
-- All component values are appropriate.
--
---------------------------------------------------------------------------------------
```

-- 22.12 function Is_Identical

```
---------------------------------------------------------------------------------------

    function Is_Identical (Left  : in Record_Component;
                           Right : in Record_Component) return Boolean;

    function Is_Identical (Left  : in Array_Component;
                           Right : in Array_Component)  return Boolean;

---------------------------------------------------------------------------------------
-- Left    - Specifies the left component to compare
-- Right   - Specifies the right component to compare
--
-- Returns True if Left and Right represent the same physical component of the
-- same record or array type, from the same physical compilation unit and the
-- same open ASIS Context variable.
--
-- Implies:
--
--      Is_Identical (Enclosing_Compilation_Unit (Component_Declaration (Left)),
--                    Enclosing_Compilation_Unit (Component_Declaration (Right)))
--      = True
--
-- All component values are appropriate.
--
---------------------------------------------------------------------------------------
```

-- 22.13 function Is_Array

```
---------------------------------------------------------------------------------------

    function Is_Array (Component : in Record_Component) return Boolean;

    function Is_Array (Component : in Array_Component)  return Boolean;

---------------------------------------------------------------------------------------
-- Component   - Specifies any component
--
-- Returns True if the component has an array subtype (contains an array
-- value).
--
-- Returns False for Nil components and any component that is not an embedded
-- array.
--
---------------------------------------------------------------------------------------
```

-- 22.14 function Is_Record

```
---------------------------------------------------------------------------------------

    function Is_Record (Component : in Record_Component) return Boolean;
    function Is_Record (Component : in Array_Component)  return Boolean;

---------------------------------------------------------------------------------------
-- Component   - Specifies any component
--
-- Returns True if the component has a record subtype.
-- Returns False for Nil components and any component that is not an embedded
-- record.
--
```

```
-------------------------------------------------------------------------------
```
-- 22.15 function Done
```
-------------------------------------------------------------------------------

    function Done (Iterator : in Array_Component_Iterator) return Boolean;

-------------------------------------------------------------------------------
-- Iterator    - Specifies the iterator to query
--
-- Returns True if the iterator has been advanced past the last array
-- component.  Returns True for a Nil_Array_Component_Iterator.
--
-------------------------------------------------------------------------------
```
-- 22.16 procedure Next
```
-------------------------------------------------------------------------------

    procedure Next (Iterator : in out Array_Component_Iterator);

-------------------------------------------------------------------------------
-- Iterator    - Specifies the iterator to advance
--
-- Advances the iterator to the next array component.  Use Done to test the
-- iterator to see if it has passed the last component.  Does nothing if the
-- iterator is already past the last component.
--
-------------------------------------------------------------------------------
```
-- 22.17 procedure Reset
```
-------------------------------------------------------------------------------

    procedure Reset (Iterator : in out Array_Component_Iterator);

-------------------------------------------------------------------------------
-- Iterator    - Specifies the iterator to reset
--
-- Resets the iterator to the first array component.
--
-------------------------------------------------------------------------------
```
-- 22.18 function Array_Index
```
-------------------------------------------------------------------------------

    function Array_Index (Iterator : in Array_Component_Iterator)
                          return Asis.ASIS_Natural;

-------------------------------------------------------------------------------
-- Iterator    - Specifies the iterator to query
--
-- Returns the Index value which, when used in conjunction with the
-- Array_Component value used to create the Iterator, indexes the same array
-- component as that presently addressed by the Iterator.
--
-- Raises ASIS_Inappropriate_Element if given a Nil_Array_Component_Iterator
-- or one where Done(Iterator) = True.  The Status value is Data_Error.
-- The Diagnosis string will indicate the kind of error detected.
--
-------------------------------------------------------------------------------
```
-- 22.19 function Array_Indexes
```
-------------------------------------------------------------------------------

    function Array_Indexes (Iterator : in Array_Component_Iterator)
                            return Dimension_Indexes;

-------------------------------------------------------------------------------
-- Iterator    - Specifies the iterator to query
--
-- Returns the index values which, when used in conjunction with the
-- Array_Component value used to create the Iterator, indexes the same array
-- component as that presently addressed by the Iterator.
--
-- Raises ASIS_Inappropriate_Element if given a Nil_Array_Component_Iterator
-- or one where Done(Iterator) = True.  The Status value is Data_Error.
-- The Diagnosis string will indicate the kind of error detected.
```

```
--
-----------------------------------------------------------------------------------------
```

-- 22.20 function Discriminant_Components

```
-----------------------------------------------------------------------------------------

    function Discriminant_Components (Type_Definition : in Asis.Type_Definition)
                                      return Record_Component_List;

    function Discriminant_Components (Component : in Record_Component)
                                      return Record_Component_List;

    function Discriminant_Components (Component : in Array_Component)
                                      return Record_Component_List;

-----------------------------------------------------------------------------------------
-- Type_Definition - Specifies the record type definition to query
-- Component       - Specifies a component which has a record subtype,
--                   Is_Record(Component) = True
--
-- Returns a list of the discriminant components for records of the indicated
-- record type.
--
-- The result describes the locations of the record type's discriminants,
-- regardless of the static or dynamic nature of the record type.
-- All return components are intended for use with a data stream representing
-- a value of the indicated record type.
--
-- All Is_Record(Component) = True arguments are appropriate.  All return
-- values are valid parameters for all query operations.
--
-- Appropriate Element_Kinds:
--      A_Type_Definition
--
-- Appropriate Type_Kinds:
--      A_Derived_Type_Definition       (derived from a record type)
--      A_Record_Type_Definition
--
-- Appropriate Asis.Data_Decomposition.Type_Model_Kinds:
--      A_Simple_Static_Model
--      A_Simple_Dynamic_Model
--      A_Complex_Dynamic_Model
--
-----------------------------------------------------------------------------------------
```

-- 22.21 function Record_Components

```
-----------------------------------------------------------------------------------------

    function Record_Components (Type_Definition : in Asis.Type_Definition)
                                return Record_Component_List;

    function Record_Components (Component : in Record_Component)
                                return Record_Component_List;

    function Record_Components (Component : in Array_Component)
                                return Record_Component_List;

-----------------------------------------------------------------------------------------
-- Type_Definition - Specifies the record type definition to query
-- Component       - Specifies a component which has a record subtype,
--                   Is_Record(Component) = True
--
-- Returns a list of the discriminants and components for the indicated simple
-- static record type.  (See rule 6.A above.)
--
-- The result describes the locations of the record type's discriminants and
-- components.  All return components are intended for use with a data stream
-- representing a value of the indicated record type.
--
-- All Is_Record (Component) = True values, having simple static types, are
-- appropriate.  All return values are valid parameters for all query operations.
--
-- Note: If an Ada implementation uses implementation-dependent record
-- components (Reference Manual 13.5.1 (15)), then each such component of
-- the record type is included in the result.
```

```
--
-- Appropriate Element_Kinds:
--      A_Type_Definition
--
-- Appropriate Type_Kinds:
--      A_Derived_Type_Definition       (derived from a record type)
--      A_Record_Type_Definition
--
-- Appropriate Asis.Data_Decomposition.Type_Model_Kinds:
--      A_Simple_Static_Model
--
---------------------------------------------------------------------------------------
```

-- 22.22 function Record_Components

```
---------------------------------------------------------------------------------------

    function Record_Components
                (Type_Definition : in Asis.Type_Definition;
                 Data_Stream     : in Portable_Data)
                return Record_Component_List;

    function Record_Components
                (Component   : in Record_Component;
                 Data_Stream : in Portable_Data)
                return Record_Component_List;

    function Record_Components
                (Component   : in Array_Component;
                 Data_Stream : in Portable_Data)
                return Record_Component_List;

---------------------------------------------------------------------------------------
-- Type_Definition - Specifies the record type definition to query
-- Component       - Specifies a component which has a record subtype,
--                   Is_Record(Component) = True
-- Data_Stream     - Specifies a data stream containing, at least, the
--                   complete set of discriminant or index constraints for the type
--
-- Returns a list of the discriminants and components for the indicated record
-- type, using the data stream argument as a guide.  The record type shall be
-- either a simple static, or a simple dynamic, record type.  (See rules 6.A
-- and 6.B above.)
--
-- The result describes the locations of the record type's discriminants and
-- all components of the appropriate variant parts.  The contents of the list
-- are determined by the discriminant values present in the data stream.
--
-- A simple static type will always return the same component list (Is_Equal
-- parts) regardless of the Data_Stream, because the layout of a simple static
-- type does not change with changes in discriminant values.  A simple dynamic
-- type returns different component lists (non-Is_Equal parts) depending on
-- the contents of the Data_Stream, because the contents and layout of a
-- simple dynamic type changes with changes in discriminant values.  All
-- return components are intended for use with a data stream representing a
-- value of the indicate record type.
--
-- The Data_Stream shall represent a fully discriminated value of the indicated
-- record type.  The stream may have been read from a file, it may have been
-- extracted from some enclosing data stream, or it may be an artificial value
-- created by the Construct_Artificial_Data_Stream operation.  Only the
-- discriminant portion of the Data_Stream is checked for validity, and, only
-- some discriminant fields may need to be checked, depending on the
-- complexity of the record type.  The best approach, for any application that
-- is constructing artificial data streams, is to always provide appropriate
-- values for all discriminant fields.  It is not necessary to provide values
-- for non-discriminant fields.
--
-- All Is_Record(Component) = True values are appropriate.  All return values
-- are valid parameters for all query operations.
--
-- Note: If an Ada implementation uses implementation-dependent record
-- components (Reference Manual 13.5.1 (15)), then each such component of the
-- record type is included in the result.
--
```

```
-- Appropriate Element_Kinds:
--      A_Type_Definition
--
-- Appropriate Type_Kinds:
--      A_Derived_Type_Definition       (derived from a record type)
--      A_Record_Type_Definition
--
-- Appropriate Asis.Data_Decomposition.Type_Model_Kinds:
--      A_Simple_Static_Model
--      A_Simple_Dynamic_Model
--
---------------------------------------------------------------------------------------
```

-- 22.23 function Array_Components

```
---------------------------------------------------------------------------------------

    function Array_Components (Type_Definition : in Asis.Type_Definition)
                               return Array_Component;

    function Array_Components (Component : in Record_Component)
                               return Array_Component;

    function Array_Components (Component : in Array_Component)
                               return Array_Component;

---------------------------------------------------------------------------------------
-- Type_Definition - Specifies the array type definition to query
-- Component       - Specifies a component which has an array subtype,
--                    Is_Array(Component) = True
--
-- Returns a single component, describing all components of the indicated
-- array type.  The array type shall be a simple static, or a simple dynamic
-- array type.  (See rules 6.A and 6.B above.)
--
-- The result contains all information necessary to index and extract any
-- component of a data stream representing a value of the indicated array
-- type.
--
-- All Is_Array (Component) = True values are appropriate.  All return values
-- are valid parameters for all query operations.
--
-- Appropriate Element_Kinds:
--      A_Type_Definition
--
-- Appropriate Type_Kinds:
--      A_Derived_Type_Definition       (derived from an array type)
--      An_Unconstrained_Array_Definition
--      A_Constrained_Array_Definition
--
-- Appropriate Asis.Data_Decomposition.Type_Model_Kinds:
--      A_Simple_Static_Model
--      A_Simple_Dynamic_Model
--
---------------------------------------------------------------------------------------
```

-- 22.24 function Array_Iterator

```
---------------------------------------------------------------------------------------

    function Array_Iterator (Component : in Array_Component)
                            return Array_Component_Iterator;

---------------------------------------------------------------------------------------
-- Component   - Specifies an array component to be used for iteration
--
-- Returns an iterator poised to fetch the 1st component of an array.
--
```

```
-------------------------------------------------------------------------------
```

-- 22.25 function Component_Data_Stream

```
-------------------------------------------------------------------------------

    function Component_Data_Stream
                (Component   : in Record_Component;
                 Data_Stream : in Portable_Data)
                return Portable_Data;

    function Component_Data_Stream
                (Component   : in Array_Component;
                 Index       : in Asis.ASIS_Positive;
                 Data_Stream : in Portable_Data)
                return Portable_Data;

    function Component_Data_Stream
                (Component   : in Array_Component;
                 Indexes     : in Dimension_Indexes;
                 Data_Stream : in Portable_Data)
                return Portable_Data;

    function Component_Data_Stream
                (Iterator    : in Array_Component_Iterator;
                 Data_Stream : in Portable_Data)
                return Portable_Data;

-------------------------------------------------------------------------------
-- Component   - Specifies the component or discriminant to be extracted
-- Index       - Specifies an index, 1..Array_Length, within an array
-- Indexes     - Specifies a list of index values, there is one value for
--               each dimension of the array type, each index N is in the
--               range 1..Array_Length (Component, N);
-- Iterator    - Specifies the array component to extract
-- Data_Stream - Specifies the data stream from which to extract the result
--
-- Returns a data stream representing just the value of the chosen Component.
-- The return value is sliced from the data stream.  The Data_Stream shall
-- represent a value of the appropriate type.  It may have been obtained from
-- a file, extracted from another data stream, or artificially constructed
-- using the Construct_Artificial_Data_Stream operation.
--
-- An artificial data stream may raise ASIS_Inappropriate_Element (the Status
-- is Value_Error).  Only the constraint values are valid, once they
-- have been properly initialized, and can be safely extracted from an
-- artificial data stream.
--
-- Raises ASIS_Inappropriate_Element if given a Nil_Array_Component_Iterator
-- or one where Done(Iterator) = True.  The Status value is Data_Error.
-- The Diagnosis string will indicate the kind of error detected.
--
-- All non-Nil component values are appropriate.
--
-------------------------------------------------------------------------------
```

-- 22.26 function Component_Declaration

```
-------------------------------------------------------------------------------

    function Component_Declaration (Component : in Record_Component)
                                   return Asis.Declaration;

-------------------------------------------------------------------------------
-- Component   - Specifies the component to be queried
--
-- Returns an Asis.Declaration, which is either A_Component_Declaration
-- or A_Discriminant_Specification.  These values can be used to determine the
-- subtype, type, and base type of the record component.  The result may be an
-- explicit declaration made by the user, or, it may be an implicit
-- component declaration for an implementation-defined component (Reference Manual
-- 13.5.1(15)).
--
-- All non-Nil component values are appropriate.
--
-- Returns Element_Kinds:
--      A_Declaration
```

```
--
-- Returns Declaration_Kinds:
--      A_Component_Declaration
--      A_Discriminant_Specification
--
-----------------------------------------------------------------------------------------
```

-- 22.27 function Component_Indication

```
-----------------------------------------------------------------------------------------

    function Component_Indication (Component : in Array_Component)
                                  return Asis.Subtype_Indication;

-----------------------------------------------------------------------------------------
-- Component   - Specifies the component to be queried
--
-- Returns an Asis.Subtype_Indication.  These values can be used to determine
-- the subtype, type, and base type of the array components.
--
-- All non-Nil component values are appropriate.
--
-- Returns Element_Kinds:
--      A_Subtype_Indication
--
-----------------------------------------------------------------------------------------
```

-- 22.28 function All_Named_Components

```
-----------------------------------------------------------------------------------------

    function All_Named_Components (Type_Definition : in Asis.Type_Definition)
                                  return Asis.Defining_Name_List;

-----------------------------------------------------------------------------------------
-- Type_Definition - Specifies the record type definition to query
--
-- Returns a list of all discriminant and component entity names defined by
-- the record type.  All record type definitions are appropriate for this
-- operation.  This query provides a means for determining whether a field,
-- with a particular name, exists for some possible instance of the record
-- type.  This list does not include the names of implementation-defined
-- components (Reference Manual 13.5.1 (15)); those name have the form of
-- An_Attribute_Reference expression.
--
-- Appropriate Element_Kinds:
--      A_Type_Definition
--
-- Appropriate Type_Kinds:
--      A_Derived_Type_Definition       (derived from a record type)
--      A_Record_Type_Definition
--
-- Appropriate Asis.Data_Decomposition.Type_Model_Kinds:
--      A_Simple_Static_Model
--      A_Simple_Dynamic_Model
--      A_Complex_Dynamic_Model
--
-----------------------------------------------------------------------------------------
```

-- 22.29 function Array_Length

```
-----------------------------------------------------------------------------------------

    function Array_Length (Component : in Record_Component)
                          return Asis.ASIS_Natural;

    function Array_Length (Component : in Array_Component)
                          return Asis.ASIS_Natural;

-----------------------------------------------------------------------------------------
-- Component   - Specifies the component to query
--
-- Returns the number of components within an array valued component.  The
-- array subtype may be multidimensional.  The result treats the array as if
-- it were unidimensional.  It is the product of the 'Lengths of the
-- individual array dimensions.
--
-- All Is_Array(Component) = True values are appropriate.
```

```
--
-------------------------------------------------------------------------------
```

-- 22.30 function Array_Length

```
-------------------------------------------------------------------------------

    function Array_Length (Component : in Record_Component;
                           Dimension : in Asis.ASIS_Natural)
                          return Asis.ASIS_Natural;

    function Array_Length (Component : in Array_Component;
                           Dimension : in Asis.ASIS_Natural)
                          return Asis.ASIS_Natural;

-------------------------------------------------------------------------------
-- Component   - Specifies the component to query
-- Dimension   - Specifies the array dimension to query
--
-- Returns the number of components within an array valued component.  The
-- array subtype may be unidimensional.  The result is the 'Length(Dimension)
-- of the array.
--
-- All Is_Array(Component) = True values are appropriate.
--
-------------------------------------------------------------------------------
```

-- 22.31 function Size

```
-------------------------------------------------------------------------------

    function Size (Type_Definition : in Asis.Type_Definition)
                  return Asis.ASIS_Natural;

    function Size (Component : in Record_Component) return Asis.ASIS_Natural;

    function Size (Component : in Array_Component)  return Asis.ASIS_Natural;

-------------------------------------------------------------------------------
-- Type_Definition - Specifies a type definition, whose 'Size is desired
-- Component       - Specifies a component, whose 'Size is desired
--
-- Returns the minimum number of bits required to hold a simple static type,
-- the number of bits allocated to hold a record field, or the number of bits
-- allocated to hold each array component.
--
--|AN Application Note:
--|AN
--|AN For components, this is the number of bits allocated
--|AN within the composite value.  It may be greater than the number
--|AN of bits occupied by data values of this component type.
--|AN Also, the data value, when occupying more space than is
--|AN minimally required, may be preceded, followed, or surrounded by
--|AN padding bits which are necessary to fully occupy the space allotted.
--
-- All non-Nil component values are appropriate.
--
-- Appropriate Element_Kinds:
--      A_Type_Definition
--
-- Appropriate Asis.Data_Decomposition.Type_Model_Kinds:
--      A_Simple_Static_Model
--
-------------------------------------------------------------------------------
```

-- 22.32 function Size

```
-------------------------------------------------------------------------------

    function Size (Type_Definition : in Asis.Type_Definition;
                   Data_Stream     : in Portable_Data)
                  return Asis.ASIS_Natural;

-------------------------------------------------------------------------------
-- Type_Definition - Specifies the type definition to query
-- Data_Stream     - Specifies a data stream containing, at least, the complete
--                   set of discriminant or index constraints for the type
--
```

```
-- Returns the 'Size of a value of this type, with these constraints.  This is
-- the minimum number of bits that is needed to hold any possible value of the
-- given fully constrained subtype.  Only the constraint portion of the
-- Data_Stream is checked.
--
-- The Data_Stream may be a data stream or it may be an artificial
-- data stream created by the Construct_Artificial_Data_Stream operation.
--
-- Appropriate Element_Kinds:
--      A_Type_Definition
--
-- Appropriate Asis.Data_Decomposition.Type_Model_Kinds:
--      A_Simple_Static_Model
--      A_Simple_Dynamic_Model
--
---------------------------------------------------------------------------------------
```

-- 22.33 function Position

```
---------------------------------------------------------------------------------------

    function Position (Component : in Record_Component)
                      return Asis.ASIS_Natural;

    function Position (Component : in Array_Component;
                       Index     : in Asis.ASIS_Positive)
                      return Asis.ASIS_Natural;

    function Position (Component : in Array_Component;
                       Indexes   : in Dimension_Indexes)
                      return Asis.ASIS_Natural;

    function Position (Iterator : in Array_Component_Iterator)
                      return Asis.ASIS_Natural;

---------------------------------------------------------------------------------------
-- Component   - Specifies the component to query
-- Index       - Specifies a value in the range 1..Array_Length (Component),
--               the index of the component to query
-- Indexes     - Specifies a list of index values, there is one value for
--               each dimension of the array type, each index N is in the
--               range 1..Array_Length (Component, N);
-- Iterator    - Specifies a particular array component to query
--
-- Returns the System.Storage_Unit offset, from the start of the first storage
-- unit occupied by the enclosing composite type, of the first of the storage
-- units occupied by the Component.  The offset is measured in storage units.
--
-- All non-Nil component values are appropriate.  Raises
-- ASIS_Inappropriate_Element with a Status of Data_Error if any index is not
-- in the expected range or if Done (Iterator) = True.  The Status value will
-- be Data_Error.  The Diagnosis string will indicate the kind of error
-- detected.
--
---------------------------------------------------------------------------------------
```

-- 22.34 function First_Bit

```
---------------------------------------------------------------------------------------

    function First_Bit (Component : in Record_Component)
                       return Asis.ASIS_Natural;

    function First_Bit (Component : in Array_Component;
                        Index     : in Asis.ASIS_Positive)
                       return Asis.ASIS_Natural;

    function First_Bit (Component : in Array_Component;
                        Indexes   : in Dimension_Indexes)
                       return Asis.ASIS_Natural;

    function First_Bit (Iterator : in Array_Component_Iterator)
                       return Asis.ASIS_Natural;

---------------------------------------------------------------------------------------
```

```
-- Component   - Specifies the component to query
-- Index       - Specifies a value in the range 1..Array_Length (Component),
--               the index of the component to query
-- Indexes     - Specifies a list of index values, there is one value for
--               each dimension of the array type, each index N is in the
--               range 1..Array_Length (Component, N);
-- Iterator    - Specifies a particular array component to query
--
-- Returns the bit offset, from the start of the first of the storage units
-- occupied by the Component, of the first bit occupied by the Component.  The
-- offset is measured in bits.
--
-- All non-Nil component values are appropriate.  Raises
-- ASIS_Inappropriate_Element with a Status of Data_Error if any index is not
-- in the expected range or if Done (Iterator) = True.  The Status value will
-- be Data_Error.  The Diagnosis string will indicate the kind of error
-- detected.
--
-------------------------------------------------------------------------------------
```

-- 22.35 function Last_Bit

```
-------------------------------------------------------------------------------------

    function Last_Bit (Component : in Record_Component)
                  return Asis.ASIS_Natural;

    function Last_Bit (Component : in Array_Component;
                   Index     : in Asis.ASIS_Positive)
                  return Asis.ASIS_Natural;

    function Last_Bit (Component : in Array_Component;
                   Indexes   : in Dimension_Indexes)
                  return Asis.ASIS_Natural;

    function Last_Bit (Iterator : in Array_Component_Iterator)
                  return Asis.ASIS_Natural;

-------------------------------------------------------------------------------------
-- Component   - Specifies the component to query
-- Index       - Specifies a value in the range 1..Array_Length (Component),
--               the index of the component to query
-- Indexes     - Specifies a list of index values, there is one value for
--               each dimension of the array type, each index N is in the
--               range 1..Array_Length (Component, N);
-- Iterator    - Specifies a particular array component to query
--
-- Returns the bit offset, from the start of the first of the storage units
-- occupied by the Index'th Element, of the last bit occupied by the Element.
-- The offset is measured in bits.
--
-- All non-Nil component values are appropriate.  Raises
-- ASIS_Inappropriate_Element with a Status of Data_Error if any index is not
-- in the expected range or if Done (Iterator) = True.  The Status value will
-- be Data_Error.  The Diagnosis string will indicate the kind of error detected.
--
-------------------------------------------------------------------------------------
```

-- 22.36 function Portable_Constrained_Subtype

```
-------------------------------------------------------------------------------------
-- Generic for Data Stream Conversions
-------------------------------------------------------------------------------------

    generic
        -- Ada notation for a constrained subtype.
        -- type Constrained_Subtype (<>) is private;
        type Constrained_Subtype is private;
    function Portable_Constrained_Subtype
            (Data_Stream : in Portable_Data)
            return Constrained_Subtype;

-------------------------------------------------------------------------------------
-- Data_Stream - Specifies an extracted component of a record
--
-- Instantiated with an appropriate scalar type, (e.g., System.Integer, can be
-- used to convert a data stream to a value that can be directly examined).
```

```
--
-- Instantiated with a record type, can be used to convert a data stream to a
-- value that can be directly examined.
--
-- Instantiations with constrained array subtypes may not convert array values
-- if they were created using the Portable_Array_Type_1,
-- Portable_Array_Type_2, or Portable_Array_Type_3 interfaces.
--
-- May raise Constraint_Error if the subtype is a scalar and the converted
-- value is not in the subtype's range.
--
-------------------------------------------------------------------------------------
```

-- 22.37 function Construct_Artificial_Data_Stream

```
-------------------------------------------------------------------------------------

    function Construct_Artificial_Data_Stream
                (Type_Definition : in Asis.Type_Definition;
                 Data_Stream     : in Portable_Data;
                 Discriminant    : in Record_Component;
                 Value           : in Portable_Data)
                return Portable_Data;

-------------------------------------------------------------------------------------
-- Type_Definition - Specifies the record type definition for the record
--                   valued data stream being constructed
-- Data_Stream     - Specifies the data stream constructed so far; initially
--                   specified as the Nil_Portable_Data value
-- Discriminant    - Specifies the discriminant of the record type that is
--                   being set or changed
-- Value           - Specifies a data stream representing a single
--                   discriminant value of the appropriate type
--
-- Used to artificially construct a data stream which represents the
-- discriminant portion of a fully constrained value of the indicated record
-- type.  This operation is called once with a value for each discriminant of
-- the record type (the order in which the discriminants are specified is not
-- important).  The return value of each call is used as the input Data_Stream
-- for the next.
--
-- The resulting artificial data stream may be used solely for the purpose of
-- creating Record_Component values.  The values of any non-discriminant
-- fields are arbitrary and quite possibly invalid.  The resulting
-- component values may then be used for any purpose.  In particular, they may
-- be used to determine First_Bit, Last_Bit, and Size values for all record
-- discriminants and components.
--
-- Appropriate Element_Kinds:
--      A_Type_Definition
--
-- Appropriate Type_Kinds:
--      A_Derived_Type_Definition       (derived from a record type)
--      A_Record_Type_Definition
--
-- Raises ASIS_Inappropriate_Element, with a Status of Data_Error, if the
-- discriminant Value is inappropriate for the specified Discriminant.
--
-------------------------------------------------------------------------------------
```

```
private

    type Record_Component is
        record
            A : Boolean;
        end record;

    Nil_Record_Component : constant Record_Component := (A => False);

    type Array_Component is
        record
            A : Boolean;
        end record;

    Nil_Array_Component : constant Array_Component := (A => False);

    type Array_Component_Iterator is
        record
            Component : Array_Component;
            Index     : Asis.ASIS_Natural := 0;
        end record;

    Nil_Array_Component_Iterator : constant Array_Component_Iterator :=
       (Component => Nil_Array_Component,
        Index     => 0);

-----------------------------------------------------------------------------------------

end Asis.Data_Decomposition;
```

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

-- 23 package Asis.Data_Decomposition.Portable_Transfer

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
package Asis.Data_Decomposition.Portable_Transfer is
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- Asis.Data_Decomposition.Portable_Transfer
--
-- This package is part of the optional Asis.Data_Decomposition package.
-- It may or may not be present in all ASIS implementations.
--
-- It provides support for logging and delogging of application data using
-- ASIS.
--
-- These interfaces do not make use of the normal ASIS implementation.
-- This is a standalone package.  It is not necessary to initialize ASIS
-- before using this interface.
--
-- This interface is intended to be separate from the normal ASIS
-- implementation.  That is, linking with this interface does not cause the
-- full ASIS implementation to be linked with an application.  This keeps the
-- memory cost of this facility to a minimum.  However, it also means that the
-- normal ASIS exceptions, Status, and Diagnosis facilities are not available.
-- This interface may propagate exceptions other than those defined in
-- Asis.Errors.  In particular, the I/O facilities may raise any of the
-- normal Ada I/O exceptions from package Ada.Io_Exceptions.
--
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- Generics that create portable data streams.
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

-- 23.1 generic package Portable_Constrained_Subtype

```
-------------------------------------------------------------------------------

    generic
        -- Ada notation for a constrained subtype.
        -- type Constrained_Subtype (<>) is private;
        type Constrained_Subtype is private;
    package Portable_Constrained_Subtype is
        function Convert (Value : in Constrained_Subtype)
           return Asis.Data_Decomposition.Portable_Data;
    end Portable_Constrained_Subtype;

-------------------------------------------------------------------------------
-- Value   - Specifies a data value which is to be converted to a portable
--           data stream
--
-- Instantiated with a scalar subtype, or a fully constrained composite
-- subtype, this function will correctly convert values of that subtype to a
-- portable data value.  "Correctly" means that no information is lost.  No
-- guarantee is made that the result will not be larger (larger 'Size) than
-- the argument.
--
-- Instantiation of this generic, with arguments other than scalar types or
-- fully constrained subtypes, will fail to compile or will fail to execute
-- and raise Constraint_Error.
--
-------------------------------------------------------------------------------
```

-- 23.2 generic package Portable_Unconstrained_Record_Type

```
-------------------------------------------------------------------------------

    generic
        type Unconstrained_Record_Type is private;
    package Portable_Unconstrained_Record_Type is
        function Convert (Value : in Unconstrained_Record_Type)
                          return Asis.Data_Decomposition.Portable_Data;
    end Portable_Unconstrained_Record_Type;

-------------------------------------------------------------------------------
```

```
-- Value   - Specifies a record value which is to be converted to a portable
--           data stream
--
-- Instantiated with a record type, this function will correctly convert
-- values of that type to a portable data value.  "Correctly" means that no
-- information is lost.  No guarantee is made that the result will not be
-- larger (larger 'Size) than the argument.
--
-- This generic may also be instantiated with fully constrained subtypes.
-- However, the Portable_Constrained_Subtype generic is likely to be a more
-- efficient mechanism for these subtypes.
--
-- Instantiation of this generic, with unconstrained array types, will not
-- fail at compile time or at runtime.  However, no guarantee is made that
-- such instantiations will operate correctly; data could be lost.
--
-------------------------------------------------------------------------------------
```

-- 23.3 generic package Portable_Array_Type_1

```
-------------------------------------------------------------------------------------

    generic
        type Element_Type is private;
        type Index_Type is (<>);
        type Array_Type is array (Index_Type range <>) of Element_Type;
    package Portable_Array_Type_1 is
        function Convert (Value : in Array_Type)
            return Asis.Data_Decomposition.Portable_Data;
    end Portable_Array_Type_1;

-------------------------------------------------------------------------------------
```

-- 23.4 generic package Portable_Array_Type_2

```
-------------------------------------------------------------------------------------

    generic
        type Element_Type is private;
        type Index_Type_1 is (<>);
        type Index_Type_2 is (<>);
        type Array_Type is array (Index_Type_1 range <>,
                                    Index_Type_2 range <>) of Element_Type;
    package Portable_Array_Type_2 is
        function Convert (Value : in Array_Type)
           return Asis.Data_Decomposition.Portable_Data;
    end Portable_Array_Type_2;

-------------------------------------------------------------------------------------
```

-- 23.5 generic package Portable_Array_Type_3

```
-------------------------------------------------------------------------------------

    generic
        type Element_Type is private;
        type Index_Type_1 is (<>);
        type Index_Type_2 is (<>);
        type Index_Type_3 is (<>);
        type Array_Type is array (Index_Type_1 range <>,
                                  Index_Type_2 range <>,
                                  Index_Type_3 range <>) of Element_Type;
    package Portable_Array_Type_3 is
        function Convert (Value : in Array_Type)
            return Asis.Data_Decomposition.Portable_Data;
    end Portable_Array_Type_3;

-------------------------------------------------------------------------------------
-- Value   - Specifies an array value which is to be converted to a portable
--           data stream
--
-- Instantiated with an unconstrained array subtype, this function will
-- correctly convert values of that subtype to a portable data value.
-- "Correctly" means that no information is lost.  No guarantee is made that
-- the result will not be larger (larger 'Size) than the argument.
-- This interface will correctly capture the array index (dope vector)
-- information associated with the array value.
--
```

```
-- Use the Portable_Constrained_Subtype generic for statically constrained
-- array subtypes.
--
-- Array types with more than 3 dimensions can be converted to portable data
-- streams by placing them inside a discriminated record type (where the
-- discriminants are the dynamic array index constraints) and then converting
-- the record value.  The record will then contain the necessary array index
-- (dope vector) information.
--
---------------------------------------------------------------------------------------

end Asis.Data_Decomposition.Portable_Transfer;
```

Annex A
(informative)
Glossary

All terms which are defined in ISO/IEC 8652:1995 as Ada technical terms are used in the ASIS specification in full accordance with those definitions. See the Ada International Standard for definitions.

Ancestor. Ancestors of a library unit are itself, its parent, its parent's parent, and so on. (Standard is an ancestor of every library unit).

ASIS application. Any programming system or any set of software components making use of ASIS queries to obtain information about any set of Ada components.

ASIS implementation. All the hardware and software that implement the ASIS specification for a given Ada implementation and that provide the functionality required by the ASIS specification.

ASIS queries. Those subprogram interfaces (and only those) defined in the ASIS standard; these are supported by types, subtypes, and exceptions also defined in the ASIS standard. Thus, ASIS queries and supporting entities are together the ASIS interface. The following informal query classification is used by the ASIS community: "black-box" queries are those ASIS queries which produce information about compilation units and "white-box" queries are those ASIS queries which produce information about Elements; semantic queries are those ASIS queries which express semantic properties of ASIS Elements in terms of other Elements; structural queries are those ASIS queries which provide the top-down decomposition and reverse bottom-up composition of the compilation unit according to its syntax structure. (Note that semantic queries are generally named "Corresponding_..." or "Implicit_..." in the ASIS specification.)

Closure. A term commonly used instead of *needed units*.

Compilation unit. "The term *compilation unit* is used to refer to a compilation_unit. When the meaning is clear from context, the term is also used to refer to the library_item of a compilation_unit or to the proper_body of a subunit". [ISO/IEC 8652:1995, 10.1.1(9)]. ASIS says "ASIS compilation unit" when the intent is to stress, that the ASIS viewpoint on an Ada compilation unit is described in the ASIS standard. Note, that the term "compilation unit" can refer to either syntactical category "compilation_unit" or to the library_item of a compilation_unit or to the proper_body of a subunit (that is, the compilation_unit without the context_clause and the **separate** (parent_unit_name)).

Compilation_Unit [type]. An ASIS private type for which values denote an Ada compilation unit or configuration pragma from the environment denoted by some open ASIS context. A non-nil value of the Compilation_Unit type also contains information about some physical object from the "external world" treated by the underlying Ada implementation as the corresponding Ada compilation unit or as a result of compiling a configuration pragma.

Container. Logical collection of ASIS compilation units. For example, some container can hold compilation units which include Ada predefined types, another container can hold implementation-defined packages. Containers provide the implementation-defined way of grouping the compilation units accessible for an ASIS application through the ASIS queries.

Container [type]. An ASIS private type for which values denote a set of compilation units being a subset of the set of compilation units making up a context.

Context. Defines a set of compilation units and configuration pragmas processed by an ASIS application. ASIS provides any information from a context by treating this set as if its elements make up an environment declarative part by modeling some view (most likely one of the views of the underlying Ada implementation) on the environment. ASIS may process several different contexts at a time.

Context [type]. An ASIS private type for which values denote a set of compilation units considered by ASIS as making up an Ada environment declarative part from which to provide information.

Dependent. Dependents of a compilation unit are all the compilation units that depend semantically on it, either directly or indirectly. A is a dependent of B, if B is a supporter of A.

Descendants. Descendants of a library unit relation are the inverse of the ancestor relation.

Element. A common abstraction used by ASIS to denote the syntax components (both explicit and implicit) of ASIS compilation units. The term Element is also used as the synonym for "the value of the ASIS Element type". See also "Explicit element" and "Implicit element".

Element [type]. An ASIS private type, whose values represent the syntax components (both explicit and implicit) of ASIS compilation units.

Environment. "Each compilation unit submitted to the compiler is compiled in the context of an environment declarative_part (or simply environment), which is a conceptual declarative_part that forms the outermost declarative region of the context of any compilation. At run time, an environment forms the declarative_part of the body of the environment task of a partition." [ISO/IEC 8652:1995(E), 10.1.4(1)]. Note that the mechanisms for creating an environment and for adding and replacing compilation units within an environment are implementation-defined.

Explicit element. An ASIS Element, representing a language construct that appears explicitly in the program text for the compilation unit (e.g., an explicit declaration).

Extension. Non-standard facilities (other library units, non-standard children of standard ASIS library units, subprograms, etc.) which provide additional information from ASIS types, or modify the behavior of otherwise standard ASIS facilities to provide alternative or additional functionality.

Family. The family of a given unit is defined as the set of compilation units that comprise the given unit's declaration, body, descendants, and subunits (and subunits of subunits and descendants, etc.).

Id. A way of identifying a particular element, from a particular compilation unit, from a particular context.

Id [type]. An ASIS private type implementing the Id abstraction. The values of this type can be written to files. These values can be read back from files and converted into values of the Element type with the use of a suitable open context.

Implementor. A company, institution, or other group (such as a vendor) who develops an ASIS implementation; thus an ASIS implementor. There are also Ada implementors, who provide Ada compilation systems; and there are ASIS-based tool (or, ASIS Application) implementors, who develop tools which are based upon the ASIS standard.

Implicit element. An ASIS Element, representing a language construct that does not exist in the program text for the compilation unit, but could occur at a given place in the program text as a consequence of the semantics of another construct, (e.g., an implicit declaration, a generic instantiation).

Line. The logical representation of a line of text from the source code of the external representation of a compilation unit.

Line [type]. An ASIS private type for the ASIS Line abstraction. The values of the Line type represent the lines of text from the source code of the external representation of compilation units.

Needed Units. The needed units of a given compilation unit is a set of compilation units ultimately needed by the given compilation unit to make up or to be included in a completed partition.

Optional functionality. The subset of ASIS facilities that are explicitly identified in the ASIS standard as optional which may legitimately be omitted from a Basic Conforming ASIS implementation, but shall be included in any Fully Conforming ASIS implementation, unless stated otherwise in the ASIS specification.

Queries. See ASIS queries.

Relation (between ASIS Compilation Units). Semantic relationships between compilation units (as discussed in chapter 10 of ISO/IEC 8652:1995). The Relation_Kinds type enumerates the kinds of relations that can exist between compilation units. See also Dependent, Extended Family, and Supporter.

Required functionality. The subset of ASIS facilities which are not explicitly identified in the ASIS standard as optional which shall be included in a Basic or Fully Conforming ASIS implementation, unless stated otherwise in the ASIS specification.

Semantic queries. See ASIS queries.

Structural queries. See ASIS queries.

Supporter. Supporters of a compilation unit are units on which it semantically depends, either directly or indirectly. B is a supporter of A, if A is a dependent of B.

Annex B
(informative)
ASIS application examples

B.1 Use to traverse compilation unit

```
-- The following example of an ASIS tool prompts the user for the name of
-- an Ada package specification, traverses that compilation unit, and prints
-- all explicit declarations with their kind. ASIS Application Example:

with Asis;                                                            --  3.0
with Asis.Errors;                                                     --  4.0
with Asis.Exceptions;                                                 --  5.0
with Asis.Implementation;                                             --  6.0
with Asis.Ada_Environments;                                           --  8.0
with Asis.Compilation_Units;                                          -- 10.0
with Asis.Elements;                                                   -- 13.0
with Asis.Iterator;                                                   -- 14.0
with Asis.Declarations;                                               -- 15.0
with Ada.Wide_Text_Io; use Ada.Wide_Text_Io;

Procedure ASIS_Application_Example is

   My_Context                : Asis.Context;                          --  3.7
   My_Unit                   : Asis.Compilation_Unit;                 --  3.10
   Unit_Name                 : Wide_String ( 1 .. 100 );
   Unit_Name_Length          : Natural;

   procedure Report_Declarations (Unit : in Asis.Compilation_Unit) is    --  3.10

      My_Element             : Asis.Element;                          --  3.6
      My_Control             : Asis.Traverse_Control                  --  3.13
                                   := Asis.Continue;
      My_State               : Boolean := True;

      procedure Process_Element
                   (Elem      : in     Asis.Element;                  --  3.6
                    Control   : in out Asis.Traverse_Control;         --  3.13
                    State     : in out Boolean);

      procedure No_Op
                   (Elem      : in     Asis.Element;                  --  3.6
                    Control   : in out Asis.Traverse_Control;         --  3.13
                    State     : in out Boolean);

      procedure Find_and_Print_Declarations is new
                    Asis.Iterator.Traverse_Element                    -- 14.1
                       (Boolean, Process_Element, No_Op);

      procedure No_Op
                   (Elem      : in     Asis.Element;                  --  3.6
                    Control   : in out Asis.Traverse_Control;         --  3.13
                    State     : in out Boolean) is
      begin
         null;
      end No_Op;


      procedure Process_Element
                   (Elem      : in     Asis.Element;                  --  3.6
                    Control   : in out Asis.Traverse_Control;         --  3.13
                    State     : in out Boolean) is

         package Kind_Io is new Ada.Wide_Text_Io.Enumeration_Io
                             (Asis.Declaration_Kinds);                --  3.9.4

         Decl_Kind : Asis.Declaration_Kinds :=                        --  3.9.4
             Asis.Elements.Declaration_Kind (Elem);                   -- 13.9
```

```
      begin -- Process_Element

         case Decl_Kind is

            when Asis.Not_A_Declaration => null;                         --  3.9.4

            when others =>

                if not Asis."="                                          --  3.9.6
                          (Asis.Elements.Declaration_Origin (Elem),      -- 13.11
                           Asis.An_Explicit_Declaration) then            --  3.9.6
                   return;
                end if;

                declare

                   Name_List : Asis.Defining_Name_List                   --  3.8
                          := Asis.Declarations.Names (Elem);             -- 15.1

                begin
                   for I in Name_List'Range loop
                       Put (Asis.Declarations.Defining_Name_Image
                           (Name_List (I)));                             -- 15.2
                       Put (" (is kind) ");
                       Kind_Io.Put (Decl_Kind);
                       New_Line;
                   end loop;
                end;

         end case;

      end Process_Element;

   begin -- Report_Declarations
      My_Element   := Asis.Elements.Unit_Declaration ( Unit );           -- 13.1
      Find_and_Print_Declarations (My_Element, My_Control, My_State);
   end Report_Declarations;

begin -- ASIS_Application_Example

   Asis.Implementation.Initialize;                                       --  6.6
   Asis.Ada_Environments.Associate(My_Context, "My Context");            --  8.3
   Asis.Ada_Environments.Open (My_Context);                              --  8.4

   Put_Line ("Type the name of an Ada package specification");
   Get_Line (Unit_Name, Unit_Name_Length);

   My_Unit := Asis.Compilation_Units.Library_Unit_Declaration            -- 10.8
                  ( Unit_Name ( 1 .. Unit_Name_Length), My_Context );

   if Asis.Compilation_Units.Is_Nil ( My_Unit )                          -- 10.15
   then
      Put ("Context does not contain the requested unit: ");
      Put (Unit_Name ( 1 .. Unit_Name_Length));
      New_Line;
   else
      Put ("Context contains the requested unit: ");
      Put (Unit_Name ( 1 .. Unit_Name_Length));
      New_Line;
      Report_Declarations ( My_Unit );
      New_Line;
   end if;

   Asis.Ada_Environments.Close( My_Context );                            --  8.5
   Asis.Ada_Environments.Dissociate( My_Context );                       --  8.6
   Asis.Implementation.Finalize;                                         --  6.8
```

```
exception

   when   Asis.Exceptions.ASIS_Inappropriate_Context                   --  5.0
        | Asis.Exceptions.ASIS_Inappropriate_Container                 --  5.0
        | Asis.Exceptions.ASIS_Inappropriate_Compilation_Unit          --  5.0
        | Asis.Exceptions.ASIS_Inappropriate_Element                   --  5.0
        | Asis.Exceptions.ASIS_Inappropriate_Line                      --  5.0
        | Asis.Exceptions.ASIS_Inappropriate_Line_Number               --  5.0
        | Asis.Exceptions.ASIS_Failed                                  --  5.0
        =>

        Put (Asis.Implementation.Diagnosis);                           --  6.10
        New_Line;
        Put ("Status Value is ");
        Put (Asis.Errors.Error_Kinds'Wide_Image                        --  4.1
                        (Asis.Implementation.Status));                 --  6.9
        New_Line;

   when others =>

        Put_Line ("ASIS Application failed because of non-ASIS reasons");

end ASIS_Application_Example;

-- ASIS Application Example using following package specification named asis_test:

--         package asis_test is
--
--              type T is ( A, B, C);
--
--              S : integer := T'BASE'SIZE ;
--
--         end asis_test ;


-- Result of executing ASIS Application Example:

--      Type the name of an Ada package specification
--      asis_test
--      Context contains the requested unit: asis_test
--      asis_test (is kind) A_PACKAGE_DECLARATION
--      T (is kind) AN_ORDINARY_TYPE_DECLARATION
--      A (is kind) AN_ENUMERATION_LITERAL_SPECIFICATION
--      B (is kind) AN_ENUMERATION_LITERAL_SPECIFICATION
--      C (is kind) AN_ENUMERATION_LITERAL_SPECIFICATION
--      S (is kind) A_VARIABLE_DECLARATION
```

B.2 Use to build call tree

```
-- This example prints call tree information (i.e., a list of all procedure,
-- function, and entry calls made within a compilation unit) for each compilation
-- unit in the context. The output format is of the form:

-- <Calling_Compilation_Unit> (calls) <Called_Program_Unit> at line <Line_Number>
-- where:

-- <Calling_Compilation_Unit>  is the Expanded Name of the Unit making the call
-- <Called_Program_Unit>       is the name of the program unit being called
-- <Line_Number>               is the first line number of the call in the source file


with Asis;                                                                  --  3.0
with Asis.Errors;                                                           --  4.0
with Asis.Exceptions;                                                       --  5.0
with Asis.Implementation;                                                   --  6.0
with Asis.Ada_Environments;                                                 --  8.0
with Asis.Compilation_Units;                                                -- 10.0
with Asis.Elements;                                                         -- 13.0
with Asis.Iterator;                                                         -- 14.0
with Asis.Declarations;                                                     -- 15.0
with Asis.Expressions;                                                      -- 17.0
with Asis.Statements;                                                       -- 18.0
with Asis.Text;                                                             -- 20.0

with Ada.Wide_Text_Io; use Ada.Wide_Text_Io;

procedure ASIS_Call_Tree_Example is

   My_Context              : Asis.Context;                                  --  3.7

   procedure No_Op
              (Elem      : in     Asis.Element;                             --  3.6
               Control   : in out Asis.Traverse_Control;                    --  3.13
               State     : in out Boolean);

   procedure Report_Calls
              (An_Element: in     Asis.Element;                             --  3.6
               Control   : in out Asis.Traverse_Control;                    --  3.13
               State     : in out Boolean);

   procedure Print_Call_Tree is new
               Asis.Iterator.Traverse_Element                               -- 14.1
                  (Boolean, Report_Calls, No_Op);

   procedure No_Op
              (Elem      : in     Asis.Element;                             --  3.6
               Control   : in out Asis.Traverse_Control;                    --  3.13
               State     : in out Boolean) is
   begin
      null;
   end No_Op;

   procedure Output_Call (Caller : Asis.Element;                            --  3.6
                          Callee : Asis.Declaration) is                     --  3.8

      Calling_Cu   : Asis.Compilation_Unit;                                 --  3.10
      Calling_Unit : Asis.Declaration;                                      --  3.8
```

```
   begin -- Output_Call

      Calling_Cu := Asis.Elements.Enclosing_Compilation_Unit (Caller);     -- 13.2

      if Asis.Compilation_Units.Is_Nil (Calling_Cu) then                   -- 10.15
         Put ("An_Unknown_Unit");
      else
         Put (Asis.Compilation_Units.Unit_Full_Name (Calling_Cu));         -- 10.19
      end if;

      Put (" (calls) ");
      Put (Asis.Declarations.Defining_Name_Image                           -- 15.2
                   (Asis.Declarations.Names (Callee) (1)));                -- 15.1
      Put (" at line ");
      Put (Asis.Text.Line_Number'Wide_Image                                -- 20.2
                (Asis.Text.First_Line_Number (Caller)));                   -- 20.8
      New_Line;

   end Output_Call;

   procedure Report_Calls (An_Element : in Asis.Element;                   --  3.6
                           Control    : in out Asis.Traverse_Control;      --  3.13
                           State      : in out Boolean) is

      Callee : Asis.Declaration;                                           --  3.8

   begin -- Report_Calls

      case Asis.Elements.Element_Kind (An_Element) is                      -- 13.6

         when Asis.An_Expression =>                                        --  3.9.1
            case Asis.Elements.Expression_Kind (An_Element) is             -- 13.21
               when Asis.A_Function_Call =>                                --  3.9.17

                  Callee := Asis.Expressions.Corresponding_Called_Function
                                  (An_Element);                            -- 17.29

                  if not Asis.Elements.Is_Nil (Callee) then                -- 13.29
                     Output_Call (An_Element, Callee);
                  end if;

               when others =>
                  null;
            end case;

         when Asis.A_Statement =>                                          --  3.9.1
            case Asis.Elements.Statement_Kind (An_Element) is              -- 13.25

               when Asis.A_Procedure_Call_Statement |                      --  3.9.20
                    Asis.An_Entry_Call_Statement =>                        --  3.9.20

                  Callee := Asis.Statements.Corresponding_Called_Entity
                                (An_Element);                              -- 18.25

                  if not Asis.Elements.Is_Nil (Callee) then                -- 13.29
                     Output_Call (An_Element, Callee);
                  end if;

               when others =>
                  null;
            end case;

         when others =>
            null;

      end case;

end Report_Calls;
```

```
   procedure Process_Units (Unit_List : in Asis.Compilation_Unit_List) is  --  3.11

      Control : Asis.Traverse_Control := Asis.Continue;                    --  3.13
      State   : Boolean := True;

   begin

      for I in Unit_List'Range loop

         case Asis.Compilation_Units.Unit_Origin (Unit_List (I)) is        -- 10.3
            when Asis.An_Application_Unit =>                               --  3.12.3
               New_Line;
               Put_Line ("Processing Unit: " &
                                 Asis.Compilation_Units.Unit_Full_Name
                                    (Unit_List (I)));                      -- 10.19
               Print_Call_Tree (Asis.Elements.Unit_Declaration             -- 13.1
                                    (Unit_List (I)), Control, State);
            when others =>
               null;
         end case;

      end loop;

   end Process_Units;

begin -- ASIS_Call_Tree_Example

   Asis.Implementation.Initialize;                                         --  6.6
   Asis.Ada_Environments.Associate (My_Context, "My Context");             --  8.3
   Asis.Ada_Environments.Open (My_Context);                                --  8.4

   Process_Units (Asis.Compilation_Units.Compilation_Units (My_Context));  -- 10.10

   Asis.Ada_Environments.Close (My_Context);                               --  8.5
   Asis.Ada_Environments.Dissociate (My_Context);                          --  8.6
   Asis.Implementation.Finalize;                                           --  6.8

exception

   when Asis.Exceptions.ASIS_Inappropriate_Context                         --  5.0
        | Asis.Exceptions.ASIS_Inappropriate_Container                     --  5.0
        | Asis.Exceptions.ASIS_Inappropriate_Compilation_Unit              --  5.0
        | Asis.Exceptions.ASIS_Inappropriate_Element                       --  5.0
        | Asis.Exceptions.ASIS_Inappropriate_Line                          --  5.0
        | Asis.Exceptions.ASIS_Inappropriate_Line_Number                   --  5.0
        | Asis.Exceptions.ASIS_Failed                                      --  5.0
      =>

      Put (Asis.Implementation.Diagnosis);                                 --  6.10
      New_Line;
      Put ("Status Value is ");
      Put (Asis.Errors.Error_Kinds'Wide_Image                              --  4.1
                      (Asis.Implementation.Status));                       --  6.9
      New_Line;

   when others =>

      Put_Line ("Asis Application failed because of non-ASIS reasons");

end ASIS_Call_Tree_Example;
```

```
-- The ASIS_Call_Tree_Example is demonstrated using a context containing the
-- following compilation units:

--          package P is
--
--             procedure P1;
--             procedure P2;
--             procedure P3(X : integer);
--             function F1 return integer;
--
--          end;

--          package body P is
--
--             procedure P1 is separate;
--             procedure P2 is separate;
--             procedure P3(X : integer) is separate;
--             function F1 return integer is separate;
--
--         begin
--             P1;
--         end;
--
--         separate (P)
--         function F1 return integer is
--         begin
--            return 0;
--         end;
--
--         separate (P)
--         procedure P1 is
--            x : integer := F1;
--         begin
--            P2;
--            P3(x);
--         end;
--
--         separate (P)
--         procedure P2 is
--         begin
--            P3(F1);
--         end;
--
--         separate (P)
--         procedure P3(X : integer) is
--         begin
--               null;
--         end;

-- Given a context containing the above set of compilation units, the sample
-- output resulting from the execution of ASIS_Call_Tree_Example is:

--         Processing Unit: P
--
--         Processing Unit: P
--         P (calls) P1 at line  9
--
--         Processing Unit: P.F1
--
--         Processing Unit: P.P1
--         P.P1 (calls) F1 at line  3
--         P.P1 (calls) P2 at line  5
--         P.P1 (calls) P3 at line  6
--
--         Processing Unit: P.P2
--         P.P2 (calls) P3 at line  4
--         P.P2 (calls) F1 at line  4
--
--         Processing Unit: P.P3
```


Annex C
(informative)
Miscellaneous ASIS I/O and IDL approaches

This Annex contains examples of miscellaneous approaches to deal with I/O and IDL issues. The first two portions contain packages for application I/O of ASIS types: Portable_Data and Id. The third portion contains an approach to providing an ASIS IDL. Annex C consists of:

C.1	Portable_Data_Io	I/O for type Portable_Data from Asis.Data_Decomposition
C.2	Package Id_Io	I/O for type Id from Asis.Ids
C.3	Implementation approach for IDL	

C.1 package Portable_Data_Io

```
------------------------------------------------------------------------------
------------------------------------------------------------------------------
-- Annex C.1  package Portable_Data_Io
------------------------------------------------------------------------------
------------------------------------------------------------------------------
with Asis;
with Asis.Data_Decomposition.Portable_Transfer;
with Ada.Io_Exceptions;
with Ada.Sequential_Io;
------------------------------------------------------------------------------
package Portable_Data_Io is
------------------------------------------------------------------------------
-- Portable_Data_Io
--
-- A package for reading/writing sequential portable data streams.
------------------------------------------------------------------------------
-- It provides support for logging and delogging of application data using
-- ASIS.
--
-- These interfaces do not make use of the normal ASIS implementation.
-- This is a standalone package.  It is not necessary to initialize ASIS
-- before using this interface.
--
-- This interface is intended to be separate from the normal ASIS
-- implementation.  That is, linking with this interface does not cause the
-- full ASIS implementation to be linked with an application.  This keeps the
-- memory cost of this facility to a minimum.  However, it also means that the
-- normal ASIS exceptions, Status, and Diagnosis facilities are not available.
-- This interface can propagate exceptions other than those defined in
-- Asis.Errors.  In particular, the I/O facilities can raise any of the
-- normal Ada I/O exceptions from package Ada.Io_Exceptions.
--
------------------------------------------------------------------------------
-- Annex C.1.1  type File_Type
------------------------------------------------------------------------------

    type File_Type is limited private;

------------------------------------------------------------------------------
-- Annex C.1.2  type File_Mode
------------------------------------------------------------------------------

    type File_Mode is (In_File,
                       Out_File);
```

```
-------------------------------------------------------------------------------
-- Annex C.1.3  procedure Create
-------------------------------------------------------------------------------

    procedure Create (File           : in out File_Type;
                      Mode           : in     File_Mode := Out_File;
                      Name           : in     Wide_String    := "";
                      Form           : in     Wide_String    := "");

-------------------------------------------------------------------------------
-- Annex C.1.4  procedure Open
-------------------------------------------------------------------------------

    procedure Open   (File           : in out File_Type;
                      Mode           : in     File_Mode;
                      Name           : in     Wide_String;
                      Form           : in     Wide_String := "");

-------------------------------------------------------------------------------
-- Annex C.1.5  procedure Close
-------------------------------------------------------------------------------

    procedure Close  (File           : in out File_Type);

-------------------------------------------------------------------------------
-- Annex C.1.6  procedure Delete
-------------------------------------------------------------------------------

    procedure Delete (File           : in out File_Type);

-------------------------------------------------------------------------------
-- Annex C.1.7  procedure Reset
-------------------------------------------------------------------------------

    procedure Reset  (File           : in out File_Type;
                      Mode           : in     File_Mode);
    procedure Reset  (File           : in out File_Type);

-------------------------------------------------------------------------------
-- Annex C.1.8  function Mode
-------------------------------------------------------------------------------

    function Mode (File              : in File_Type) return File_Mode;

-------------------------------------------------------------------------------
-- Annex C.1.9  function Name
-------------------------------------------------------------------------------

    function Name (File              : in File_Type) return Wide_String;

-------------------------------------------------------------------------------
-- Annex C.1.10  function Form
-------------------------------------------------------------------------------

    function Form (File              : in File_Type) return Wide_String;

-------------------------------------------------------------------------------
-- Annex C.1.11  function Is_Open
-------------------------------------------------------------------------------

    function Is_Open (File           : in File_Type) return Boolean;

-------------------------------------------------------------------------------
-- Annex C.1.12  function Read
-------------------------------------------------------------------------------

    --  Read operations for single value and array values:

    function Read (File              : in File_Type)
        return Asis.Data_Decomposition.Portable_Value;

    function Read (File              : in File_Type)
        return Asis.Data_Decomposition.Portable_Data;
```

```
------------------------------------------------------------------------------
-- Annex C.1.13  procedure Write
------------------------------------------------------------------------------

--  Write operations for single value and array values:

   procedure Write (File             : in File_Type;
                    Data_Stream      : in Asis.Data_Decomposition.Portable_Value);

   procedure Write (File             : in File_Type;
                    Data_Stream      : in Asis.Data_Decomposition.Portable_Data);

------------------------------------------------------------------------------
-- Annex C.1.14  generic package Write_Data_Type
------------------------------------------------------------------------------

    generic
        type Data_Type is private;
        with function Convert (Value : in Data_Type)
            return Asis.Data_Decomposition.Portable_Data;
        -- Instantiated from Asis.Data_Decomposition.Portable_Transfer.Convert
    package Write_Data_Type is
        procedure Write (File        : in File_Type;
                         Value       : in Data_Type);

        --  Convenience interface. Can be more efficient, but is
        --  exactly equivalent to:
        --
        --      Write (File, Convert(Value));
        --
    end Write_Data_Type;

------------------------------------------------------------------------------
-- Annex C.1.15  function End_Of_File
------------------------------------------------------------------------------

    function End_Of_File (File       : in File_Type) return Boolean;

------------------------------------------------------------------------------
-- Annex C.1.16  Exceptions
------------------------------------------------------------------------------

    Status_Error : exception renames Ada.Io_Exceptions.Status_Error;
    Mode_Error   : exception renames Ada.Io_Exceptions.Mode_Error;
    Name_Error   : exception renames Ada.Io_Exceptions.Name_Error;
    Use_Error    : exception renames Ada.Io_Exceptions.Use_Error;
    Device_Error : exception renames Ada.Io_Exceptions.Device_Error;
    End_Error    : exception renames Ada.Io_Exceptions.End_Error;
    Data_Error   : exception renames Ada.Io_Exceptions.Data_Error;

private
    package Seq_Io is new Ada.Sequential_Io
(Asis.Data_Decomposition.Portable_Value);
    type File_Type is new Seq_Io.File_Type;

------------------------------------------------------------------------------

end Portable_Data_Io;
```

C.2 package Asis.Ids.Id_Io

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- Annex C.2  package Asis.Ids.Id_Io
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
with Asis;
with Asis.Ids;
with Ada.Direct_Io;
with Ada.Io_Exceptions;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
package Id_Io is
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- Asis.Ids.Id_Io provides Id I/O Facilities.
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- This interface is a copy of the Ada.Direct_Io interface.
-- The internals of this package are implementation dependent and the
-- amount of space taken, in an Id file, by an ASIS Id value is variable.
-- An ASIS Id value has a fixed 'Size, but, the size of the data represented
-- by that Id value can be arbitrarily large. It can contain access values.
-------------------------------------------------------------------------------
-- Annex C.2.1  type File_Type
-------------------------------------------------------------------------------

    type File_Type is limited private;

-------------------------------------------------------------------------------
-- Annex C.2.2  type File_Mode
-------------------------------------------------------------------------------

    type File_Mode is (In_File,
                       Inout_File,
                       Out_File);

    --  Count has an ASIS implementation-defined upper bound.
    --  Note: An Id can take up more than one "slot" in an Id file.
    --  The 1st Id in a file will probably be at From => 1, but, the 2nd
    --  Id in a file is not necessarily at From => 2.

-------------------------------------------------------------------------------
-- Annex C.2.3  type Count
-------------------------------------------------------------------------------

    type Count is new Integer range 0 ..
        Asis.Implementation_Defined_Integer_Constant;

-------------------------------------------------------------------------------
-- Annex C.2.4  type Positive_Count
-------------------------------------------------------------------------------

    subtype Positive_Count is Count range 1 .. Count'Last;

-------------------------------------------------------------------------------
-- Annex C.2.5  procedure Create
-------------------------------------------------------------------------------

    procedure Create (File      : in out File_Type;
                      Mode      : in     File_Mode := Inout_File;
                      Name      : in     Wide_String    := "";
                      Form      : in     Wide_String    := "");

-------------------------------------------------------------------------------
-- Annex C.2.6  procedure Open
-------------------------------------------------------------------------------

    procedure Open (File        : in out File_Type;
                    Mode        : in     File_Mode;
                    Name        : in     Wide_String;
                    Form        : in     Wide_String := "");
```

```
-------------------------------------------------------------------------------
-- Annex C.2.7  procedure Close
-------------------------------------------------------------------------------

    procedure Close (File       : in out File_Type);

-------------------------------------------------------------------------------
-- Annex C.2.8  procedure Delete
-------------------------------------------------------------------------------

    procedure Delete (File      : in out File_Type);

-------------------------------------------------------------------------------
-- Annex C.2.9  procedure Reset
-------------------------------------------------------------------------------

    procedure Reset (File       : in out File_Type; Mode : File_Mode);
    procedure Reset (File       : in out File_Type);

-------------------------------------------------------------------------------
-- Annex C.2.10 function Mode
-------------------------------------------------------------------------------

    function Mode (File         : in File_Type) return File_Mode;

-------------------------------------------------------------------------------
-- Annex C.2.11 function Name
-------------------------------------------------------------------------------

    function Name (File         : in File_Type) return Wide_String;

-------------------------------------------------------------------------------
-- Annex C.2.12 function Form
-------------------------------------------------------------------------------

    function Form (File         : in File_Type) return Wide_String;

-------------------------------------------------------------------------------
-- Annex C.2.13 function Is_Open
-------------------------------------------------------------------------------

    function Is_Open (File      : in File_Type) return Boolean;

-------------------------------------------------------------------------------
-- Annex C.2.14 procedure Read
-------------------------------------------------------------------------------
-- Read operations

    procedure Read (File        : in  File_Type;
                    Item        : out Asis.Ids.Id;
                    From        : in  Positive_Count);
    procedure Read (File        : in  File_Type;
                    Item        : out Asis.Ids.Id);
    procedure Read (File        : in  File_Type;
                    Item        : out Asis.Element;
                    From        : in  Positive_Count;
                    Context     : in  Asis.Context);
    procedure Read (File        : in  File_Type;
                    Item        : out Asis.Element;
                    Context     : in  Asis.Context);
```

```
-------------------------------------------------------------------------------
-- Annex C.2.15 procedure Write
-------------------------------------------------------------------------------
-- Write operations

   procedure Write (File        : in File_Type;
                    Item        : in Asis.Ids.Id;
                    To          : in Positive_Count);
    procedure Write (File       : in File_Type;
                    Item        : in Asis.Ids.Id);
    procedure Write (File       : in File_Type;
                    Item        : in Asis.Element;
                    To          : in Positive_Count);
    procedure Write (File       : in File_Type;
                    Item        : in Asis.Element);

-------------------------------------------------------------------------------
-- Annex C.2.16 procedure Set_Index
-------------------------------------------------------------------------------

    procedure Set_Index (File   : in File_Type;
                         To     : in Positive_Count);

-------------------------------------------------------------------------------
-- Annex C.2.17 function Index
-------------------------------------------------------------------------------

    function Index (File        : in File_Type) return Positive_Count;

-------------------------------------------------------------------------------
-- Annex C.2.18 function Size
-------------------------------------------------------------------------------

    function Size (File         : in File_Type) return Count;

-------------------------------------------------------------------------------
-- Annex C.2.19 function End_Of_File
-------------------------------------------------------------------------------

    function End_Of_File (File  : File_Type) return Boolean;

-------------------------------------------------------------------------------
-- Annex C.2.20 Exceptions
-------------------------------------------------------------------------------

    -- Exceptions

    Status_Error : exception renames Ada.Io_Exceptions.Status_Error;
    Mode_Error   : exception renames Ada.Io_Exceptions.Mode_Error;
    Name_Error   : exception renames Ada.Io_Exceptions.Name_Error;
    Use_Error    : exception renames Ada.Io_Exceptions.Use_Error;
    Device_Error : exception renames Ada.Io_Exceptions.Device_Error;
    End_Error    : exception renames Ada.Io_Exceptions.End_Error;
    Data_Error   : exception renames Ada.Io_Exceptions.Data_Error;

private
    --  ASIS implementation dependent.
    package Ids_Id_Io is new Ada.Direct_Io (Asis.Ids.Id);
    type File_Type is new Ids_Id_Io.File_Type;

-------------------------------------------------------------------------------

end Id_Io;
```

C.3 Implementation approach for IDL

An ASIS application can use the required ASIS interfaces within a Common Object Request Broker Architecture (CORBA) networked environment using the Interface Definition Language (IDL).

Figure C-1 depicts how the ASIS CORBA clients and servers can be created from the ASIS API. ASIS in IDL can be obtained by reverse engineering the ASIS API expressed in this International Standard. This can be done either manually or through automated tools. Once the ASIS in IDL is created, CORBA clients and servers can be created via the automatic generation of IDL described in the remaining portion of this annex.

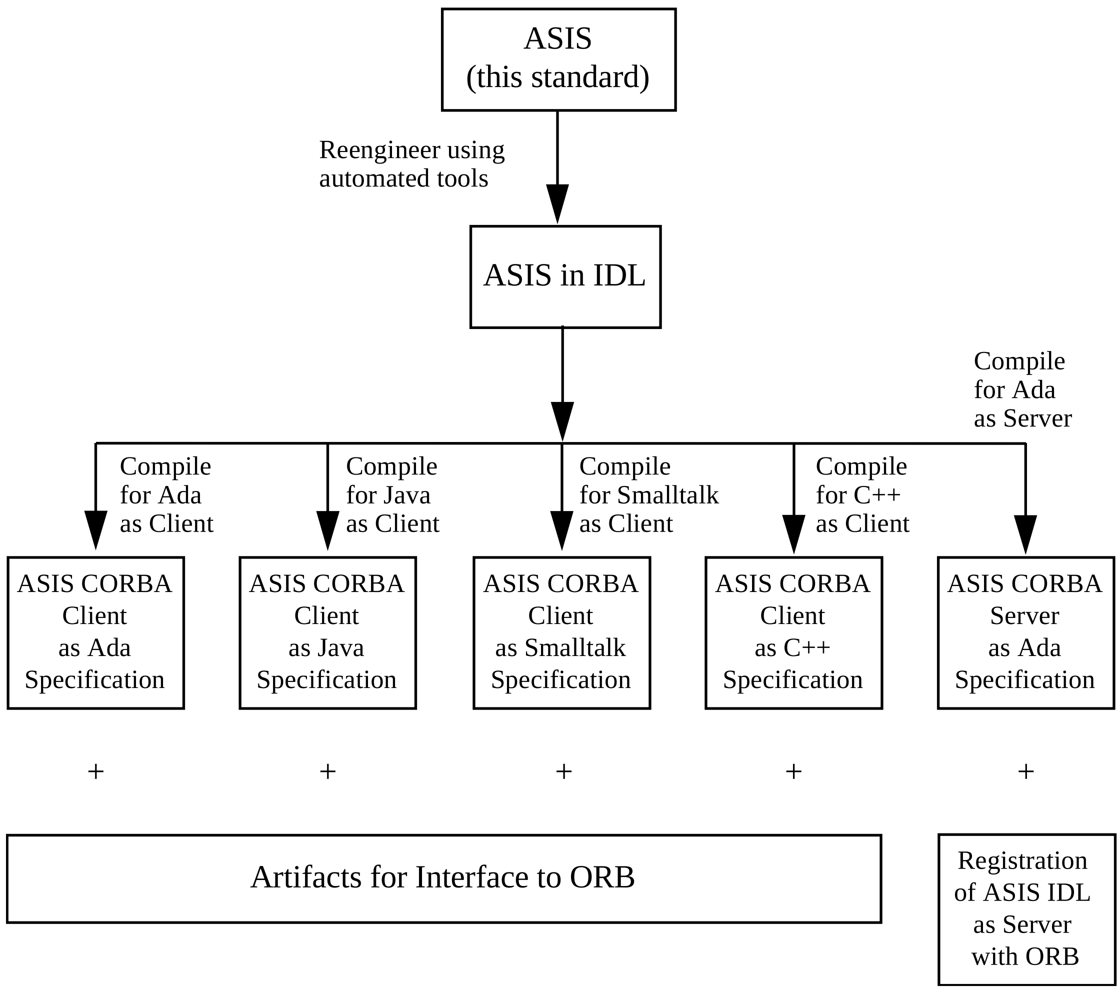

Figure C.1 — Generation of client/server ASIS artifacts

C.3.1 ASIS API server

When compiled on the server side using the Ada option, an Ada specification is created with the necessary objects to register the ASIS IDL with the ORB. This specification should be nearly identical to the original ASIS specification. The ASIS specification created by the compilation of the ASIS IDL should map to the original ASIS specification supported by the implementor. In this way, an implementor's ASIS interface can be used locally on the host node or through the CORBA client / server mechanism as desired by the tool implementor. ASIS implementors should be able to easily take advantage of an IDL client / server capability with little additional work.

C.3.2 ASIS API client tool

The ASIS IDL can be compiled on the client side producing an Ada package specification, C++ code, or Smalltalk code. This allows tools in any language to access the ASIS interface. Assuming the Ada compilation option, a logical Ada package body executable is created having hooks to the ASIS server via the ORB. If the Ada compilation option is used, then tools developed using Ada could access this Ada specification as if the ASIS package body were locally available. Any client tool can now use the ASIS CORBA client as an Ada specification through an Ada with clause. Client tools in Java, in C++, or in Smalltalk can also be easily built.

C.3.3 Approach to implement the Traverse_Element generic

An IDL should first be created from the ASIS specification (this International Standard). Automatic IDL generators can create IDL from Ada. Language features such as generics are problematic for this automatic conversion as generics cannot be automatically translated into IDL. The package Asis.Iterator is the only required ASIS package containing a generic. The following steps should be performed to create an IDL to support the required ASIS interfaces:

a) Remove child package Asis.Iterator.

b) Remove the optional packages (Asis.Data_Decomposition and Asis.Data_Decomposition.Portable_Transfer).

c) Perform a global replace of "Asis." with "ASIS_CORBA."

d) Rename package Asis as ASIS_CORBA.

e) Submit package ASIS_CORBA to an automatic IDL generation process.

f) Insert the attached IDL segments for Interface Iterator (see Annex C3.2) and Interface Generics (See Annex C3.2) to restore the effects of Asis.Iterator into the automatically generated IDL. [Note: the Interface Element and enum Traverse_Control could be automatically generated; they are shown here as they are supporters of Interface Generics and Interface Iterator.]

g) After the IDL is compiled with Ada as the target language, the ASIS_CORBA package will appear as shown in the attached Ada code.

h) On the Client side, a client can use package Asis directly when the specification has been mapped into ASIS_CORBA. The type Asis.Element will map into the package ASIS_CORBA.Element (there will also be the package Asis.Elements mapping into the package ASIS_CORBA.Elements). The package Asis.Iterator will map into ASIS_CORBA.Iterator. The generics are placed into a separate IDL module called Interface Generics which will create the package Asis.Generics. This will allow clients to map My_Pre_Operation and My_Post_Operation procedures into the Pre_Operation and Post_Operation procedures of Asis.Generics. This mapping can be achieved by:

   - Implementing the ASIS_CORBA.Generics in a separate server and pass the reference to the Generics object from the ASIS client to the ASIS server.
   - Using a multithreaded CORBA implementation with an additional server thread in the client.
   - Implementing the client mapping to accept callbacks in the client (not portable).

C.3.4 IDL to implement the Traverse_Element generic

It is possible to automatically generate IDL from the ASIS specification. The following interfaces for Interface Generics and Interface Iterator should be manually inserted into the automatically generated IDL prior to compilation. Compilation of the resulting IDL will produce the necessary artifacts to register the IDL with the Object Request Broker (ORB) and the output Ada depicted in Annex C.3.5

```
module ASIS_CORBA
{
    // ...
    interface Element { };        // This is the private type Asis.Element


     enum Traverse_Control {      // This type is found in package Asis

          Continue,               // Continues the normal depth-first traversal.
          Abandon_Children,       // Prevents traversal of the current element’s
                                  //   children.
          Abandon_Siblings,       // Prevents traversal of the current element’s
                                  // children and remaining siblings.
          Terminate_Immediately}; // Does exactly that.


    interface Generics            // abstract
    {                             // Generic portion of Asis.Iterator.Traverse_Control

            void Pre_Operation
                (in    Element           This_Element,
                 inout Traverse_Control  What_Next);

            void Post_Operation
                (in    Element            This_Element,
                 inout Traverse_Control   What_Next);

                                  // no state info needed, children will add state

       };

    interface Iterator            // Abstract
    {                             // This is Asis.Iterator.Traverse_Element

            void Traverse_Element
                 (in    Element            Starting_From,
                  inout Traverse_Control   What_Next,
                  in    Traversal          Using);


      };

      // ...
};
```

C.3.5 Ada code output by the IDL compiler

The following code segment represents the Ada specification generated by an IDL compiler for the IDL code presented in Annex C.3.4 when targeted for an Ada output.

```
with CORBA.Object;
package ASIS_CORBA is
    -- ...
    type Traverse_Control is      -- This type is found in package Asis
        (Continue,                -- Continues the normal depth-first traversal.
          Abandon_Children,       -- Prevents traversal of the current element's
                                  -- children.
          Abandon_Siblings,       -- Prevents traversal of the current element's
                                  -- children and remaining siblings.
          Terminate_Immediately); -- Does exactly that.
    -- ...
    package Element is            -- This is the private type Asis.Element

        type Ref is new CORBA.Object.Ref with null record;
        -- Narrow/Widen functions
        function To_Ref (From : CORBA.Any) return Ref;
        function To_Ref (From : CORBA.Object.Ref'CLASS) return Ref;

    end Element;
    -- ...
end ASIS_CORBA;

with CORBA.Object; with ASIS_CORBA.Element;
package ASIS_CORBA.Generics is -- generic portion of Asis.Iterator.Traverse_Control

    type Ref is new CORBA.Object.Ref with null record;      -- abstract
    -- Narrow/Widen functions
    function To_Ref (From : CORBA.Any) return Ref;
    function To_Ref (From : CORBA.Object.Ref'CLASS) return Ref;

     procedure Pre_Operation
        (Self           : Ref;
         This_Element   : in     ASIS_CORBA.Element.Ref;
         What_Next      : in out ASIS_CORBA.Traverse_Control);

     procedure Post_Operation
         (Self          : Ref;
          This_Element  : in     ASIS_CORBA.Element.Ref;
          What_Next     : in out ASIS_CORBA.Traverse_Control);
          -- no state info needed, children will add state
end ASIS_CORBA.Generics

with CORBA.Object; with ASIS_CORBA.Element; with ASIS_CORBA.Generics;
package ASIS_CORBA.Iterator is

    type Ref is new CORBA.Object.Ref with null record;
    -- Narrow/Widen functions
    function To_Ref (From : CORBA.Any) return Ref;
    function To_Ref (From : CORBA.Object.Ref'CLASS) return Ref;

    procedure Traverse_Element
       (Self          : Ref;
        Starting_From : in     ASIS_CORBA.Element.Ref;
        What_Next     : in out ASIS_CORBA.Traverse_Control;
        Using         : in     ASIS_CORBA.Generics.Ref);
        -- Using contains state and implementations of My_Pre_Operation and
        -- My_Post_Operation.
end ASIS_CORBA.Iterator;
```

Annex D
(informative)
Rationale

D.1 Benefits of code analysis

Why is something like ASIS needed? ASIS provides a basis for implementing portable tools that are aimed at analyzing static properties in Ada source code. Such code analysis capability has in general been under leveraged in software organizations; but for the Ada language in particular, it can greatly enhance the development process. Code analysis automation can harness the excellent software engineering features of Ada to facilitate code comprehension, high reliability, and high quality of the software product. The following text presents some motivational background.

D.1.1 Definition

Code analysis is the inspection of software source code for the purpose of extracting information about the software. Such information can pertain to individual software elements (e.g., standards compliance, test coverage), the element attributes (e.g., quality, correctness, size, metrics), and element relationships (e.g., complexity, dependencies, data usage, call trees); thus, it can support documentation generation, code review, maintainability assessment, reverse engineering, and other software development activities.

Extracted information falls into two major categories: *descriptive* reports which present some view of the software without judgment (e.g., dependency trees, call trees), and *proscriptive* reports which look for particular deficiencies — or their absence (e.g., stack overflow, excessive complexity, unintended recursion).

D.1.2 Applicability

Broadly speaking, the application of automated code analysis in the software development process promises, among other things, to:

- Promote ***discipline and consistency*** during development, increasing productivity and reducing unintended variation.
- Provide empirical evidence and ***metrics*** for process monitoring and improvement.
- Supplement code ***inspection and review***, diversifying beyond the limitations of testing or manual checking.
- Preserve ***architectural integrity*** in the software as compromises are made during development.
- Avoid violations of ***coding standards***, such as the use of inefficient language constructs.
- Increase the ***correctness and quality*** of delivered software, reducing defects via comprehensive assessment.
- Enhance ***safety and security*** by applying formal methods to verify assertions in program code.
- Expedite ***program comprehension*** during maintenance, for engineers new to the code.
- Support ***reengineering and reuse*** of legacy code, reducing costs.
- Result in ***reduced risk*** to budget and schedule.

The software development life-cycle phases where code analysis can be beneficially applied include all those in which source code exists: preliminary design (software architecture and interface definition), through testing and system integration, to maintenance and reengineering.

Hence, automated code analysis is a technology that primarily supports the back end of the software life cycle.

D.1.3 Motivation

Over the years, a wide-ranging set of commercial code analysis tools has become increasingly available [11]; examples of such tools include:

- Data flow analysis and usage metrics
- Invocation (call) trees and cross-reference
- Dependency trees and impact analysis
- Timing and sizing estimation
- Test-case generation and coverage analysis
- Usage counts of language constructs
- Quality assessment metrics
- Coding style and standards compliance
- Safety and security verification
- Code browsing and navigation
- Documentation generation
- Reverse engineering and re-engineering
- Language translation and code restructuring

Unfortunately, the current state-of-the-practice in software development either omits code analysis support altogether or only incorporates it as an ancillary, undisciplined, *ad hoc* resource. For example, it is not uncommon to find within a given project various home-grown tools that support the above goals but which are not recognized as overtly participating in the development process. Such tools can be quite obtuse (very indirect extraction of information) and are typically incomplete (handling only a subset of the development language). Further, they tend to be project-specific (or even person-specific), and cannot be reused in another project: they are later redeveloped from scratch.

These observations corroborate that the need for code analysis is genuine, and that a common set of uniform tools could provide significant benefits to projects. But in the case of Ada software, commercial code analysis tools have historically proven to be barely adequate, manifesting a variety of problems whose nature and origin are described below.

D.2 Technology for code analysis

For Ada, why is ASIS the best approach? Code analysis tools are not new, having been available for decades; but the advent of the Ada language has exposed a variety of analysis limitations and has consequently demanded more comprehensive technology. The following text articulates various Ada-specific issues from a historical perspective: it reviews several technologies that have historically been applied — with varying degrees of success — to code analysis specifically targeted to Ada software. The review is not comprehensive, but it sketches the evolution of issues that have propelled the development of the ASIS concept.

D.2.1 Code parsers

Historically, many commercial code analysis tools have been supplied by compiler vendors in conjunction with their compiler products. But as the community of CASE tool vendors has grown, such tools are often available independent of any compiler. Tool developers have found that conventional parser technology is sufficient for most traditional languages; thus when Ada came along, most vendors expected it would suffice to simply adapt their parsers to handle Ada syntax. But for Ada, the result has held many disappointments:

- Textual code editors are often sensitive to Ada syntax but not to Ada semantics.
- Graphical design editors yield valid graphics, but invalid Ada designs.
- Source-level tools such as debuggers are forced to understand and traverse the internal data structures of program libraries rather than the text of original source files.
- Reverse engineering and test tools manifest difficulties when trying to resolve overloaded subprogram names or renamings.
- Except for compilers, Ada tools do not require Ada Compiler Validation Capability (ACVC) certification; hence, such tools typically fail to handle the complete repertoire of Ada language features.

Consider the case of a toolsmith who wants to develop a call-tree analyzer. For such a tool to accurately process Ada source files, the toolsmith would be forced to build almost the entire front end of an Ada compiler — a decidedly major undertaking that far out-scopes the original tool building effort. But CASE tool vendors are not in the compiler business; most are reluctant to make this major investment, or have tried and failed. Yet tools built on parser technology alone are not able to fully support the semantic richness of Ada.

D.2.2 DIANA

Many Ada compilers store program units into libraries. They typically structure the information according to some proprietary internal form, such as trees of DIANA (Descriptive Intermediate Attributed Notation for Ada — note that the following discussion applies to all internal forms, but that DIANA is singled out due to its public documentation [5, 6]: DIANA had been intended for standardization, but failed due to the unexpectedly wide variation in internal forms). Such trees thus encode both syntactic and semantic information about Ada programs. The root of a DIANA tree corresponds to a compilation unit; the nodes correspond to smaller Ada structures, such as declarations and statements. Node attributes contain descriptive information and references to other nodes.

DIANA trees offer great convenience and power to toolsmiths, and are sufficient to support the implementation of a large variety of tools (including code generators in compilers). For example, with access to DIANA, the toolsmith who wanted to develop the call-tree analyzer would have a fairly straightforward project. Furthermore, the tool would exhibit better performance, bypassing the needless regeneration (and redundant storage) of intermediate compilation results that are already available in the Ada libraries.

The power of DIANA is sufficient to support the implementation of a virtually unlimited variety of tools. In general, any tool that requires the semantic information generated by an Ada compiler can benefit from access to DIANA. But as with any technology, the use of DIANA also has drawbacks:

- A given implementation of DIANA by a vendor is ***subject to change***: upgrades can obsolete tools written against previous versions, hampering maintenance.
- Similarly, DIANA ***implementations vary*** from vendor to vendor: porting a tool across platforms is a risky endeavor.
- DIANA is ***hard to use***: the trees are quite complex, making it difficult to write and debug tools written against a DIANA specification.
- The lack of a simple mapping to Ada makes DIANA ***hard to understand***: as an abstracted representation of an Ada program, it does not map intuitively to concrete Ada structures.

- DIANA is ***not extensible***; but tools may need to add attributes for storing graphical or other tool-specific information.

D.2.3 LRM-interface

Thus a growing need arose to make tool development possible at the Ada level rather than at the internal representation level. It was these issues that drove some Ada compiler vendors to independently develop proprietary higher-level interfaces to encapsulate their Ada program libraries.

In particular, to overcome the drawbacks of DIANA while retaining all of its advantages, Rational Software Corporation developed their LRM-Interface product [8] in the late 1980's. It provided nearly the same power as DIANA, through services that extracted a variety of information from the internal DIANA trees. The LRM-Interface was also considerably easier to understand than DIANA, because it used the already-familiar terminology defined in the Ada LRM (the original *Reference Manual for the Ada Programming Language* [12], or its more recent version, the *Ada 95 Reference Manual* [7]). Furthermore, the LRM-Interface was not subject to change (or at least much less so than was the underlying DIANA), so tools written against it were easily migrated to updated implementations.

Regardless of LRM-Interface specifics, this and other similar approaches generally provide great flexibility: for example, an *ad hoc* tool can be easily and quickly built by in-house engineers, without funding the development or specialization of a commercial tool. But as expected, this approach also has shortcomings:

- While DIANA as a data structure is not extensible, the above interfaces can be extended in the sense of user-supplied secondary functions built on the functions already provided; even so, all the functions are ***read-only*** and cannot modify any state.
- Importing the subject source code into the tool environment can require edits that necessarily result in ***code distortion***, such that original code attributes might not be preserved (e.g., line numbers or the byte sizing of data).
- Tools are ***vendor dependent***, such that a given tool cannot access Ada libraries from multiple vendors, or equivalent tools from multiple vendors cannot access a given Ada library.
- Data interchange is ***not standardized*** among tools, so users can't configure their own integrated toolsets by choosing from competing or complementary vendors.
- Within a software engineering environment (SEE), Ada semantic information remains isolated from, and ***not integrated*** with, other engineering data present in the environment.

D.2.4 ASIS

Historically, only a few Ada vendors provided access to the information contained in their proprietary Ada program libraries, and each such interface was unique. Thus began to emerge the need for an open standard that would allow uniform, vendor-independent access to that information.

Leveraging some informal efforts, the STARS program initiated the development of the Ada Semantic Interface Specification (ASIS) in 1990; but shortly thereafter, the activity became unfunded due to the STARS decision to no longer support standardization efforts. Despite this, several of the involved vendors (primarily TeleSoft) continued the ASIS work on a volunteer basis. Some time later, Rational also became an active participant and seeded the draft standard by contributing their LRM-Interface specification to ASIS.

In 1992, the Ada Board recognized the potential benefits to the Ada community of an ASIS standard, and recommended that the Ada Joint Program Office (AJPO) director support "by whatever means possible the development of an ASIS standard and its submission to ISO/WG9 for publication." The Association for Computing Machinery (ACM) Special Interest Group on Ada (SIGAda) took on this important work though volunteer effort in the ASIS Working Group (ASISWG) [2]. The ASISWG developed the interface to ISO/IEC 8652:1987. In December 1993, ASIS was viable and the director of the AJPO recommended this interface be used by tools needing information from the Ada program library. The ASISWG then became focused towards developing ASIS for ISO/IEC 8652:1995. As the ASISWG has no standardization authority, an ASIS Rapporteur Group (ASISRG) was established on 28 April 1995 by the ISO/IEC JTC1/SC22 WG9 to standardize ASIS as an International Standard for Ada. ASISWG and ASISRG jointly cooperated to evolve ASIS as an important interface to the Ada compilation environment.

Like its LRM-Interface predecessor, ASIS defines a set of queries and services that provide uniform programmatic access to the syntactic and semantic information contained within Ada library units (i.e., vendor independence). In addition, for each Ada vendor, ASIS clients are shielded from the evolving proprietary details that implement the vendor's library representations and internal forms (i.e., version independence). ASIS is designed for implementation on a variety of machines and operating systems, while also supporting the Ada semantic requirements of a wide range of client tools.

ASIS services are essentially primitive, intended to support higher level, more sophisticated services that address the varied needs of specialized tools. While ASIS currently operates in a read-only mode, it could eventually be extended to support some (probably limited) update capability, enabling client tools to save application-dependent data (e.g., graphical information) within an existing Ada library. Although an ASIS implementor could readily support read-write features, members of the safety-critical community have emphasized the danger of providing a generalized write capability, since this could enable editing of the internal representation to differ from the original source code.

The long-term key is to achieve a critical mass of ASIS implementations. This will promote a new generation of semantically integrated Ada tools, which in turn will increase programmer productivity and decrease Ada development costs. In summary, the availability of ASIS implementations promises to:

- Stimulate ***improved quality*** within existing Ada CASE tools; currently, these tend to be weak in supporting full Ada semantics (e.g., in preserving renamed entities, resolving overloaded subprogram names, etc.).
- Enhance ***safety and security*** by providing for a new class of powerful analysis tools that apply formal methods to verify assertions in program code (e.g., using Pragma Annotate).
- Eliminate the need to import Ada source code into secondary Ada compilation environments, resulting in ***no distortion*** or loss in the subject code (e.g., preserving original line numbers and the byte sizing of data).
- Maximize ***interoperability*** between Ada CASE tools and Ada compilation environments, thus maximizing tool availability.
- Enable the ***data interchange*** of Ada semantic information between complementary or competing Ada CASE tools, thus maximizing user choices for the best capabilities of each.
- Improve the overall ***performance*** of Ada CASE tools, by eliminating the regeneration (and redundant storage) of Ada semantic information that already exists in Ada libraries.
- Facilitate in-house development of informal but powerful ad hoc Ada tools, providing ***flexibility*** as needed without funding Ada CASE vendor specializations.
- Promote standardization in software engineering environments (SEEs), enabling ***data integration*** of Ada semantic information with other engineering data present in the environment.

- Establish ***enabling technology*** for new Ada CASE tools, by eliminating the need for tool vendor investment in proprietary Ada compiler technology; this will have a major impact on stimulating the development of new code analysis capabilities.

D.2.5 Benefits of ASIS standard

As an International Standard, ASIS benefits the Information Technology community by facilitating the development of powerful CASE tools portable amongst the various environments provided by Ada implementors. This portability can only be achieved through the standardization of ASIS at the international level. A standardized ASIS promotes the development of powerful tools for the software engineering environment by providing access to important semantic information otherwise available only through proprietary interfaces. Further, ASIS benefits the Information Technology community as a valuable resource for application development (e.g., decoupling system to system interfaces). The international standardization of ASIS facilitates the use of this important capability in the development of system software applications.

D.3 Design considerations for ASIS

Why has ASIS taken the form that it has? To guide the assessment of design decisions for ASIS, the ASIS Working Group has defined a number of design goals. Most of these embody sound software engineering principles, while others represent a consensus approach regarding controversial issues. The following text discusses some of these.

D.3.1 Design goals

ASIS for ISO 8652:1987, known as ASIS version 1.1.1 [1], was intended to be used as a *de facto* standard. For ISO/IEC 8652:1995, it was necessary to revise ASIS to take into account the changes in the Ada language definition. When starting the revision of ASIS, the ASIS Working Group agreed upon the following overall design goals: the ASIS design should

- Minimize the cost of converting existing ASIS-based applications to the new Ada standard.
- Be simple and uniform with minimal concepts and minimal operations.
- Allow for standardization at the international level.

The identification of design goals for ASIS was the major accomplishment of the 19-22 April 1994 ASISWG meeting in Boulder, Colorado. These goals were updated at the 27-28 June 1994 ASISWG meeting in Reston, Virginia.

The ASIS design goals have been subdivided into the categories shown below. At a lower level are several design rules identified as important for the interface specification. Also discussed were several potential goals which were felt should not be part of the ASIS interface; these are very useful in providing scope. Several administrative goals were also included.

Design goals (functionality):

- ASIS should provide all semantic information described by the Ada Reference Manual.
- ASIS should not invent new functionality.
- ASIS should provide a read-only interface to the Ada compilation environment.
- ASIS should be able to support Ada semantics for a wide range of tools and application requirements.
- Each ASIS interface should be fully defined and unambiguous.
- ASIS should be easy to learn and easy to use.
- ASIS should use the Ada Reference Manual terminology and style.

- ASIS should have a callable Ada interface.
- ASIS should provide the ability to traverse the full syntactic structure represented in the Ada compilation environment for the original text to the point where the syntax has semantic relevance.
- ASIS should not be misleading, be complete as possible, and provide all information queried (or make obvious to users that they are not getting all the information, and provide an interface to get that additional information).

Design goals (open systems):
- ASIS should be implementor independent.
- ASIS should be non-proprietary.
- ASIS should be designed in such a way that it is implementable by all implementors.
- ASIS should produce semantically consistent results regardless of the implementor.
- ASIS should increase portability of tools across Ada compilers.
- ASIS should make Ada compilation environments Open Systems and Reusable Components for use by other tools.
- ASIS should simplify semantic information retrieval by hiding implementation dependent aspects of Ada environments.

Design goals (migration of ASIS for ISO 8652:1987 to ASIS for ISO/IEC 8652:1995):
- ASIS should ensure that ASIS-based tools are convertible to new ASIS.
- ASIS should minimize the cost of converting ASIS tools.

Design goals (miscellaneous):
- ASIS should be efficient.
- ASIS should be well engineered.
- ASIS documentation should be in ISO format.
- ASIS should allow for standardization at the international level.
- ASIS should be written to not preclude future extensions.
- ASIS should be designed to be simple and uniform with minimal concepts and minimal operations.

Design rules:
- ASIS should be as strongly typed as possible.
- ASIS should declare its own basic types, and rename (or hide) all implementor types and operations, for tool portability.
- ASIS should use exceptions to report failures and errors; ASIS should supplement exceptions with status functions.
- ASIS should allow only ASIS exceptions to escape the interface.
- ASIS should supplement weak compile time type checking of element values by run-time use of ASIS_Inappropriate_Element Exception.
- ASIS should have an existence query for all optional syntax.
- ASIS should be able to report a span for every explicit element.

NOT design goals:

- It is not a goal to support tools having dynamic run-time requirements (e.g., symbolic debugger).
- It is not a goal to define the semantics of the interface in the face of simultaneous library updates.

Administrative goals:

- Resolve issues by consensus rather than simple majority.
- Keep the effort scaled down so it is possible and achievable.

D.3.2 Major changes from ASIS for ISO 8652:1987

The main changes in the ASIS definition are as follows:

- New queries were added for the new language features, such as child units and protected types, and old ones were revised to take into account language changes.
- Some queries of ASIS were aggregated, in order to minimize their total number.
- Whereas ASIS for ISO 8652:1987 was defined as an interface to a program library, ASIS is now an interface to an Ada compilation environment, following the new ideas of Ada about the program structure and the compilation process.
- The set of package specifications defining ASIS was reorganized into a hierarchy of child units.
- Terminology was revised and comments in the ASIS package specifications were reworded to make them consistent with Ada.

D.3.3 Essence of Ada and ASIS

There are some fundamental differences between a language definition (i.e., the definition of Ada) and the interface definition of a tool, such as ASIS. Clearly the Ada language definition, as described in the Ada Reference Manual, and ASIS are different in nature and have different aims.

Ada defines the notion of a legal compilation unit, the notion of a legal complete partition and the dynamic semantics of a complete partition. It also defines some aspects of the process of compiling a compilation unit and some properties of the results of this process.

On the other hand, ASIS is more like the requirements definition for a tool which provides information about a set of Ada software components, which may or may not make up a complete partition. Moreover, as a tool, ASIS provides management facilities which go beyond what is needed in a language definition (e.g., ASIS needs terminology for designating the then-part and the else-part of an if statement).

Despite these differences in nature, the terminology used in Ada and ASIS should be the same whenever possible. Clearly, there should be no contradictions. Closeness in terminology eases understanding of ASIS by those who already know Ada, saves documentation effort, and is justified by improved maintenance, since every correction in Ada can impact ASIS.

Nevertheless, ASIS needs some new terms. Here are some cases:

- To cut down the number of queries, ASIS sometimes needs to aggregate Ada-defined syntactic categories.
- To walk through a heterogeneous data structure, such as a list of compilation units and configuration pragmas, ASIS needs to introduce a unifying concept for item types in the structure.

- Finally, some concepts in Ada are vaguely defined compared to the needs of ASIS. It can then be better to introduce a new name rather than restrict the meaning of the Ada concept. The Ada compilation environment and ASIS context concepts can serve as an example here.

D.4 Major issues regarding ASIS

How has the Ada language revision affected ASIS? The following text presents a discussion on a variety of issues that have arisen in making this transition.

D.4.1 Ada environment and compilation units

One of the major points in the revision of ASIS was the replacement of the idea of an interface to an Ada program library in ASIS 1.1.1 by an interface to an Ada compilation environment. This change has in fact some farther-reaching consequences than it seems at a first look.

The Ada Reference Manual 10.1.4(1) says: "Each compilation unit submitted to the compiler is compiled in the context of an environment declarative_part (or simply, an environment), which is a conceptual declarative_part that forms the outermost declarative region of the context of any compilation. At run time, an environment forms the declarative_part of the body of the environment task of a partition". And the next paragraph says: "The declarative_items of the environment are library_items appearing in an order such that there are no forward semantic dependencies." Reference Manual 10.1.4(5) then says: "When a compilation unit is compiled, all compilation units upon which it depends semantically shall already exist in the environment;..."

At a first look, there seems to be a contradiction between Reference Manual 10.1.4(1,2) which defines an environment as a declarative_part and says that an environment contains library_items, but not compilation_units, and Reference Manual 10.1.4(5) which speaks about the existence of compilation units in the environment.

But this apparent "contradiction" is resolved in Reference Manual 10.1.1(9), which says: "The term compilation unit is used to refer to a compilation_unit. When the meaning is clear from context, the term is also used to refer to the library_item of a compilation_unit or to the proper_body of a subunit (that is, the compilation_unit without the context_clause and without the separate (parent_unit_name))."

This means the Ada Reference Manual defines compilation_unit as a syntactical category, and at the same time it defines "compilation unit" (without underscore and typed in regular font) as another technical term, which can denote either one of the syntactical categories "compilation_unit", "library_item" or "proper_body". ASIS is aware of this subtle difference and consistently use the term "compilation unit". In ASIS "Compilation_Unit" is the name of a type. Objects of this type are always compilation_units (except if they are nonexistent, unknown, or null).

The purpose of the Ada concept of an environment is threefold. It allows Ada to define what is a legal compilation unit, what is a legal partition and what is the behavior of a partition at run-time. Strictly speaking, an Ada environment contains only library_items; indeed, context clauses do not make sense inside of a declarative part, and subunits are enclosed in their parent unit in place of the corresponding stub.

D.4.2 ASIS context and inconsistency

ASIS has to deal with sets of compilation units. To avoid any confusion, it therefore introduces a new term, the concept of a context, which stands for a set of compilation_units maintained by an Ada compilation system. A context has a state, consisting of the compilation units belonging to it, and the configuration pragmas applying to them, but also depending on the history of compiler runs, or other actions taken by the compilation system. Thus, the ASIS concept of a context resembles closely an Ada environment when used as a dynamic concept like in the Ada Reference Manual 10.1.4(3), which says: "The mechanisms for creating an environment and for adding and replacing compilation units within an environment are implementation-defined."

Ada does not deal with arbitrary sets of compilation units as a whole, but ASIS does.

EXAMPLE Suppose we first compile a package P, then a procedure A withing P, then we modify P, recompile it, and finally compile the procedure B which also is withing P; we also suppose that all compiler runs are successful (i.e., do not detect compilation errors):

```
compile
     package P is ...
compile
     with P;
     procedure A ...

edit
     package P and change it to P'

compile  -- reference to P'
     package P is ...
compile
  with P;  -- reference to P'
  procedure B ...
```

As far as the legality of compilation units is the concern, the environments can be distinguished as: P alone, P together with A, P' alone, and P' together with B. It could well be that compiling A with P' yields a compilation error. Nevertheless, all these environments contain only consistent unit sets following Reference Manual 10.1.4 (5), which says: "When a compilation unit is compiled, all compilation units upon which it depends semantically shall already exist in the environment; the set of these compilation units shall be consistent in the sense that the new compilation unit shall not semantically depend (directly or indirectly) on two different versions of the same compilation unit, nor on an earlier version of itself."

Only when the results of these compilations are used to build a partition, the inconsistency of depending on two versions for P can be detected by the implementation following Reference Manual 10.2 (27), which says: "The implementation shall ensure that all compilation units included in a partition are consistent with one another, and are legal according to the rules of the language."

As a conclusion, we can say that Ada does not deal with the above sequence of compiler runs as a whole, and especially does not bother about possible "inconsistencies".

On the contrary, it is clearly desirable for ASIS to deal with the previously described situation as a whole, so that tools can be built capable of analyzing dependencies between compilation units and possible inconsistencies in the state of the "context". It is the idea that an ASIS context can refer to such a situation in some implementation-dependent way. As defined now, in no case ASIS will provide access to both versions of P, even indirectly by references from A and B. Depending on the ASIS implementation, the compilation unit A may or may not be part of a context. If it is, then ASIS reports that A is obsolete (e.g., by saying that A is inconsistent with P). If issued, queries about A can raise exceptions or produce wrong results. This does not preclude some ASIS implementation providing correct answers to structural queries about A (which are about syntax only).

As we have just seen, ASIS introduces the idea of inconsistencies within a context. An inconsistency is always expressed in terms of pairs of compilation units, the second depending on the first: (A depends on P). An inconsistency is reported if the latest compiler runs for A and P used different versions of P, or different versions of any other unit upon which P depends.

This is perhaps the place to speak about the concept of a version of a compilation unit. Even though Ada uses the term in Reference Manual 10.1.4 (5), it does not define it. On the contrary, ASIS does not use it at all. ASIS provides queries for retrieving a library unit having a given name; it is stated that at most one unit can be retrieved, which means that library units, but also compilation units, are uniquely identified by their fully expanded names, and that the concept of a version does not make sense in ASIS.

The Ada environment and ASIS context concepts, though similar, are nevertheless different, and the same holds for the concept of consistency. However, and even if technically perhaps not entirely correct, the idea of ASIS being an interface to an Ada compilation environment reflects intuitively quite well the very nature of ASIS.

D.4.3 Implicit declarations

Sometimes a concept is missing as such in ASIS, but it can be rebuilt indirectly (e.g., implicit declarations of a declarative region can only be retrieved indirectly). Ada distinguishes between explicit and implicit declarations (Reference Manual 3.1 (5)): "A declaration is a language construct that associates a name with (a view of) an entity. A declaration can appear explicitly in the program text (an explicit declaration), or can be supposed to occur at a given place in the text as consequence of the semantics of another construct (an implicit declaration)." In many ways, Ada deals in an uniform way with explicit and implicit declarations. We will see that this is not true for ASIS.

ASIS defines several queries returning lists of explicit declarations from different kinds of declarative regions (i.e., Body_Declarative_Items, Visible_Part_Declarative_Items, Private_Part_Declarative_Items, and Block_Declarative_Items).

ASIS has several approaches to deal with implicit declarations:

a) A first case is **type declarations**. To retrieve the implicit declarations associated with a type, ASIS provides queries to get all the predefined and inherited subprograms associated with it, and also its implicit components, when it is a record.

b) A second case is **statement_identifiers**. Labels, block names and loop names are statement_identifiers. Reference Manual 5.1(12) says that "For each statement_identifier, there is an implicit declaration (with the specified identifier) at the end of the declarative_part of the innermost block_statement or body that encloses the Statement_Identifier." ASIS does not provide a query listing the statement_identifiers implicitly declared at the end of a declarative_part. The only way to find out about statement_identifiers is to access the statements and then to retrieve the identifiers attached to them by means of the queries Label_Names and Statement_Identifier.

The two cases show that it is not easy for an ASIS-based application to retrieve all implicit declarations of a given declarative region.

Nevertheless, the approach taken by ASIS is justifiable because compilers usually do not store the implicit statement_identifier declarations in their intermediate representations, and it would therefore be a major burden for an ASIS implementation to yield them as a result of a query.

D.4.4 Abstract "=" for private types

An Is_Equal function has been defined for the private types Context (Clause 3.5), Element (Clause 3.6), Compilation_Unit (Clause 3.10), Container (Clause 9.1), Line (Clause 20.1), Id (Clause 21.1), Record_Component (Clause 22.1), and Array_Component (Clause 22.3). An abstract function "=" has been defined for each private type because the semantics for the Is_Equal function for these types may differ from the semantics of the default "=" operator. This prevents improper use of the "=" operator.

D.4.5 Usage names and expressions

To minimize the number of queries, ASIS sometimes unifies several concepts. For usage names, Ada makes a difference between names which are and which are not expressions. ASIS considers all usage names as expressions.

For instance, the name of the procedure in a procedure call statement is considered to be an expression. Only when trying to apply to such an "expression" a query specific to an expression, yielding for example the type of the expression, ASIS will say that it is not an expression by returning a Nil_Element.

This approach allows ASIS to have a smaller element kinds classification. ASIS for ISO 8652:1987 already worked this way, and compatibility is another reason for keeping the approach.

D.4.6 Select alternative

Let's start by recalling the Ada syntax of a select_alternative (Reference Manual 9.7.1 (4-7)):

```
select_alternative ::= accept_alternative
                     | delay_alternative
                     | terminate_alternative

accept_alternative ::= accept_statement [sequence_of_statements]

delay_alternative ::= delay_statement [sequence_of_statements]

terminate_alternative ::= terminate;
```

When analyzing a selective_accept, ASIS sees the select_alternative just as a sequence_of_statements, and does not distinguish between its three possible forms. The advantage of this approach shows up when the ASIS-application programmer needs to deal with statements in a uniform way, for instance in a recursive descent. The price to pay is that she has to do her own analysis when the distinction is needed, by examining the first statement in the sequence.

D.4.7 Attribute definition clauses

In Reference Manual 13.3 (2), Ada says that an attribute_definition_clause is defined as:

```
attribute_definition_clause ::=
             for local_name'attribute_designator use expression;
           | for local_name'attribute_designator use name;
```

In Reference Manual 4.1, Ada says that a name may be an attribute_reference, which then is formally defined in Reference Manual 4.1.4 (2) by:

```
attribute_reference ::= prefix'attribute_designator
```

ASIS uses this latter definition to simplify the attribute_definition_clause definition:

```
attribute_definition_clause ::=
             for attribute_reference use expression;
           | for attribute_reference use name;
```

From an attribute_definition_clause, ASIS therefore retrieves an attribute_reference, which can be further decomposed into the prefix and the attribute_designator parts.

D.4.8 Configuration pragmas

According to Reference Manual 10.1.5 (8), the purpose of configuration pragmas is: "They are generally used to select a partition-wide or system-wide option." The same paragraph says about their placement that "[T]hey shall appear before the first compilation_unit of a compilation.", and later on about their "scope" that "The pragma applies to all compilation_units appearing in the compilation, unless there are none, in which case it applies to all future compilation_units compiled into the same environment".

ASIS does not use the notion of a compilation ordering. The first reason for this is to keep ASIS as small as possible. The second justification is that compilers usually do not keep track of compilations, but only of their effects.

However, some ASIS-based applications could need information about configuration pragmas. ASIS therefore introduces a special kind of compilation unit, called A_Configuration_Compilation, containing only configuration pragmas. By retrieving the content of A_Configuration_Compilation, it is possible to know the configuration pragmas which are in effect for the next compiler run within this context.

It is important to notice here that ASIS provides also a query to retrieve all the pragmas which were in effect when a given compilation unit was compiled, including both configuration pragmas and pragmas specific to the unit; clearly, some of the former could differ from those found currently in A_Configuration_Compilation.

D.4.9 Queries with additional context parameter

Several queries, including Corresponding_Declaration and Corresponding_Body, are overloaded with an additional parameter, Context. The additional queries are there for flexibility and power, if an ASIS implementation can take advantage of it.

Some ASIS implementations may be able to support multiple open Contexts at the same time. Tools that make use of more than one Context could be severely limited if they do not have the control provided by the Context parameter. An example of such a tool is one that compared two contexts, identifying units appearing in both and ensuring their semantic dependent units were consistent, perhaps for some configuration management purpose.

The Corresponding_Declaration in a different context should always be the declaration upon which the body depends semantically in that context, not just an unrelated declaration of the same name.

ASIS implementations that only allow one open Context can implement these pairs of queries in tandem, with one simply calling the other and providing the current Context as a parameter.

D.4.10 Ids

Ids provide a mechanism to refer to the same ASIS element from one tool analysis to a separate tool analysis. The type Id is a way to identify a particular element (i.e., a unique reference to an Element) which is efficient and persistent as long as the environment is not recompiled Ids allow a tool to create a database of visited elements to support a variety of tool requirements. An Id can be passed from one ASIS analysis tool to another, facilitating the sharing of analysis thus eliminating the requirement to recompute the analysis for the same element. It could also be used to generate hyper-linked analysis reports and support the pipelining of analysis amongst integrated tools. Ids can be written to files. Ids can be read from files and converted into an Element value with the use of a suitable open Context

D.4.11 Data decomposition

The optional Data_Decomposition package was developed to support the special needs of applications in decoupling system software interfaces. Such interfaces could support communications between two platforms or communications between two subsystems within a system. Typically an interface between two software systems is accomplished through a rigorously defined interface. This rigorous interface is beneficial since it defines the expected communications protocol between the two computers. However, this interface is a serious impediment to upgrading a system when only part of the system is to be upgraded. The new system typically requires a new interface to support the new capabilities. Hence software on both sides of the interface needs to be upgraded to enhance the system.

Data_Decomposition allows the upgrade of part of a critical system without changing software in the rest of the system. Hence it provides significant flexibility for the life-cycle of a system. Decoupling the interface between two software systems can be accomplished when each system has access to semantic information in the other system's Ada compilation environment. This semantic information describes objects sent across the interface and could serve as a means to identify its operations. Data_Decomposition was specifically developed to support a one-way interface between a United States Air Force satellite and the ground station's message analysis (delog) processing.

The initial delogger implementation required a rigorous interface protocol and specialized processing for each possible message. This was expensive from a development perspective. The time required to build the "types dictionary" took 704 hours for each build, of which there were many.

Their practical solution was to develop a generalized processor, capable of interpreting any message type sent across the interface. This required knowledge of the semantics of data in the message. This *Universal Delogger* was built using ASIS to analyze messages sent for data analysis [13]. The process was:

a) Implement both the satellite software and the ground software in Ada;
b) Send each message from the satellite to the ground station as a byte-stream of application data with a header: the data based on an Ada record type definition of heterogeneous composite types; and the header an identifier for this Ada record definition.
c) Use the ASIS interface to obtain the Ada record type information associated with the message header.
d) Use the ASIS interface to map (decompose) the byte-stream into the appropriate Ada record type.
e) Decode the message for analysts and analytical processing based on the composite types in the Ada record. The named components with their values are then presented to an operator.

Using this approach, the ground station can perform generalized processing for any message from any satellite providing it includes an ASIS interface to the Ada compilation environment that developed the satellite software. Developmental cost benefit: Many, but most notable: each major build had to rebuild the message dictionary; the time required to build this "types dictionary" was reduced from 704 hours to 2 hours for each rebuild. Also notable is the ground station can process messages from newer satellite systems without any changes in the software on the ground station, even though message content is changed.

D.5 Conclusion

The Ada language definition and ASIS have different aims. When taking this into account, it can be said that ASIS follows, overall, as much as possible the syntax structure of Ada; that it uses Ada-defined terms consistently; and it does not introduce unnecessarily new terms and concepts. Clearly, ASIS has a need for unifying concepts (e.g., for traversing heterogeneous lists), and it introduces them quite carefully and advisedly. Considering that ASIS users and ASIS implementors are both interested in achieving stability of the ASIS definition as soon as possible, and that ASIS has already proved its effectiveness for developing useful tools, the ASIS Working Group and the ASIS Rapporteur Group believe the ASIS definition is now ready for standardization.

D.6 Acronyms

ACM	Association for Computing Machinery
ACVC	Ada Compiler Validation Capability
AJPO	Ada Joint Program Office
ANSI	American National Standards Institute
API	Application Program Interface
APSE	Ada Programming Support Environment
ASCII	American National Standard Code for Information Interchange
ASIS	Ada Semantic Interface Specification
ASISRG	ASIS Rapporteur Group
ASISWG	ASIS Working Group
CASE	Computer-Aided Software Engineering
CORBA	Common Object Request Broker Architecture (OMG)
COTS	Commercial, Off-The-Shelf
CPU	Central Processing Unit
DIANA	Descriptive Intermediate Attributed Notation for Ada
DII	Dynamic Interface Invocation
IDL	Interface Definition Language
IEC	International Electrotechnical Commission
ISO	International Standards Organization
LRM	Language Reference Manual (Ada)
OMG	Object Management Group
ORB	Object Request Broker
SEE	Software Engineering Environment
SIGAda	Special Interest Group on Ada
STARS	Software Technology for Adaptable, Reliable Systems

Bibliography

1. ASISWG/SIGAda; May 1994: Ada Semantic Interface Specification, version 1.1.1, ASIS for ISO 8652:1987.
2. ASISWG/ASISRG World Wide Web site at => *http://www.acm.org/sigada/WG/asiswg*
3. Colket, C. Barnes, G. Blake, S. Cooper, D. Jørgensen, J. Roby, C. Rittersdorf, D. Rybin, S. Strohmeier, A. Thomas, B.; January/February 1997: "Architecture of ASIS, A Tool to Support Code Analysis of Complex Systems," SIGAda ACM Ada Letters, volume XVII #1, pages 35-40.
4. Ehrenfried, D.; July/August 1994: "Static Analysis of Ada Programs," SIGAda ACM Ada Letters, volume XIV #4, pages 28-35.
5. Goos, W. Wulf, A. Evans, K. Butler; 1983: DIANA: An Intermediate Language for Ada, Springer-Verlag LNCS #161.
6. Intermetrics; 1986: *Reference Manual for the Descriptive Intermediate Annotated Notation for Ada (DIANA)*.
7. *ISO/IEC 8652:1995, Information technology — Programming languages — Ada*.
8. Rational; May 1989: *Rational Design and Document Support Tools Guide*, document 8043A, version 1.0.
9. Rybin, S. Strohmeier, A. Kuchumov, V. Fofanov, V.; 1996: "ASIS for GNAT: From the Prototype to the Full Implementation," *Proceedings of Ada-Europe Conference on Reliable Software Technologies*, Springer-Verlag LNCS #1088, pages 298-311.
10. Software Productivity Consortium; 1989: *Ada Quality and Style: Guidelines for Professional Programmers*, Van Nostrand Reinhold.
11. Software Productivity Consortium; June 1990: "Tools for Static Analysis of Ada Source Code," technical report 90015-N.
12. United States Department of Defense, Ada Joint Program Office; February 83: *Reference Manual for the Ada Programming Language*, ANSI/MIL-STD-1815A, ISO-8652:1987.
13. White, M.; April 1993: “ASIS for Analysis of Recorded Data,” Software Technology Conference (STC'93).

Index

Both ASIS interfaces and Ada syntactic categories are included in this index. In general, ASIS interfaces are denoted with the first letter of each name in uppercase and Ada syntactic categories are denoted in lower case.

A

A_Base_Attribute, 41
A_Bit_Order_Attribute, 41
A_Block_Statement, 14, 42, 104, 105, 123, 183, 184, 185, 186
A_Body_Stub, 34
A_Body_Version_Attribute, 41
A_Box_Default, 36
A_Callable_Attribute, 41
A_Caller_Attribute, 41
A_Case_Path, 43, 183
A_Case_Statement, 42, 182, 183
A_Ceiling_Attribute, 41
A_Character_Literal, 39, 139, 160, 161, 162, 164, 166
A_Choice_Parameter_Specification, 34, 135, 194
A_Class_Attribute, 41
A_Clause, 13, 20, 31, 32, 92, 94, 95, 96, 100, 121, 128, 129, 132, 136, 154, 157, 158, 185, 197, 199
A_Code_Statement, 42, 195
A_Complex_Dynamic_Model, 218, 221, 225
A_Component_Clause, 44, 198, 199
A_Component_Declaration, 33, 35, 114, 115, 116, 154, 155, 224, 225
A_Component_Definition, 36, 95, 114, 148, 153
A_Component_Size_Attribute, 41
A_Compose_Attribute, 41
A_Concatenate_Operator, 40
A_Conditional_Entry_Call_Statement, 42, 182
A_Configuration_Compilation, 46, 66, 67, 71, 72, 78, 91
A_Constant_Declaration, 33, 95, 114, 115, 116
A_Constrained_Array_Definition, 37, 114, 147, 215, 223
A_Constrained_Attribute, 41
A_Constraint, 36, 98, 146, 150
A_Controlled_Pragma, 32
A_Convention_Pragma, 32
A_Copy_Sign_Attribute, 41
A_Count_Attribute, 41
A_Decimal_Fixed_Point_Definition, 37, 145, 146
A_Declaration, 13, 14, 20, 31, 32, 94, 95, 96, 97, 98, 99, 100, 101, 104, 106, 110, 116, 121, 127, 128, 129, 132, 136, 154, 155, 157, 158, 160, 164, 185, 224, 238
A_Default_In_Mode, 36, 56, 96
A_Deferred_Constant_Declaration, 33, 95, 114, 116
A_Defining_Character_Literal, 33, 111, 112
A_Defining_Enumeration_Literal, 33, 111, 112
A_Defining_Expanded_Name, 33, 111, 112
A_Defining_Identifier, 33, 110, 111, 112, 181, 184
A_Defining_Name, 13, 20, 31, 32, 94, 95, 99, 110, 111, 115, 139, 152, 161, 162, 167, 169, 170, 171, 172, 173, 176, 181, 190
A_Defining_Operator_Symbol, 33, 99, 110, 111
A_Definite_Attribute, 41
A_Definition, 13, 20, 31, 32, 94, 95, 96, 97, 100, 101, 140, 154, 156, 169, 170, 183
A_Delay_Relative_Statement, 42, 193
A_Delay_Until_Statement, 42, 193
A_Delta_Attribute, 41
A_Delta_Constraint, 39, 146, 151
A_Denorm_Attribute, 41
A_Derived_Record_Extension_Definition, 37, 96, 141, 142, 143, 144
A_Derived_Type_Definition, 37, 96, 141, 142, 143, 144, 221, 222, 223, 225, 229
A_Digits_Attribute, 41
A_Digits_Constraint, 39, 146, 151
A_Discard_Names_Pragma, 32
A_Discrete_Range, 37, 98, 149, 150, 152, 153, 156, 166, 169, 183
A_Discrete_Range_Attribute_Reference, 39, 151, 153
A_Discrete_Simple_Expression_Range, 39, 150, 151, 153, 199
A_Discrete_Subtype_Definition, 37, 98, 119, 132, 147, 149, 150, 153
A_Discrete_Subtype_Indication, 39, 149, 150, 153
A_Discriminant_Association, 39, 152, 172, 173, 174
A_Discriminant_Constraint, 39, 152
A_Discriminant_Specification, 33, 35, 95, 115, 116, 154, 224, 225
A_Divide_Operator, 40
A_First_Attribute, 41, 167
A_First_Bit_Attribute, 41
A_Floating_Point_Definition, 37, 145, 146
A_Floor_Attribute, 41
A_For_Loop_Statement, 42, 105, 183, 184, 185
A_Fore_Attribute, 41
A_Formal_Access_Type_Definition, 38, 148, 149, 158
A_Formal_Constrained_Array_Definition, 38, 147, 148, 158
A_Formal_Decimal_Fixed_Point_Definition, 38, 158
A_Formal_Declaration, 34
A_Formal_Derived_Type_Definition, 37, 96, 142, 143, 149, 158

A_Formal_Discrete_Type_Definition, 38, 158
A_Formal_Floating_Point_Definition, 38, 158
A_Formal_Function_Declaration, 34, 96, 120, 136, 138, 140, 175
A_Formal_Modular_Type_Definition, 38, 158
A_Formal_Object_Declaration, 34, 96, 115, 116, 136, 138
A_Formal_Ordinary_Fixed_Point_Definition, 38, 158
A_Formal_Package_Declaration, 34, 136, 137, 139
A_Formal_Package_Declaration_With_Box, 34, 136, 137, 139
A_Formal_Private_Type_Definition, 37, 96, 158
A_Formal_Procedure_Declaration, 34, 96, 120, 136, 138, 189
A_Formal_Signed_Integer_Type_Definition, 38, 158
A_Formal_Tagged_Private_Type_Definition, 37, 96, 158
A_Formal_Type_Declaration, 34, 113, 118, 119, 136, 138, 143, 144
A_Formal_Type_Definition, 37, 97, 114, 140, 142, 143, 149, 158
A_Formal_Unconstrained_Array_Definition, 38, 147, 148, 158
A_Fraction_Attribute, 41
A_Full_Type_Declaration, 34, 104
A_Function, 45, 46, 73, 74, 75, 76, 89, 90
A_Function_Body, 25, 45, 46, 66, 71, 73, 74, 75, 76, 83, 89, 90
A_Function_Body_Declaration, 33, 91, 105, 120, 121, 122, 123, 124, 126, 127, 132, 134, 135, 140, 175
A_Function_Body_Stub, 34, 120, 124, 126, 133, 134, 135, 140, 175
A_Function_Body_Subunit, 46, 75, 76, 82, 83, 89, 90
A_Function_Call, 40, 165, 175, 176, 196, 241
A_Function_Declaration, 33, 91, 95, 120, 124, 125, 126, 127, 128, 132, 139, 140, 143, 175
A_Function_Instance, 45, 73, 74, 75, 76, 89, 90
A_Function_Instantiation, 34, 91, 124, 126, 136, 137, 140, 175
A_Function_Renaming, 45, 46, 73, 74, 75, 76, 89, 90
A_Function_Renaming_Declaration, 34, 91, 120, 124, 125, 126, 127, 130, 131, 139, 140, 175
A_Generic_Association, 39, 138, 171, 172, 174
A_Generic_Declaration, 34
A_Generic_Function, 45, 73, 74, 75, 76, 89, 90
A_Generic_Function_Declaration, 34, 91, 105, 120, 124, 125, 126, 135, 136, 175
A_Generic_Function_Renaming, 45, 73, 74, 75, 76, 89, 90
A_Generic_Function_Renaming_Declaration, 34, 91, 130, 131
A_Generic_Instantiation, 34
A_Generic_Package, 45, 46, 73, 74, 75, 76, 89, 90
A_Generic_Package_Declaration, 34, 91, 105, 123, 124, 125, 126, 128, 129, 135, 136
A_Generic_Package_Renaming, 45, 46, 73, 74, 75, 76, 89, 90
A_Generic_Package_Renaming_Declaration, 34, 91, 130, 131
A_Generic_Procedure, 45, 46, 73, 74, 75, 76, 89, 90
A_Generic_Procedure_Declaration, 34, 91, 105, 120, 125, 126, 135, 189
A_Generic_Procedure_Renaming, 45, 46, 73, 74, 75, 76, 89, 90
A_Generic_Procedure_Renaming_Declaration, 34, 91, 130, 131
A_Generic_Renaming, 46
A_Generic_Unit_Declaration, 46
A_Generic_Unit_Instance, 46
A_Goto_Statement, 42, 188
A_Greater_Than_Operator, 40
A_Greater_Than_Or_Equal_Operator, 40
A_Known_Discriminant_Part, 37, 113, 153, 154
A_Last_Attribute, 41, 167
A_Last_Bit_Attribute, 41
A_Leading_Part_Attribute, 41
A_Length_Attribute, 41, 167
A_Less_Than_Operator, 40
A_Less_Than_Or_Equal_Operator, 40
A_Library_Unit_Body, 46
A_Limited_Private_Trait, 35
A_Limited_Trait, 35
A_Linker_Options_Pragma, 32
A_List_Pragma, 32
A_Locking_Policy_Pragma, 32
A_Loop_Parameter_Specification, 33, 95, 119, 184
A_Loop_Statement, 42, 104, 105, 183, 184, 185
A_Machine_Attribute, 41
A_Machine_Emax_Attribute, 41
A_Machine_Emin_Attribute, 41
A_Machine_Mantissa_Attribute, 41
A_Machine_Overflows_Attribute, 41
A_Machine_Radix_Attribute, 41
A_Machine_Rounds_Attribute, 41
A_Max_Attribute, 41
A_Max_Size_In_Storage_Elements_Attribute, 41
A_Min_Attribute, 41
A_Minus_Operator, 40
A_Mod_Operator, 40
A_Model_Attribute, 41
A_Model_Emin_Attribute, 41
A_Model_Epsilon_Attribute, 41
A_Model_Mantissa_Attribute, 41
A_Model_Small_Attribute, 41
A_Modular_Type_Definition, 37, 145
A_Modulus_Attribute, 41
A_Multiply_Operator, 40
A_Name_Default, 36, 138
A_Named_Array_Aggregate, 40, 168
A_Nil_Default, 36
A_Nonexistent_Body, 25, 46, 70, 74, 75, 76, 82, 83, 87, 91, 92, 93
A_Nonexistent_Declaration, 25, 46, 70, 73, 74, 75, 76, 87, 91, 92, 93
A_Normalize_Scalars_Pragma, 32
A_Not_Equal_Operator, 40
A_Not_In_Range_Membership_Test, 40, 177, 178
A_Not_In_Type_Membership_Test, 40, 177, 178
A_Not_Operator, 40

A_Null_Component, 37, 154
A_Null_Literal, 40
A_Null_Record_Definition, 37, 141, 155
A_Null_Statement, 42
A_Number_Declaration, 34
A_Pack_Pragma, 32
A_Package, 45, 73, 74, 75, 76, 89, 90
A_Package_Body, 45, 46, 73, 74, 75, 76, 83, 89, 90
A_Package_Body_Declaration, 34, 91, 105, 121, 122, 123, 124, 126, 129, 134, 135
A_Package_Body_Stub, 34, 124, 126, 133, 134, 135
A_Package_Body_Subunit, 46, 75, 76, 82, 83, 89, 90
A_Package_Declaration, 34, 91, 105, 123, 124, 125, 126, 128, 129
A_Package_Instance, 45, 46, 73, 74, 75, 76, 89, 90
A_Package_Instantiation, 34, 73, 91, 124, 126, 136, 137
A_Package_Renaming, 45, 73, 74, 75, 76, 89, 90
A_Package_Renaming_Declaration, 34, 91, 125, 126, 130, 131
A_Page_Pragma, 32
A_Parameter_Association, 39, 171, 172, 174, 176, 190
A_Parameter_Specification, 31, 33, 95, 96, 115, 116, 120, 121, 149, 191
A_Parenthesized_Expression, 40, 174
A_Partition_ID_Attribute, 41
A_Path, 13, 20, 31, 32, 94, 100, 105, 181, 182
A_Plus_Operator, 40
A_Pool_Specific_Access_To_Variable, 38, 148
A_Pos_Attribute, 41
A_Position_Attribute, 41
A_Positional_Array_Aggregate, 40, 168
A_Pragma, 13, 20, 31, 32, 46, 91, 92, 93, 94, 105, 106, 115, 116, 121, 122, 126, 127, 128, 129, 132, 136, 154, 156, 157, 158, 182, 183, 185, 186, 192, 195, 199
A_Pragma_Argument_Association, 39, 106, 171, 172
A_Pred_Attribute, 41
A_Predefined_Unit, 47
A_Preelaborate_Pragma, 32
A_Priority_Pragma, 32
A_Private_Body, 47
A_Private_Declaration, 47
A_Private_Extension_Declaration, 33, 34, 95, 113, 117, 118, 119, 129, 143, 144
A_Private_Extension_Definition, 37, 95, 114, 142, 143, 156, 157
A_Private_Trait, 35
A_Private_Type_Declaration, 33, 95, 113, 117, 118, 119, 129, 143, 144
A_Private_Type_Definition, 37, 95, 114, 156
A_Procedure, 45, 46, 73, 74, 75, 76, 89, 90
A_Procedure_Body, 25, 45, 46, 66, 71, 73, 74, 75, 76, 82, 83, 89, 90
A_Procedure_Body_Declaration, 33, 91, 105, 120, 121, 122, 123, 124, 126, 127, 132, 134, 135, 189
A_Procedure_Body_Stub, 34, 120, 124, 126, 133, 134, 135, 189
A_Procedure_Body_Subunit, 46, 75, 76, 82, 83, 89, 90
A_Procedure_Call_Statement, 42, 188, 189, 190, 196, 241
A_Procedure_Declaration, 33, 91, 95, 119, 120, 125, 126, 127, 132, 139, 143, 189
A_Procedure_Instance, 45, 46, 73, 74, 75, 76, 89, 90
A_Procedure_Instantiation, 34, 91, 124, 126, 136, 137, 189
A_Procedure_Renaming, 45, 46, 73, 74, 75, 76, 89, 90
A_Procedure_Renaming_Declaration, 34, 91, 120, 124, 125, 126, 127, 130, 131, 139, 189
A_Protected_Body_Declaration, 34, 91, 105, 123, 125, 126, 131, 132, 134, 135
A_Protected_Body_Stub, 34, 125, 126, 133, 134, 135
A_Protected_Body_Subunit, 46, 75, 76, 82, 83, 89, 90
A_Protected_Definition, 37, 105, 114, 157, 158
A_Protected_Type_Declaration, 33, 34, 113, 117, 118, 119, 123, 124, 126, 131, 143, 144
A_Public_Body, 47
A_Public_Declaration, 47
A_Public_Declaration_And_Body, 25, 47, 75
A_Pure_Pragma, 32
A_Qualified_Expression, 40, 178, 179, 195
A_Queuing_Policy_Pragma, 32
A_Raise_Statement, 42, 195
A_Range_Attribute, 41, 167
A_Range_Attribute_Reference, 39, 151, 178
A_Read_Attribute, 41
A_Real_Literal, 39, 160
A_Real_Number_Declaration, 33, 34, 115, 116
A_Record_Aggregate, 40, 167
A_Record_Component_Association, 39, 167, 170, 174
A_Record_Definition, 37, 105, 141, 154, 155
A_Record_Representation_Clause, 44, 105, 198, 199
A_Record_Type_Definition, 35, 37, 96, 141, 148, 214, 221, 222, 223, 225, 229
A_Rem_Operator, 40
A_Remainder_Attribute, 41
A_Remote_Call_Interface_Pragma, 32
A_Remote_Types_Pragma, 32
A_Renaming, 46
A_Renaming_Declaration, 34
A_Representation_Clause, 44, 101, 119, 132, 197, 198
A_Requeue_Statement, 42, 193
A_Requeue_Statement_With_Abort, 42, 193
A_Restrictions_Pragma, 32
A_Return_Statement, 42, 187
A_Reverse_Trait, 35
A_Reviewable_Pragma, 32
A_Root_Integer_Definition, 38
A_Root_Real_Definition, 38
A_Root_Type_Definition, 37, 98, 110, 149
A_Round_Attribute, 41
A_Rounding_Attribute, 41
A_Safe_First_Attribute, 42
A_Safe_Last_Attribute, 42
A_Scale_Attribute, 42

A_Scaling_Attribute, 42
A_Select_Path, 43, 193
A_Selected_Component, 40, 104, 112, 116, 120, 137, 147, 149, 165, 166, 178, 187, 194, 195, 197
A_Selective_Accept_Statement, 42, 182
A_Separate_Body, 47
A_Shared_Passive_Pragma, 32
A_Signed_Integer_Type_Definition, 37, 144, 145
A_Signed_Zeros_Attribute, 42
A_Simple_Dynamic_Model, 218, 221, 223, 225, 227
A_Simple_Expression_Range, 39, 145, 146, 150, 151, 178
A_Simple_Static_Model, 218, 221, 222, 223, 225, 226, 227
A_Single_Protected_Declaration, 33, 34, 114, 123, 124, 126, 131
A_Single_Task_Declaration, 33, 114, 123, 125, 126, 131
A_Size_Attribute, 42
A_Slice, 40, 165, 166
A_Small_Attribute, 42
A_Statement, 13, 14, 20, 31, 32, 94, 95, 96, 97, 98, 99, 100, 104, 106, 122, 164, 181, 183, 184, 185, 186, 187, 188, 192, 195, 241
A_Storage_Pool_Attribute, 42
A_Storage_Size_Attribute, 42
A_Storage_Size_Pragma, 32
A_String_Literal, 39, 160
A_Subprogram_Body, 46
A_Subprogram_Declaration, 46
A_Subprogram_Renaming, 46
A_Subtype_Declaration, 33, 113, 114, 118, 119, 143
A_Subtype_Indication, 36, 114, 141, 148, 149, 150, 153, 179, 225
A_Subunit, 46, 134
A_Succ_Attribute, 42
A_Suppress_Pragma, 33
A_Tag_Attribute, 42
A_Tagged_Private_Type_Definition, 37, 95, 114, 156
A_Tagged_Record_Type_Definition, 35, 37, 96, 141, 148
A_Task_Body_Declaration, 34, 91, 105, 121, 122, 123, 124, 126, 131, 134, 135
A_Task_Body_Stub, 34, 124, 126, 133, 134, 135
A_Task_Body_Subunit, 46, 75, 76, 82, 83, 89, 90
A_Task_Definition, 37, 105, 114, 157, 158
A_Task_Dispatching_Policy_Pragma, 33
A_Task_Type_Declaration, 33, 113, 117, 118, 119, 123, 125, 126, 131, 143, 144
A_Terminate_Alternative_Statement, 42
A_Terminated_Attribute, 42
A_Then_Abort_Path, 44
A_Timed_Entry_Call_Statement, 42, 182
A_Truncation_Attribute, 42
A_Type_Conversion, 40, 178, 179
A_Type_Declaration, 34
A_Type_Definition, 14, 36, 97, 114, 127, 140, 142, 143, 214, 215, 218, 221, 222, 223, 225, 226, 227, 229
A_Unary_Minus_Operator, 40
A_Unary_Plus_Operator, 40
A_Universal_Fixed_Definition, 38
A_Universal_Integer_Definition, 38
A_Universal_Real_Definition, 38
A_Use_Package_Clause, 44, 92, 136, 197
A_Use_Type_Clause, 44, 136, 197
A_Val_Attribute, 42
A_Valid_Attribute, 42
A_Value_Attribute, 42
A_Variable_Declaration, 14, 33, 34, 35, 95, 114, 115
A_Variant, 37, 105, 154, 155, 156
A_Variant_Part, 37, 105, 154, 155, 156
A_Version_Attribute, 42
A_Volatile_Components_Pragma, 33
A_Volatile_Pragma, 33
A_While_Loop_Statement, 42, 105, 183, 184, 185
A_Wide_Image_Attribute, 42
A_Wide_Value_Attribute, 42
A_Wide_Width_Attribute, 42
A_Width_Attribute, 42
A_With_Clause, 44, 92, 197
A_Write_Attribute, 42
Abandon_Children, 49, 109
Abandon_Siblings, 49, 109
Aborted_Tasks, 194
abs, 40, 111, 140, 160
Accept_Body_Exception_Handlers, 192
Accept_Body_Statements, 192
Accept_Entry_Direct_Name, 191
Accept_Entry_Index, 191
Accept_Parameters, 191
accept_statement, 191
access_definition, 35
Access_To_Function_Result_Profile, 148, 149, 158
Access_To_Object_Definition, 38, 148, 158
access_to_subprogram, 176, 190
access_to_subprogram_definition, 148
Access_To_Subprogram_Definition, 38
Access_To_Subprogram_Parameter_Profile, 148, 158
Access_Type_Definition, 30, 97, 148, 149
Access_Type_Kind, 97
Access_Type_Kinds, 31, 37, 38, 94, 97
Actual_Parameter, 171
Ada_Environments, xiv, 2, 11, 19, 21, 24, 60, 65, 69, 237, 238, 240, 242
Ada_Environments.Containers, 19
aggregate, 167, 168, 195
aliased, 35
All_Named_Components, 225
allocator, 179
Allocator_Qualified_Expression, 179
Allocator_Subtype_Indication, 179
AN, 3
An_Abort_Statement, 42, 194
An_Abs_Operator, 40
An_Abstract_Limited_Private_Trait, 36
An_Abstract_Limited_Trait, 36
An_Abstract_Private_Trait, 36
An_Abstract_Trait, 36

An_Accept_Statement, 42, 105, 184, 191, 192
An_Access_Attribute, 41
An_Access_Definition_Trait, 31, 35
An_Access_To_Constant, 38, 148
An_Access_To_Function, 38, 149
An_Access_To_Procedure, 38, 149
An_Access_To_Protected_Function, 38, 149
An_Access_To_Protected_Procedure, 38, 149
An_Access_To_Variable, 38, 148
An_Access_Type_Definition, 37, 98, 148, 149
An_Address_Attribute, 41
An_Adjacent_Attribute, 41
An_Aft_Attribute, 41
An_Aliased_Trait, 35
An_Alignment_Attribute, 41
An_All_Calls_Remote_Pragma, 32
An_Allocation_From_Qualified_Expression, 40, 179
An_Allocation_From_Subtype, 40, 179
An_And_Operator, 40
An_And_Then_Short_Circuit, 40, 177
An_Application_Unit, 47, 242
An_Array_Component_Association, 39, 168, 169, 170
An_Assignment_Statement, 42, 181, 182
An_Association, 13, 20, 31, 32, 94, 100, 140, 159, 169, 171, 173
An_Asynchronous_Pragma, 32
An_Asynchronous_Select_Statement, 42, 182
An_At_Clause, 44, 197, 198
An_Atomic_Components_Pragma, 32
An_Atomic_Pragma, 32
An_Attach_Handler_Pragma, 32
An_Attribute_Definition_Clause, 44, 154, 197, 198
An_Attribute_Reference, 40, 99, 116, 120, 149, 151, 165, 166, 167, 178, 197, 198, 225
An_Elaborate_All_Pragma, 32
An_Elaborate_Body_Pragma, 32
An_Elaborate_Pragma, 32
An_Else_Path, 43
An_Elsif_Path, 43, 182
An_Entry_Body_Declaration, 34, 105, 120, 121, 122, 123, 124, 126, 132, 133
An_Entry_Call_Statement, 42, 188, 189, 190, 241
An_Entry_Declaration, 34, 120, 125, 126, 132, 189, 192
An_Entry_Index_Specification, 34, 119, 133
An_Enumeration_Literal, 39, 139, 160, 161, 162, 164, 166
An_Enumeration_Literal_Specification, 33, 116, 144
An_Enumeration_Representation_Clause, 44, 198
An_Enumeration_Type_Definition, 37, 144
An_Equal_Operator, 40
An_Exception_Declaration, 34, 135
An_Exception_Handler, 13, 20, 31, 32, 94, 105, 122, 181, 186, 192, 194, 195
An_Exception_Renaming_Declaration, 34, 130, 131
An_Exit_Statement, 42, 186, 187
An_Explicit_Declaration, 36, 238
An_Explicit_Dereference, 40, 164, 165
An_Exponent_Attribute, 41
An_Exponentiate_Operator, 40
An_Export_Pragma, 32
An_Expression, 13, 14, 20, 31, 32, 94, 95, 99, 104, 115, 130, 131, 133, 138, 139, 145, 146, 150, 151, 152, 156, 159, 160, 161, 164, 165, 167, 168, 169, 170, 171, 172, 173, 174, 176, 177, 179, 181, 182, 183, 184, 187, 188, 191, 193, 194, 198, 199, 241
An_Extension_Aggregate, 40, 167, 168
An_External_Tag_Attribute, 41
An_Identifier, 39, 104, 112, 116, 120, 137, 139, 147, 149, 155, 160, 161, 162, 163, 164, 166, 169, 170, 171, 172, 178, 187, 188, 191, 194, 195, 197, 198
An_Identity_Attribute, 41
An_If_Path, 43, 182
An_If_Statement, 42, 182
An_Image_Attribute, 41
An_Implementation_Defined_Attribute, 42, 167
An_Implementation_Defined_Pragma, 33
An_Implementation_Unit, 47
An_Implicit_Inherited_Declaration, 36
An_Implicit_Predefined_Declaration, 36
An_Import_Pragma, 32
An_In_Mode, 36
An_In_Out_Mode, 36
An_In_Range_Membership_Test, 40, 177, 178
An_In_Type_Membership_Test, 40, 177, 178
An_Incomplete_Type_Declaration, 33, 113, 117, 143
An_Index_Constraint, 39, 152
An_Indexed_Component, 40, 165, 188
An_Inline_Pragma, 32
An_Input_Attribute, 41
An_Inspection_Point_Pragma, 32
An_Integer_Literal, 39, 160
An_Integer_Number_Declaration, 33, 34, 115, 116
An_Interrupt_Handler_Pragma, 32
An_Interrupt_Priority_Pragma, 32
An_Object_Declaration, 34
An_Object_Renaming_Declaration, 34, 116, 129, 130, 131
An_Operator_Symbol, 39, 99, 137, 139, 160, 161, 162, 163, 164, 166
An_Optimize_Pragma, 32
An_Or_Else_Short_Circuit, 40, 177
An_Or_Operator, 40
An_Or_Path, 43, 44, 193
An_Ordinary_Fixed_Point_Definition, 37, 146
An_Ordinary_Trait, 35
An_Ordinary_Type_Declaration, 33, 34, 113, 117, 118, 119, 143, 144
An_Others_Choice, 37, 156, 169, 170, 183, 194
An_Out_Mode, 36
An_Output_Attribute, 41
An_Unbiased_Rounding_Attribute, 42
An_Unchecked_Access_Attribute, 42
An_Unconstrained_Array_Definition, 37, 146, 147, 215, 223
An_Unknown_Attribute, 42, 167
An_Unknown_Discriminant_Part, 37, 113, 153
An_Unknown_Pragma, 33
An_Unknown_Unit, 44, 46, 47, 69, 70, 73, 74, 75, 76, 77, 89, 90, 91, 92, 93

An_Xor_Operator, 40
Ancestor, 234
ancestor_part, 168
ancestor_subtype_indication, 157
Ancestor_Subtype_Indication, 156, 157
Ancestors, 47, 48
and, 140
architecture, 18
array_aggregate, 168
Array_Component, 212, 214, 215, 218, 219, 221, 222, 223, 224, 225, 226, 227, 228, 230, 264
array_component_association, 168, 170
Array_Component_Associations, 168
Array_Component_Choices, 168, 169
Array_Component_Definition, 146, 147, 158
Array_Component_Iterator, 216, 220, 223, 224, 227, 228, 230
Array_Component_List, 215
Array_Components, 223
Array_Index, 220
Array_Indexes, 220
Array_Iterator, 223
Array_Length, 224, 225, 226, 227, 228
Array_Type, 232
array_type_definition, 114, 147
Asis, 2
ASIS application, 11, 234
ASIS implementation, 234
ASIS queries, 234
ASIS_Failed, 4, 25, 28, 52, 54, 55, 60, 61, 159, 161, 162, 200, 206, 239, 242
ASIS_Implementor, 53
ASIS_Implementor_Information, 53
ASIS_Implementor_Version, 53
ASIS_Inappropriate_Compilation_Unit, 8, 52, 70, 82, 239, 242
ASIS_Inappropriate_Container, 52, 66, 239, 242
ASIS_Inappropriate_Context, 8, 24, 52, 54, 61, 65, 66, 67, 239, 242
ASIS_Inappropriate_Element, 8, 14, 25, 52, 91, 104, 109, 127, 163, 176, 190, 210, 213, 220, 224, 227, 228, 229, 239, 242
ASIS_Inappropriate_Line, 8, 52, 239, 242
ASIS_Inappropriate_Line_Number, 8, 52, 205, 206, 239, 242
ASIS_Integer, 27, 28, 107, 209
ASIS_Natural, 27, 28, 86, 201, 220, 225, 226, 227, 228
ASIS_Positive, 27, 28, 215, 217, 224, 227, 228
ASIS_Version, 53
Assignment_Expression, 181
Assignment_Variable_Name, 181
Associate, 11, 19, 21, 24, 60, 238, 242
Association, 30, 99, 169, 170, 171, 172, 173, 174
Association_Kind, 99, 174
Association_Kinds, 31, 39, 94, 99, 167, 174
Association_List, 30, 106, 137, 167, 168, 176, 190
asynchronous_select, 193
Attribute, 81, 85
attribute_definition_clause, 265
attribute_designator, 163, 166, 167
Attribute_Designator_Expressions, 167
Attribute_Designator_Identifier, 166
Attribute_Kind, 99
Attribute_Kinds, 31, 41, 94, 99
attribute_reference, 48, 151, 165, 166, 175, 189
Attribute_Time, 85
Attribute_Value_Delimiter, 81
Attribute_Values, 81
Attributes_Are_Supported, 58

B

benefits, 258, 259
Block_Declarative_Items, 185
Block_Exception_Handlers, 186
block_statement, 111, 183, 185, 186
Block_Statements, 186
body, 10, 25, 45, 47, 48, 76, 83, 192
Body_Block_Statement, 121, 122, 129, 131, 132, 186
Body_Declarative_Items, 121, 122, 129, 131, 132
Body_Exception_Handlers, 121, 122, 129, 131, 132
Body_Statements, 121, 122, 129, 131, 132
Body_Stub, 133

C

Calendar, 84, 85
Call_Statement_Parameters, 57, 109, 174, 190
Call_Statement_Parameters_Normalized, 57, 190
Called_Name, 188
Can_Be_Main_Program, 79
Capacity_Error, 8, 51, 61, 206
Case_Expression, 183
case_statement, 43
case_statement_alternative, 183
Case_Statement_Alternative, 30
Case_Statement_Alternative_Choices, 183
character, 200, 202
Character_Position, 201, 202, 204
Character_Position_Positive, 201, 202
Choice_Parameter_Specification, 194
Circular_Length, 86, 88
Clause, 30, 100, 197, 198, 199
Clause_Kind, 100
Clause_Kinds, 31, 44, 94, 100, 101
Clause_Names, 197
Clauses, xv, 2, 13, 20, 21, 32, 197
Close, 19, 21, 61, 238, 242
Closure, 49, 234
code analysis, 254
comment, 200
Comment_Image, 207
compilation, xiv, xv, 1, 2, 10, 11, 12, 13, 19, 20, 21, 25, 27, 28, 29, 44, 45, 46, 48, 49, 60, 62, 65, 67, 69, 78, 86, 90, 161, 197, 200
compilation unit, 262
Compilation unit, 234

Compilation_Command_Line_Options, 81
Compilation_CPU_Duration, 84
Compilation_Pragmas, 92, 93, 104
Compilation_Span, 203
compilation_unit, 46, 70, 77, 88, 91, 93, 234, 262
Compilation_Unit, xiv, 15, 19, 27, 29, 44, 45, 50, 52, 54, 61, 62, 69, 70, 71, 72, 73, 74, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 87, 91, 92, 93, 123, 125, 164, 203, 205, 206, 209, 210, 234, 237, 240, 264
Compilation_Unit_Bodies, 67, 72
Compilation_Unit_Body, 71
Compilation_Unit_List, 27, 45, 50, 66, 67, 71, 72, 77, 82, 86, 88, 89, 242
Compilation_Unit_Span, 203
Compilation_Units, xiv, 2, 11, 19, 21, 45, 67, 69, 72, 84, 86, 88, 89, 91, 237, 238, 240, 241, 242
Compilation_Units.Relations, xiv
Compilation_Units.Times, xiv
Component, 221, 222, 223, 224, 225, 226, 227, 228
component_choice_list, 169
component_clause, 199
Component_Clause, 30, 199
Component_Clause_List, 30, 198
Component_Clause_Position, 199
Component_Clause_Range, 199
Component_Clauses, 198
Component_Data_Stream, 224
component_declaration, 167, 169
Component_Declaration, 30, 114, 219, 224
Component_Definition, 30, 114, 147, 153
Component_Expression, 170
Component_Indication, 225
component_selector_name, 56, 168, 169
Component_Subtype_Indication, 153
condition, 43, 182, 184, 187
Condition_Expression, 182
conditional_entry_call, 193
Configuration_Pragmas, 92
Configuraton_Pragmas, 46
Consistent_Length, 86, 88
constrained_array_definition, 147
Constrained_Subtype, 228, 231
constraint, 57, 118, 174
Constraint, 30, 98, 150, 151, 152
Constraint_Error, 231
Constraint_Kind, 98
Constraint_Kinds, 31, 36, 39, 94, 98, 146
Construct_Artificial_Data_Stream, 222, 224, 227, 229
Container, 19, 52, 65, 66, 67, 68, 70, 234, 264
Container_List, 65
Containers, 65
Containers, 65, 69
context, 20, 262, 266
Context, xiv, 8, 11, 19, 21, 24, 25, 27, 28, 29, 46, 50, 52, 54, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 76, 77, 78, 82, 88, 89, 92, 101, 104, 117, 123, 124, 125, 133, 134, 204, 209, 210, 218, 235, 237, 240, 264
Context_Clause, 30, 93
Context_Clause_Elements, 90, 92, 93, 104
Context_Clause_List, 30, 92
Continue, 49, 109, 237, 242
Control, 108
Convert, 231, 232
Converted_Or_Qualified_Expression, 179
Converted_Or_Qualified_Subtype_Mark, 178
CORBA, 250
Corresponding_, 16
Corresponding_Base_Entity, 130
Corresponding_Body, 16, 25, 73, 76, 77, 125, 126, 136
Corresponding_Body_Stub, 16, 134
Corresponding_Called_Entity, 189, 241
Corresponding_Called_Function, 175, 241
Corresponding_Children, 16, 25, 72
Corresponding_Constant_Declaration, 115
Corresponding_Declaration, 16, 25, 74, 123, 136
Corresponding_Destination_Statement, 188
Corresponding_Entry, 16, 192
Corresponding_Equality_Operator, 128
Corresponding_Expression_Type, 159
Corresponding_First_Subtype, 117, 150, 159
Corresponding_Generic_Element, 139
Corresponding_Last_Constraint, 118
Corresponding_Last_Subtype, 118
Corresponding_Loop_Exited, 16, 187
Corresponding_Name_Declaration, 16, 136, 150, 159, 160, 164
Corresponding_Name_Definition, 16, 115, 159, 160, 162, 163, 164
Corresponding_Name_Definition_List, 160, 163, 164
Corresponding_Parent_Declaration, 25, 73
Corresponding_Parent_Subtype, 143, 159
Corresponding_Pragmas, 106, 142
Corresponding_Representation_Clauses, 119, 142
Corresponding_Root_Type, 143
Corresponding_Subprogram_Derivation, 127
Corresponding_Subunit, 133
Corresponding_Subunit_Parent_Body, 82, 83
Corresponding_Type, 16, 127
Corresponding_Type_Declaration, 16, 117
Corresponding_Type_Operators, 140
Corresponding_Type_Structure, 144
CR, 3
Create_Element, 210
Create_Id, 16, 210

D

Data_Decomposition, xv, 2, 3, 20, 21, 212, 221, 231, 266
Data_Error, 8, 51, 61, 214, 220, 224, 227, 228, 229
Data_Stream, 222, 224, 226, 227, 228, 229
Debug_Image, 64, 83, 107, 206, 208, 211
Declaration, 14, 30, 91, 95, 96, 110, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 127, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138, 139, 143, 144, 159, 175, 184, 189, 192, 194, 224, 240, 241

Declaration_Kind, 13, 14, 95, 237
Declaration_Kinds, 31, 33, 34, 35, 94, 95, 96, 120, 123, 129, 134, 192, 237
Declaration_List, 30, 105, 131, 140, 141, 142, 144
Declaration_Origin, 94, 96, 238
Declaration_Origins, 31, 36, 96
Declaration_Subtype_Mark, 116, 121, 130, 138
Declarations, xv, 2, 13, 20, 21, 32, 110, 237, 238, 240, 241
declarative_item, 121, 185
Declarative_Item_List, 30, 128, 129, 157, 185
declarative_part, 10, 29, 185
default_expression, 56, 57, 171, 173, 176
Default_In_Mode_Supported, 56
Default_Kind, 96
default_name, 36, 56, 171
Default_Name, 60
Default_Parameters, 60
deference, 190
defining_character_literal, 162
Defining_Containers, 65, 66
defining_identifier, 33, 56, 57, 138, 153, 162, 168, 169, 170, 177, 190
Defining_Name, 30, 95, 111, 112, 115, 139, 162, 170, 176
Defining_Name_Image, 110, 111, 112, 238, 241
Defining_Name_Kind, 95
Defining_Name_Kinds, 31, 33, 94, 95, 99
Defining_Name_List, 14, 30, 110, 164, 181, 225, 238
defining_operator_symbol, 162
Defining_Prefix, 112
defining_program_unit_name, 71, 73, 162
Defining_Selector, 112
Definition, 14, 30, 97, 98, 113, 114, 141, 142, 145, 146, 149, 150, 154, 155, 157, 158
Definition_Kind, 97
Definition_Kinds, 31, 35, 36, 94, 95, 97, 158
Definition_List, 30, 147
Definitions, xv, 2, 13, 14, 20, 21, 32, 140
delay_alternative, 44
Delay_Expression, 193
Delimiter_Image, 64, 83, 107, 206, 208, 211
Delta_Expression, 145, 146, 151
Dependent, 235
Dependent_Units, 88
Dependents, 48
derived_type_definition, 141, 142, 143, 144, 162
Descendants, 47, 48, 72, 235
design goals, 259
designator, 111, 166, 167
Diagnosis, 8, 51, 54, 55, 210, 214, 220, 224, 227, 228, 231, 239, 242, 244
DIANA, 256
Digits_Expression, 145, 151
Dimension, 226
Dimension_Indexes, 215, 220, 224, 227, 228
direct_name, 198
discrete_choice_list, 43, 156, 169
discrete_range, 98, 150, 151, 152
Discrete_Range, 30, 98, 166, 199
Discrete_Range_Kind, 98
Discrete_Range_Kinds, 31, 37, 39, 94, 98, 149, 150, 151
Discrete_Range_List, 30, 152
Discrete_Ranges, 151, 152
Discrete_Subtype_Definition, 30, 119, 132, 147
Discrete_Subtype_Definitions, 147, 158
Discriminant, 229
discriminant_association, 57, 152, 172, 173
Discriminant_Association, 30, 57, 172, 173, 174
Discriminant_Association list, 153
Discriminant_Association_List, 30, 152
Discriminant_Associations, 109, 152, 174
Discriminant_Associations_Normalized, 57
Discriminant_Components, 221
discriminant_constraint, 152
Discriminant_Direct_Name, 155
Discriminant_Expression, 173
discriminant_part, 113
Discriminant_Part, 113, 116, 129, 131, 138
discriminant_selector_name, 57, 152, 153, 167, 172
Discriminant_Selector_Names, 172
discriminant_specification, 152, 154, 157, 167, 173
Discriminant_Specification_List, 30, 154
Discriminants, 153, 154
Dissociate, 19, 21, 24, 61, 238, 242
Done, 216, 220, 224, 227, 228

E

Elaborate, 90, 92, 93
Elaboration_Order, 86, 88, 89
Element, xiv, 12, 13, 14, 19, 20, 24, 25, 27, 29, 30, 50, 52, 54, 61, 62, 75, 76, 82, 83, 91, 94, 95, 99, 101, 102, 103, 105, 106, 107, 108, 115, 139, 161, 162, 164, 170, 171, 173, 174, 175, 184, 186, 193, 197, 200, 202, 203, 204, 205, 206, 208, 209, 210, 215, 235, 237, 238, 240, 241, 264
Element Kinds, 12, 27, 31
Element_Image, 16, 111, 206
Element_Kind, 14, 15, 94, 241
Element_Kinds, 20, 31, 32, 35, 94, 97, 100, 105, 106, 129
Element_List, 15, 27, 29, 30, 50, 101, 121, 135, 156, 183, 194
Element_Span, 16, 202
Element_Type, 232
Elements, xiv, 2, 13, 14, 15, 16, 20, 21, 31, 32, 91, 237, 238, 240, 241, 242
elsif, 43
Enclosing_Compilation_Unit, 15, 46, 62, 91, 101, 104, 117, 124, 126, 134, 210, 219, 241
Enclosing_Container, 66, 67, 70
Enclosing_Context, 62, 65, 66, 67, 70, 71, 72, 73, 74, 75, 77, 78, 82, 83, 86, 89, 117, 124, 125, 135, 210
Enclosing_Element, 15, 16, 103, 106, 108, 124, 126, 140, 142, 155, 159, 162, 164, 169, 170, 171, 172, 173, 181, 183

Entry_Barrier, 132, 133
entry_call_alternative, 43
entry_call_statement, 189, 190
entry_declaration, 132
Entry_Family_Definition, 132
Entry_Index_Specification, 119, 132, 133
enumeration_aggregate, 112
Enumeration_Literal_Declarations, 144
enumeration_type_definition, 144
environment, 1, 235, 262
Environment_Error, 8, 51, 54
ER, 3
Error_Kinds, 4, 51, 54, 55, 239, 242
Errors, xv, 2, 20, 21, 51, 53, 54, 231, 237, 239, 240, 244
Exception_Choices, 194
exception_handler, 122, 186
Exception_Handler, 30, 194, 195
Exception_Handler_List, 30, 122, 186
Exceptions, xv, 2, 8, 20, 21, 51, 52, 237, 239, 240, 242
Exists, 63, 79
Exit_Condition, 187
Expected_Enclosing_Element, 103, 104
explicit declarations, 264
Explicit element, 235
explicit_actual_parameter, 171, 176, 190
explicit_generic_actual_parameter, 137, 171
exponent, 160
expression, 12, 39, 128, 151, 152, 156, 159, 165, 166, 167, 168, 173, 174, 176, 177, 178, 181, 182, 183, 184, 187, 188, 190, 191, 193, 194, 195, 198, 199
Expression, 14, 30, 99, 115, 116, 120, 130, 133, 137, 138, 145, 146, 149, 150, 151, 159, 160, 162, 164, 165, 166, 167, 168, 170, 171, 173, 174, 175, 176, 177, 178, 179, 181, 182, 183, 184, 186, 187, 188, 191, 193, 195, 198, 199
Expression_Kind, 99, 241
Expression_Kinds, 31, 39, 94, 99, 167
Expression_List, 30, 147, 165, 167, 169, 172, 194
Expression_Parenthesized, 174
expressions, 264, 265
Expressions, xv, 2, 13, 14, 20, 21, 32, 159, 240, 241
Extension, 235
extension_aggregate, 167, 168
Extension_Aggregate_Expression, 168

F

Family, 49, 235
Finalize, 21, 54, 238, 242
first_bit, 199
First_Bit, 227, 229
First_Column, 202, 205
First_Line, 202, 205, 206
First_Line_Number, 15, 202, 241
For_Loop_Parameter_Specification, 184
Formal_Parameter, 56, 170
formal_parameter_selector_name, 57, 170, 176, 177, 190
formal_part, 176, 190
Formal_Subprogram_Default, 138
formal_type_declaration, 113
Formal_Type_Definition, 30, 97, 114
Formal_Type_Kind, 97
Formal_Type_Kinds, 31, 37, 94, 96, 97
full_type_declaration, 33, 113, 143
function_call, 165, 175
Function_Call_Parameters, 57, 109, 173, 174, 176
Function_Call_Parameters_Normalized, 57, 177
function_prefix, 175

G

Generic_Actual_Part, 56, 109, 136, 137, 139, 174
Generic_Actual_Part_Normalized, 56, 138
generic_association, 137, 171
Generic_Formal_Parameter, 30, 96, 138
generic_formal_parameter_declaration, 137, 170
Generic_Formal_Parameter_List, 30
generic_formal_parameter_selector_name, 56, 137, 138, 170
generic_formal_part, 137
Generic_Formal_Part, 135, 136
generic_instantiation, 137
Generic_Macro_Expansion_Supported, 59
Generic_Unit_Name, 136, 137, 139
goto, 188
Goto_Label, 188
guard, 43, 44, 193
Guard, 193

H

Handler, 195
Handler_Statements, 195
Has_Associations, 60, 63
Has_Attribute, 58, 81
Hash, 16, 107, 209

I

Id, 16, 20, 209, 210, 211, 235, 264
Id_Io, 247
identifier, 161, 166, 183, 200
Identifier, 30
Identifier_List, 30
IDL, 250
Ids, 2, 16, 20, 21, 209, 247, 266
  <, 209
  >, 209
if_statement, 43
Implementation, xiv, 2, 19, 21, 24, 51, 53, 56, 237, 238, 239, 240, 242
Implementation_Defined, 27, 50, 68, 208, 211, 217
Implementation_Defined_Integer_Constant, 27, 28, 201, 217

Implementation_Defined_Integer_Type, 27, 28
Implementation_Status, 4
Implementor, 235
implicit declarations, 264
Implicit element, 236
Implicit_Components, 154, 155, 156
Implicit_Components_Supported, 58, 155
Implicit_Inherited_Declarations, 141
Implicit_Inherited_Subprograms, 142
Include_Pragmas, 92, 105, 121, 122, 128, 129, 131, 135, 154, 156, 157, 182, 185, 186, 192, 195, 198, 199
Inconsistent_Length, 86, 88
Index, 224, 227, 228
index_constraint, 152
Index_Expressions, 165
index_subtype_definition, 147
Index_Subtype_Definitions, 146, 147, 158
Index_Type, 232
Index_Type_1, 232
Index_Type_2, 232
Index_Type_3, 232
indexed_component, 165
Indexes, 224, 227, 228
Inherited_Declarations_Supported, 59
Inherited_Subprograms_Supported, 59
Initialization_Error, 8, 51
Initialization_Expression, 14, 16, 114, 115, 116, 121, 138
Initialize, 8, 21, 53, 238, 242
Integer_Constraint, 144, 145
Internal_Error, 8, 51, 54, 55
IP, 3
IR, 3
Is_Array, 219, 223, 225, 226
Is_Body_Required, 79
Is_Call_On_Dispatching_Operation, 196
Is_Commentary_Supported, 58
Is_Declare_Block, 185
Is_Defaulted_Association, 138, 172, 173, 174, 176, 190
Is_Dispatching_Call, 196
Is_Dispatching_Operation, 139
Is_Equal, 62, 67, 68, 70, 73, 74, 75, 76, 77, 86, 101, 124, 125, 138, 152, 168, 176, 190, 204, 210, 218, 222
Is_Finalized, 54
Is_Formal_Parameter_Named_Notation_Supported, 56
Is_Identical, 62, 65, 66, 67, 68, 70, 71, 72, 73, 74, 77, 78, 82, 86, 89, 101, 117, 118, 124, 125, 204, 210, 219
Is_Initialized, 24, 53
Is_Line_Number_Supported, 58
Is_Name_Repeated, 123, 184
Is_Nil, 16, 77, 101, 164, 203, 204, 205, 208, 210, 218, 238, 241
Is_Normalized, 56, 57, 109, 138, 152, 168, 169, 170, 171, 172, 173, 176, 177, 190
Is_Open, 63
Is_Part_Of_Implicit, 102, 119, 122, 128, 138, 140, 152, 161, 162, 168, 176, 190, 208
Is_Part_Of_Inherited, 103, 127
Is_Part_Of_Instance, 102, 103, 124, 208
Is_Prefix_Call, 175
Is_Prefix_Call_Supported, 57
Is_Private_Present, 129, 158
Is_Record, 219, 221, 222
Is_Referenced, 161
Is_Span_Column_Position_Supported, 58
Is_Subunit, 134, 135
Is_Text_Available, 202, 203, 206, 208
issues, 262
Iterator, xiv, 2, 19, 21, 49, 108, 220, 224, 227, 228, 237, 240

K

known_discriminant_part, 33, 152, 154, 157

L

label, 188
Label_Names, 181
label_statement_identifier, 111, 181
last_bit, 199
Last_Bit, 228, 229
Last_Column, 202, 205
Last_Line, 16, 202, 205, 206
Last_Line_Number, 16, 202
Length, 167, 203, 204, 207, 226
library_item, 44, 74, 76
Library_Item, 74, 75, 76
Library_Unit, 72, 73, 74
library_unit_body, 46, 47, 48, 67, 71, 72, 74, 75, 76, 77, 79, 82
library_unit_declaration, 46, 47, 48, 66, 71, 74, 75, 76, 79
Library_Unit_Declaration, 71, 238
Library_Unit_Declarations, 66, 71
library_unit_renaming_declaration, 47, 66, 71, 74, 76, 79
Line, 20, 25, 52, 200, 203, 204, 207, 208, 236, 264
Line_Image, 207
Line_List, 200, 201, 203, 204, 205, 206, 208
Line_Number, 52, 200, 201, 202, 206
Line_Number_Positive, 201, 202, 206
Lines, 16, 111, 200, 204, 205, 206, 207
List_Index, 27, 28, 30, 45, 65, 214, 215
List_Index_Implementation_Upper, 28
Literals, 32
loop_parameter_specification, 119
loop_statement, 111, 183, 185
Loop_Statements, 185
Lower_Bound, 150, 153
LRM-Interface, 257

M

Maximum_Line_Length, 201
Maximum_Line_Number, 200, 201
Membership_Test_Expression, 177
Membership_Test_Range, 178
Membership_Test_Subtype_Mark, 178
Missing_Length, 86, 88
mod, 140
mod_clause, 198
Mod_Clause_Expression, 198
Mod_Static_Expression, 145
Mode_Kind, 96
Mode_Kinds, 31, 33, 34, 36, 94, 96
modular_type_definition, 145

N

name, 3, 16, 21, 29, 60, 79, 160, 165, 166, 175, 176, 184, 186, 191, 193, 197
Name, 21, 28, 29, 30, 60, 61, 62, 63, 65, 68, 71, 77, 112, 115, 155, 160, 161, 162, 164, 183, 186, 191, 193, 197
Name_Error, 8, 51, 61
Name_Image, 160
Name_List, 30, 161, 197, 238
named_array_aggregate, 169
names, 265
Names, 14, 61, 110, 113, 114, 115, 116, 119, 120, 121, 122, 128, 129, 130, 131, 132, 133, 135, 136, 138, 139, 197, 238, 241
Needed Units, 236
Needed_Units, 49
Next, 216, 220
Nil_Array_Component, 215, 230
Nil_Array_Component_Iterator, 216, 220, 224, 230
Nil_ASIS_Time, 84, 85
Nil_Compilation_Unit, 25, 45, 50, 69, 70, 71, 73, 75, 76, 77, 78, 81, 82, 85, 87
Nil_Compilation_Unit_List, 45, 50, 66, 67, 71, 72, 82, 88
Nil_Container, 65, 66, 68
Nil_Context, 29, 50
Nil_Element, 16, 29, 32, 44, 50, 56, 57, 91, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 109, 113, 114, 115, 117, 124, 125, 126, 128, 132, 133, 134, 135, 139, 146, 150, 153, 162, 170, 175, 183, 187, 189, 191, 194, 195, 198, 210
Nil_Element_List, 30, 50, 92, 93, 105, 106, 110, 119, 120, 121, 122, 132, 137, 140, 142, 148, 152, 155, 157, 158, 167, 168, 169, 172, 176, 181, 185, 186, 190, 191, 199
Nil_Id, 209, 210, 211
Nil_Line, 200, 203, 208
Nil_Line_List, 201, 208
Nil_Portable_Data, 217, 229
Nil_Record_Component, 214, 230
Nil_Relationship, 88
Nil_Span, 202, 203, 204, 205, 206
Non_Comment_Image, 207
Normalized, 137, 152, 190
not, 140
Not_A_Class, 47, 69
Not_A_Clause, 44, 100
Not_A_Constraint, 39, 98, 146
Not_A_Declaration, 33, 91, 95, 96, 116, 117, 119, 125, 126, 133, 134, 135, 175, 189, 194, 238
Not_A_Declaration_Origin, 36, 96
Not_A_Default, 36
Not_A_Defining_Name, 33, 95, 184
Not_A_Definition, 36, 97, 113, 114, 132, 150
Not_A_Discrete_Range, 39, 98
Not_A_Formal_Type_Definition, 37, 97
Not_A_Mode, 36, 96
Not_A_Path, 43, 100
Not_A_Pragma, 32, 94
Not_A_Representation_Clause, 44, 101
Not_A_Root_Type_Definition, 38, 98
Not_A_Statement, 42, 100
Not_A_Trait, 35, 95
Not_A_Type_Definition, 37, 97
Not_A_Type_Model, 218
Not_A_Unit, 45, 69, 70, 74, 75, 76, 79, 91, 92, 93
Not_An_Access_Type_Definition, 38, 97
Not_An_Association, 39, 100
Not_An_Attribute, 41, 99
Not_An_Element, 20, 32, 94, 115, 116, 127, 139, 160, 171, 187, 191, 193, 198
Not_An_Error, 8, 51, 54, 55
Not_An_Expression, 39, 99, 187, 195
Not_An_Operator, 40, 99
Not_An_Origin, 47, 70
Not_Implemented_Error, 4, 8
Not_Implemented_Error,, 51

O

object_declaration, 114
Object_Declaration_View, 14, 114, 115, 116, 131
Object_Declarations_Normalized, 58
Object_Form, 80
Object_Name, 80
Obsolete_Reference_Error, 8, 25, 51, 161, 162
Open, 19, 21, 60, 61, 63, 80, 160, 238, 242
Operator_Kind, 99
Operator_Kinds, 31, 40, 94, 99
operator_symbol, 165
Optional functionality, 236
or, 140

P

package_body, 186
Package_Body_Block, 122
package_declaration, 79
parameter_and_result_profile, 176, 190
parameter_association, 176, 190
Parameter_Error, 8, 51, 54, 60, 61

parameter_profile, 148, 190
Parameter_Profile, 119, 120, 121, 122, 130, 132, 133, 135, 136, 138
parameter_specification, 148, 170, 176, 190
Parameter_Specification, 30, 162
Parameter_Specification_List, 30, 120, 148, 191
Parameters, 28, 29, 53, 54, 60, 61, 64
Parent_Body, 82
parent_subtype_indication, 141, 143
Parent_Subtype_Indication, 141
parent_unit_name, 47, 73, 78
parser, 255
Path, 30, 100, 182, 183, 193
Path_Kind, 100
Path_Kinds, 31, 43, 94, 100
Path_List, 30, 182
Permissions, 2, 19, 56
Portable_Array_Type_1, 20, 229, 232
Portable_Array_Type_2, 20, 229, 232
Portable_Array_Type_3, 20, 229, 232
Portable_Constrained_Subtype, 20, 228, 231, 232, 233
Portable_Data, 212, 216, 217, 222, 224, 226, 228, 229, 231, 232
Portable_Data_Io, 244
Portable_Positive, 217
Portable_Transfer, 3, 21, 212, 231, 244
Portable_Transfer, 2, 20, 212
Portable_Unconstrained_Record_Type, 20, 231
Portable_Value, 217
Position, 227
Position_Number_Image, 111, 112
positional_array_aggregate, 168, 169
Post_Operation, 108
pragma, 11, 44, 90, 104, 115, 121, 122, 163, 188, 199
pragma_argument_association, 163
Pragma_Argument_Associations, 106
pragma_argument_identifier, 170
Pragma_Element, 30, 94, 106
Pragma_Element_List, 30, 92, 93, 105, 106
Pragma_Kind, 94
Pragma_Kinds, 31, 32, 94
Pragma_Name_Image, 106
pragmas, 265
Pragmas, 44, 105, 182
Pre_Operation, 108
Predefined_Operations_Supported, 59
prefix, 57, 78, 111, 112, 165, 166, 175
Prefix, 164, 165, 166
primary, 174
private_extension_declaration, 157
Private_Part_Declarative_Items, 128, 129, 136
Private_Part_Items, 157
private_type_declaration, 113
procedure_call_statement, 189, 190
procedure_prefix, 189
Program_Text, 19, 49, 106, 111, 160, 206, 207
proper_body, 47, 134
protected_body, 132
protected_definition, 113, 114
protected_operation_item, 132
Protected_Operation_Items, 131

Q

Qualified_Expression, 195
queries, 12

R

Raised_Exception, 195
range, 156, 166
Range_Attribute, 151, 153
range_attribute_reference, 151
range_constraint, 145, 146, 150, 151
Range_Constraint, 30, 145, 146, 150, 151, 178
Real_Range_Constraint, 145, 146, 151
real_range_specification, 146
record_aggregate, 167
Record_Component, 30, 155, 156, 212, 214, 218, 219, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 264
record_component_association, 167, 170
Record_Component_Associations, 109, 167, 173, 174
Record_Component_Associations_Normalized, 56, 168
Record_Component_Choices, 169
Record_Component_List, 30, 154, 155, 214, 221, 222
Record_Components, 154, 156, 212, 214, 215, 221, 222
record_definition, 154, 155
Record_Definition, 30, 141, 148
record_representation_clause, 199
record_type_definition, 35
Reference, 139, 164
References, 161
relation, 48, 82, 88
Relation, 88, 236
Relation_Kinds, 47, 72, 88, 236
Relations, 2, 19, 21, 86
Relationship, 86, 88, 89
rem, 40, 140
Renamed_Entity, 130
renaming_declaration, 71, 131
representation_clause, 112, 119, 185, 197, 198
Representation_Clause, 30, 100, 198
Representation_Clause_Expression, 197, 198
Representation_Clause_Kind, 100
Representation_Clause_Kinds, 31, 44, 94, 100
Representation_Clause_List, 30, 119
Representation_Clause_Name, 197, 198, 199
Representation_Value_Image, 111, 112
requeue, 193
Requeue_Entry_Name, 193
Required functionality, 236
Reset, 220
Result_Profile, 120, 122, 130, 133, 136, 138
Return_Expression, 187
Root_Type_Definition, 30, 98

Root_Type_Kind, 98
Root_Type_Kinds, 31, 37, 38, 94, 98

S

select_alternative, 43, 44, 265
Select_Alternative, 30
select_statement, 43
selected_component, 165, 166
selector, 111
Selector, 166
selector_name, 167, 176
Semantic_Dependence_Order, 26, 82, 86, 88, 89
sequence_of_statements, 43, 44, 185, 186
Sequence_Of_Statements, 182
Set_Status, 55
Short_Circuit_Operation_Left_Expression, 177
Short_Circuit_Operation_Right_Expression, 177
signed_integer_type_definition, 145
simple_expression, 150, 151
single_protected_declaration, 114
single_task_declaration, 114
Size, 212, 217, 226, 227, 229, 231, 232
slice, 165, 166
Slice_Range, 165, 166
Span, 201, 202, 203, 204, 205, 206, 207
Specification, 119
Specification_Subtype_Definition, 119
Standard.Duration, 84
State, 108
State_Information, 108
statement, 14, 25, 181, 182, 183, 184, 185, 186, 187, 188, 189, 191, 192, 193, 194, 195
Statement, 30, 100, 122, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195
Statement_Kind, 14, 100, 184, 241
Statement_Kinds, 31, 42, 94, 100, 184
Statement_List, 30, 105, 121, 182, 185, 186, 192, 195
Statement_Paths, 182
Statements, xv, 2, 13, 20, 21, 32, 181, 240, 241
static_expression, 145, 146, 167, 198
Status, 8, 24, 25, 51, 54, 55, 60, 61, 115, 159, 161, 162, 163, 200, 206, 210, 214, 220, 224, 227, 228, 229, 231, 239, 242, 244
Storage_Error, 4, 8, 51, 61
subcomponent, 103
subprogram_body, 47
Subprogram_Body_Block, 122
subprogram_declaration, 127
Subprogram_Default_Kinds, 31, 34, 36, 94, 96
subprogram_specification, 36
Subtype_Constraint, 57, 149, 150, 153
subtype_declaration, 117, 118
subtype_indication, 38, 57, 114, 148, 150, 153
Subtype_Indication, 30, 141, 148, 153, 157, 179, 225
subtype_mark, 116, 149, 174, 178
Subtype_Mark, 30, 149, 150, 153, 158
subunit, 25, 44, 46, 47, 48, 67, 69, 71, 72, 124, 134, 135
Subunit, 82, 134
Subunits, 25, 82
Supporter, 236
Supporters, 48
System.Storage_Unit, 227

T

tagged type, 35, 139, 189, 196
Task_Body_Block, 122
task_definition, 113, 114, 158
task_type_declaration, 113
terminate_alternative, 43
Terminate_Immediately, 49, 109
Text, xv, 2, 15, 16, 20, 21, 25, 64, 107, 111, 200, 208, 211, 240, 241
Text_Error, 8, 25, 51, 200
Text_Form, 80
Text_Name, 11, 79, 80
The_Container, 66, 67, 68
The_Context, 60, 61, 63, 64, 65, 71, 72, 73, 74, 75, 76, 82, 83, 88, 89, 92, 117, 123, 124, 125, 133, 134, 210
The_Element, 105
The_Id, 209, 210, 211
The_Line, 204, 207, 208
The_Span, 205
Time, 84, 85
Time_Of, 84
Time_Of_Last_Update, 11, 84
timed_entry_call, 193
Times, 19
Times, 2, 21, 84
Trait_Kind, 35, 95
Trait_Kinds, 31, 33, 35, 36, 37, 94, 95
Traverse_Control, 19, 27, 49, 108, 237, 240, 241, 242
Traverse_Element, 19, 108, 123, 237, 240
triggering_alternative, 43
type_conversion, 174
type_declaration, 113, 117, 118, 144, 159
Type_Declaration_View, 110, 113, 114, 129, 131, 138
type_definition, 33, 113, 141, 142, 157
Type_Definition, 30, 97, 114, 127, 140, 141, 143, 144, 145, 147, 148, 149, 157, 218, 221, 222, 223, 225, 226, 229
Type_Kind, 97, 110
Type_Kinds, 31, 36, 37, 94, 96, 97
Type_Model_Kind, 218
Type_Model_Kinds, 218, 221

U

Unchecked_Conversion, 217
unconstrained_array_definition, 147
Unconstrained_Record_Type, 231
Unhandled_Exception_Error, 8, 51
Unique_Name, 78
Unit Kinds, 27, 45
Unit_Class, 69

Unit_Classes, 47, 69
Unit_Declaration, 21, 91, 104, 123, 125, 134, 238, 242
Unit_Full_Name, 78, 241, 242
Unit_Kind, 69, 70, 77, 87, 92, 134
Unit_Kinds, 45, 46, 69, 70, 75, 79, 81
Unit_Origin, 70, 242
Unit_Origins, 47, 70
universal_integer, 111
Upper_Bound, 150, 151, 153
use_clause, 185, 197
Use_Error, 8, 51, 61

V

Value, 229
Value_Error, 8, 24, 25, 51, 115, 159, 163, 210, 224
Value_Image, 160
variant, 154, 155, 156, 222
Variant, 30, 156
Variant_Choices, 156
Variant_Component_List, 30
Variant_List, 30, 156
variant_part, 155, 156
Variant_Part, 155, 156
Variants, 155, 156
Visible_Part_Declarative_Items, 128, 136
Visible_Part_Items, 157

W

While_Condition, 184
Wide_String, 49, 53, 54, 55, 60, 63, 64, 68, 71, 78, 79, 80, 81, 83, 85, 107, 112, 160, 206, 208, 211
with_clause, 197
Within_Element, 161

X

xor, 40, 140