TASK: UQ19
CDRL:251901
21 Oct 1991

UQ19 Ada Semantic Interface Specification (ASIS) Version 0.4 – Vendor Independent ASIS (VIA)

Informal Technical Data

STARS-RC-251901/003/00
21 October 1991

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	21 October 1991	Informal Technical Data

4. TITLE AND SUBTITLE

UQ19 Ada Semantic Interface Specification(ASIS) Version 0.4 - Vendor Independent ASIS (VIA)

5. FUNDING NUMBERS

STARS Contract
F19628-88-D-0031

6. AUTHOR(S)

Unisys Defense Systems, Inc.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Unisys Corporation
12010 Sunrise Valley Drive
Reston, VA 22091

8. PERFORMING ORGANIZATION REPORT NUMBER

STARS-RC-251901/003/00

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Department of the Air Force
Headquarters, Electronic Systems Division (AFSC)
Hanscom AFB, MA 01731-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

251901

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release;
distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Ada Semantic Interface Specification is layered vendor-independent open architecture. ASIS queries and services provide a consistent interface to information within the Ada Program Library. Clients of ASIS are shielded and free from the implementation details of each Ada vendor's proprietary library and intermediate representations.

14. SUBJECT TERMS

ASIS, Ada Program Library, Open Architecture.

15. NUMBER OF PAGES
160

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

TASK: UQ19
CDRL: 251901
21 October 1991

INFORMAL TECHNICAL REPORT

For The

SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS (STARS)

Ada Semantic Interface Specification (ASIS)
Version 0.4
Vendor Independent ASIS (VIA)

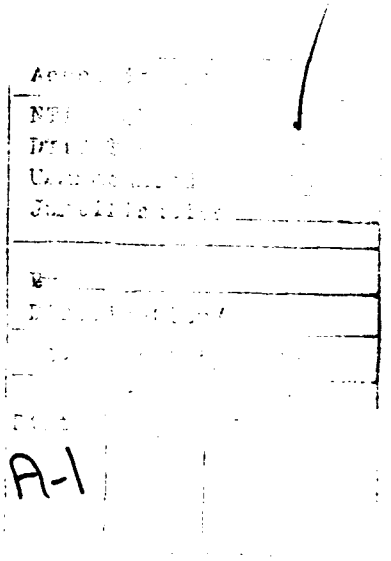
STARS-RC-251901/003/00
Publication No. GR-7670-1344(NP)
21 October 1991

Data Type: A005, Informal Technical Data

CONTRACT NO. F19628-88-D-0031
Delivery Order 0002

Prepared for:

Electronic Systems Division
Air Force Systems Command, USAF
Hanscom AFB, MA 01731-5000



Prepared by:

Telesoft
under contract to
Unisys Defense Systems, Inc.
Tactical Systems Division
12010 Sunrise Valley Drive
Reston, VA 22091

TASK: UQ19
CDRL: 251901
21 October 1991

INFORMAL TECHNICAL REPORT

For The

SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS
(STARS)

Ada Semantic Interface Specification (ASIS)
Version 0.4
Vendor Independent ASIS (VIA)

STARS-RC-251901/003/00
Publication No. GR-7670-1344(NP)
21 October 1991

Data Type: A005, Informal Technical Data

CONTRACT NO. F19628-88-D-0031
Delivery Order 0002

Prepared for:

Electronic Systems Division
Air Force Systems Command, USAF
Hanscom AFB, MA 01731-5000

Prepared by:

Telesoft
under contract to
Unisys Defense Systems, Inc.
Tactical Systems Division
12010 Sunrise Valley Drive
Reston, VA 22091

TASK: UQ19
CDRL: 251901
21 October 1991

Data ID: STARS-RC-251901/003/00

Distribution Statement "A"
per DoD Directive 5230.24
Authorized for public release; Distribution is unlimited.

Developed by: Telesoft under contract to
Unisys Defense Systems, Inc.

This software, developed under the Software Technology for Adaptable, Reliable Systems (STARS) program, is approved for release under Distribution "A" of the Scientific and Technical Information Program Classification Scheme (DoD Directive 5230.24) unless otherwise indicated. Sponsored by the U.S. Defense Advanced Research Projects Agency (DARPA) under contract F19628-88-D-0031, the STARS program is supported by the military services, SEI, and MITRE, with the U.S. Air Force as the executive contracting agent.

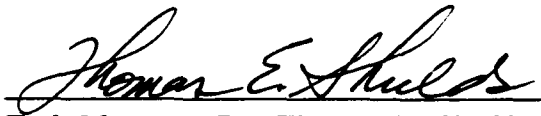
TASK: UQ19
CDRL: 251901
21 October 1991

INFORMAL TECHNICAL REPORT
Ada Semantic Interface Specification (ASIS)
Version 0.4
Vendor Independent ASIS (VIA)

Principal Author(s):

Steve Blake, TeleSoft *Date*

Approvals:



10/8/91

Task Manager *Dr. Thomas E. Shields* *Date*

(Signatures on File)

TASK: UQ19
CDRL: 251901
21 October 1991

INFORMAL TECHNICAL REPORT
Ada Semantic Interface Specification (ASIS)
Version 0.4
Vendor Independent ASIS (VIA)

Change Record:

Data ID	*Description of Change*	*Date*	*Approval*
STARS-RC-251901/003/00	Based on ASIS WG meeting of June 6, 1991.	21 October 1991	*on file*
STARS-RC-251901/002/00	Successor Volume: Integrates Rational's LRM interface, eliminating dependencies on the Rational environment.	28 May 1991	*on file*
STARS-RC-251901/001/00	Original Issue	11 February 1991	*on file*

Contents

1 SPECIFICATION LISTING

```
--|
--|Item: Ada Semantic Interface Specification (ASIS)
--|$Revision:"0.4"$
--|$State: Prototype $
--|
--|Distribution Statement "A", per DoD Directive 5230.24
--|Authorized for public release; distribution is unlimited.
--|
--|Copyright (c) 1991, Unisys Defense Systems, Inc., Reston, Virginia
--|                    and Telesoft, San Diego, California
--|Copyright is assigned to the U.S. Government, upon delivery thereto,
--|in accordance with the DFAR Special Works Clause. However, Copyright
--|in computer software developed by Rational and from which ASIS is
--|derived in part is not assigned hereby; such software includes
--|Ada_Program, Compilation_Units, Declarations, Names_And_Expressions,
--|Pragmas, Representation_Clauses, Statements, and Type_Information, known
--|collectively as the Rational LRM Interfaces. Rational LRM Interfaces
--|Copyright (c) 1987, 1988, 1989, 1990, 1991, Rational, Santa Clara,
--|California.
--|
--|Developed by:  Telesoft under contract to Unisys Defense Systems, Inc.
--|
--|This software, developed under the Software Technology for Adaptable,
--|Reliable Systems (STARS) program, is approved for release under
--|Distribution "A" of the Scientific and Technical Information Program
--|Classification Scheme (DoD Directive 5230.24) unless otherwise indicated.
--|Sponsored by the U.S. Defense Advanced Research Projects Agency (DARPA)
--|under contract F19628-88-D-0031, the STARS program is supported by the
--|military services, SEI, and MITRE, with the U.S. Air Force as the executive
--|contracting agent.
--|
--|Permission to use, copy, modify, and comment on this software and its
--|documentation for purposes stated under Distribution "A" and without fee
--|is hereby granted, provided that this notice appears in each whole or
--|partial copy.  This software retains Contractor indemnification to
--|the Government regarding copyrights pursuant to the above referenced
--|STARS contract.  The Government disclaims all responsibility against
--|liability, including costs and expenses for violation of proprietary
--|rights, or copyrights arising out of the creation or use of this
--|software.
--|
--|In addition, the Government, Unisys, and its subcontractors disclaim all
--|warranties with regard to this software, including all implied warranties
```

```
--|of merchantability and fitness, and in no event shall the Government,
--|Unisys, or its subcontractor(s) be liable for any special, indirect or
--|consequential damages or any damages whatsoever resulting from the loss of
--|use, data, or profits, whether in action of contract, negligence or other
--|tortious action, arising in connection with the use or performance of this
--|software.
--|


--|$Log:"ASIS_spec.ps,v"$"--|Revision"0.4""1991/11/08""21:24:04""cm"--|Based"on"ASIS"WG"meeting"Ju
--|Revision 0.4  1991/11/08  20:56:33  cm
--|Based on ASIS WG meeting June 6, 1991
--|
--|Revision 0.4  1991/10/21  17:01:39  cm
--|Based on ASIS WG meeting June 6, 1991.
--|
--|Revision 0.3  1991/05/30  15:16:14  guy
--|Integrates Rational's LRM interface, eliminating dependencies on the
--|Rational environment.
--|
--|Revision 0.2  1991/01/17  08:58:49  cm
--|Initial submittal.
--|



------------------ASIS-PROLOGUE--------------------------------
--
-- System       : ADA SEMANTIC INTERFACE SPECIFICATION (ASIS)
-- Version      : 0.4
--              :
-- Components   : Specifications are in the following order:
--              :
--              :
--              :   ASIS_Ada_Program
--              :   ASIS_Environment
--              :   ASIS_Libraries
--              :   ASIS_Compilation_Units
--              :   ASIS_Elements
--              :   ASIS_Declarations
--              :   ASIS_Type_Definitions
--              :   ASIS_Names_And_Expressions
--              :   ASIS_Statements
--              :   ASIS_Representation_Clauses
--              :   ASIS_Text
```

```
--              :
--              :
-- Release date : Sept 27, 1991
-- Last update  : July 10, 1991
-- Date created : June 01, 1990
-- Machine/System Compiled/Run on : Sun 4, SunOS Release 4.1.1
--
------------------------------------------------------------------
--
-- Abstract     :
--
-- The Ada Semantic Interface Specification is a layered
-- vendor-independent open architecture. ASIS queries and
-- services provide a consistent interface to information
-- within the Ada Program Library.  Clients of ASIS are shielded
-- and free from the implementation details of each Ada vendor's
-- proprietary library and intermediate representations.
--



------------------ Revision history --------------------------
--
-- DATE         VERSION AUTHOR          HISTORY
-- 12/28/90     0.1     Steve Blake     Initial Release
-- 01/11/91     0.2     Steve Blake     Unisys/STARS Phase I deliverable
-- 05/13/91     0.3     Steve Blake     Unisys/STARS Phase II deliverable
-- 07/10/91     0.4*    Steve Blake     Internal reveiw release
-- 09/27/91     0.4     Steve Blake     Unisys/STARS Phase II deliverable
--
------------------ Distribution and Copyright ----------------
--
-- This prologue must be included in all copies of this software.
--
-- The ASIS version 0.4 specification has been developed by
-- TeleSoft under STARS subcontract to Unisys.  This information
-- is only releasable to STARS contractors and STARS subcontractors
-- until it has been approved for wider release by the ESD STARS
-- program office.
--
-- Version 0.4 of the ASIS specification is being distributed to
-- STARS contractors with the intention that they review the
-- specification for its applicability to Ada tools, its completeness,
-- and its relevance.  The ASIS working group has been established to
-- review comments and account for them in future versions of ASIS.
```

```
-- Please forward all comments to the members of the working group at
-- the following e-mail addresses:
--
--       stars_asis@stars.reston.unisys.com
--
--
------------------ Disclaimer ---------------------------------
--
-- This software and its documentation are provided "AS IS" and
-- without any expressed or implied warranties whatsoever.
-- No warranties as to performance, merchantability, or fitness
-- for a particular purpose exist.
--
-- Because of the diversity of conditions and hardware under
-- which this software may be used, no warranty of fitness for
-- a particular purpose is offered.  The user is advised to
-- test the software thoroughly before relying on it.  The user
-- must assume the entire risk and liability of using this
-- software.
--
-- In no event shall any person or organization of people be
-- held responsible for any direct, indirect, consequential
-- or inconsequential damages or lost profits.
--
-------------------END-PROLOGUE--------------------------------




-----------------------------------------------------------------------
-- ASIS PRELIMINARY REVIEW VERSION 0.4
-- Sept 27, 1991
-----------------------------------------------------------------------


package ASIS_Ada_Program is

  ---------------------------------------------------------------------
  -- ASIS_Ada_Program encapsulates declarations that are made
  -- available to ASIS and its clients in a vendor-independent manner.


  ---------------------------------------------------------------------
  -- LIBRARY
```

```
subtype Library is implementation_defined_limited_private_type;

-- The Ada Program Library abstraction (a limited private type).
--
-- A logical library handle is associated with name and parameters values
-- which are used by the implementation to identify and connect to the
-- physical information in the "library file".
--
-- The term "library file" refers to the information on the compilation
-- units that must be maintained by the compiler or compiling environment.
-- It does not assume the method a vendor chooses to implement or maintain
-- the information.
--
-- Some environments may not need name and parameters values to identify
-- their library file.  Other environments may choose to implement the
-- library as a single external file in which case the name and parameters
-- values would supply the name and form values needed to open such a file.


-------------------------------------------------------------------------
-- COMPILATION UNITS

subtype Compilation_Unit is implementation_defined_private_type;

-- The Ada Compilation Unit abstraction (a private type).
--
-- The text of a program can be submitted to the compiler in one or
-- more compilations.  Each compilation is a succession of compilation
-- units.
--
-- Compilation units are composed of two distinct parts:
--  1) A context clause
--  2) The declaration of a library unit or a secondary unit.
--
-- The context clause contains zero or more with clauses and use clauses.
--
-- ASIS treats Pragmas that appear immediately after the context clause
-- and before the subsequent declaration part as belonging to the context
-- clause part.
--
-- The declaration associated with a compilation unit can either
-- be a package, procedure, function, generic, or subunit.
--
-- The abstract type Compilation_Unit is is a handle to 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.
--

Nil_Compilation_Unit: Compilation_Unit renames
   implementation_defined_constant;

-- A constant Ada Compilation Unit with a void or null value.


-------------------------------------------------------------------------
-- ELEMENTS

subtype Element is implementation_defined_private_type;

-- The Ada program element abstraction (a private type).
--
-- The type Element is a distinct abstract type representing handles to the
-- lexical elements that form the text of compilation units.  Elements deal
-- with the internal or "textual" view of compilation units.
--
-- Operations are provided to split a Compilation_Unit object into two
-- component Elements:
--  1) A context clause represented by an Element_List containing
--     with clauses, use clauses, and pragmas.
--  2) An Element associated with the declaration.
--
-- Use the context clause operations in this package to decompose
-- with clauses and use clauses.
--
-- Use the ASIS_ELEMENTS pragma operations to decompose pragmas.
--

Nil_Element: Element renames
   implementation_defined_constant;

-- A constant Ada program element with a void or null value.


-------------------------------------------------------------------------
-- ELEMENT SUBTYPES

subtype Declaration is ASIS_Ada_Program.Element;
-- Use ASIS_Declarations operations.
```

```
subtype Type_Definition is ASIS_Ada_Program.Element;
-- Use ASIS_Type_Definitions operations.

subtype Identifier_Definition is ASIS_Ada_Program.Element;

subtype Identifier_Reference is ASIS_Ada_Program.Element;

subtype Expression is ASIS_Ada_Program.Element;
-- Use ASIS_Names_And_Expression operations.

subtype Name_Expression is ASIS_Ada_Program.Element;
-- Use ASIS_Names_And_Expression operations.

subtype Statement is ASIS_Ada_Program.Element;
-- Use ASIS_Statements operations.



-------------------------------------------------------------------------
-- TIME

subtype ASIS_Time is Calendar.Time;

-- A private type from the predefined library package Calendar.
-- Operations on ASIS_Time are provided by Calendar.


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


-------------------------------------------------------------------------
-- ASIS_INTEGER

subtype ASIS_Integer is Integer range
  implementation_defined .. implementation_defined;

-- A subtype of the predefined type Integer that allows each ASIS
-- implementation to place constraints on the lower and upper bounds.

-------------------------------------------------------------------------
-- ASIS_POSITIVE

subtype ASIS_Positive is ASIS_Integer range 0 .. implementation_defined;
```

```
-- A subtype of the predefined type Integer that allows each ASIS
-- implementation to place a constraint on the upper bound.

-------------------------------------------------------------------------
-- ASIS_LIST_INDEX

subtype ASIS_List_Index is ASIS_Integer range 1 .. implementation_defined;

-- A subtype of the predefined type Integer that allows each ASIS
-- implementation to place a constraint on the upper bound on the number
-- of components allowed in lists.
--
-- Used as a subtype for the array index subtype definition of the various
-- lists in ASIS.


-------------------------------------------------------------------------
-- COMPILATION_UNIT_LIST

subtype Compilation_Unit_List is implementation_defined_array;

-- Defines a list of Ada compilation units.  For some operations that
-- return a Compilation_Unit_List value, the order of components may vary
-- across implementations of ASIS.  Variances are noted in the commentary
-- of individual operations.
--
-- Implemented as an unconstrained array:
-- array ( ASIS_List_Index range <> ) of Compilation_Unit;


Nil_Compilation_Unit_List : Compilation_Unit_List  renames
   implementation_defined_constant;

-- A constant Ada Compilation Unit List with a void or null value.


package Compilation_Unit_List_Operations is

  function "&" (Left  : in Compilation_Unit_List;
                Right : in Compilation_Unit_List)
    return Compilation_Unit_List renames implementation_defined;

  function "&" (Left  : in Compilation_Unit_List;
                Right : in Compilation_Unit)
    return Compilation_Unit_List renames implementation_defined;
```

```
  function "&" (Left  : in Compilation_Unit;
                Right : in Compilation_Unit_List)
    return Compilation_Unit_List renames implementation_defined;

  function "&" (Left  : in Compilation_Unit;
                Right : in Compilation_Unit)
    return Compilation_Unit_List renames implementation_defined;

end Compilation_Unit_List_Operations;


------------------------------------------------------------------------
-- ELEMENT_LIST

subtype Element_List is implementation_defined_array;

-- Defines a list of Ada program elements.  For some operations that
-- return Element_List values, the order of components may vary
-- across implementations of ASIS.  Variances are noted in the commentary
-- of individual operations.
--
-- Implemented as an unconstrained array:
-- array ( ASIS_List_Index range <> ) of Element;


Nil_Element_List : Element_List  renames
   implementation_defined_constant;

-- A constant Element List with a void or null value.


package Element_List_Operations is

  function "&" (Left  : in Element_List;
                Right : in Element_List)
    return Element_List renames implementation_defined;

  function "&" (Left  : in Element_List;
                Right : in Element)
    return Element_List renames implementation_defined;

  function "&" (Left  : in Element;
                Right : in Element_List)
    return Element_List renames implementation_defined;
```

```
  function "&" (Left  : in Element;
                Right : in Element)
    return Element_List renames implementation_defined;

end Element_List_Operations;


-----------------------------------------------------------------------
-- ASIS_CHARACTER


subtype ASIS_Character is implementation_defined_enumeration_type;

-- An enumeration type.

package ASIS_Character_Operations is

  function "="  (Left, Right : in ASIS_Character) return Boolean
    renames implementation_defined;

  function "<"  (Left, Right : in ASIS_Character) return Boolean
    renames implementation_defined;

  function "<=" (Left, Right : in ASIS_Character) return Boolean
    renames implementation_defined;

  function ">"  (Left, Right : in ASIS_Character) return Boolean
    renames implementation_defined;

  function ">=" (Left, Right : in ASIS_Character) return Boolean
    renames implementation_defined;

  function To_ASIS_Character (Value : in Standard.Character)
    return ASIS_Character renames implementation_defined;

  function To_Standard_Character (Value : in ASIS_Character)
    return Standard.Character renames implementation_defined;

end ASIS_Character_Operations;


-----------------------------------------------------------------------
-- ASIS_STRING
```

```
subtype ASIS_String is implementation_defined_array;

-- packed array ( Positive range <> ) of ASIS_Character;

Nil_String : constant ASIS_String (1..0) := "";

package ASIS_String_Operations is

  function "="  (Left, Right : in ASIS_String) return Boolean
  renames implementation_defined;

  function "<"  (Left, Right : in ASIS_String) return Boolean
    renames implementation_defined;

  function "<=" (Left, Right : in ASIS_String) return Boolean
    renames implementation_defined;

  function ">"  (Left, Right : in ASIS_String) return Boolean
    renames implementation_defined;

  function ">=" (Left, Right : in ASIS_String) return Boolean
    renames implementation_defined;

  function "&" (Left : in ASIS_String; Right : in ASIS_String)
    return ASIS_String renames implementation_defined;

  function "&" (Left : in ASIS_String; Right : in ASIS_Character)
    return ASIS_String renames implementation_defined;

  function "&" (Left : in ASIS_Character; Right : in ASIS_String)
    return ASIS_String renames implementation_defined;

  function "&" (Left : in ASIS_Character; Right : in ASIS_Character)
    return ASIS_String renames implementation_defined;

  function To_ASIS_String (Value : in Standard.String)
    return ASIS_String renames implementation_defined;

  function To_Standard_String (Value : in ASIS_String)
    return Standard.String renames implementation_defined;

end ASIS_String_Operations;
```

```
-------------------------------------------------------------------------
-- ASIS_TEXT


Maximum_Line_Length : ASIS_Integer renames implementation_defined_constant;
-- The maximum allowed length of a line of text in an Ada program.


Maximum_Line_Number : ASIS_Integer renames implementation_defined_constant;
-- The maximum allowed number of lines of text in an Ada program.


subtype Line_Number is ASIS_Integer range 0..Maximum_line_Number;
-- The range of line numbers within the text of programs.
-- The value 0 is used to indicate an invalid line number.


subtype Line is implementation_defined_private_type;

 -- An Ada text line abstraction (a private type).


-------------------------------------------------------------------------
-- ASIS_TEXT LINE_LIST


subtype Line_List is implementation_defined_array;

-- List of Lines
-- Implemented as an unconstrained array:
-- array ( Line_Number range <> ) of Line;


Nil_Line_List : Line_List  renames
   implementation_defined_constant;

-- A constant Line List with a void or null value.


package Line_List_Operations is

  function "&" (Left  : in Line_List;
                Right : in Line_List)
    return Line_List renames implementation_defined;
```

```
   function "&" (Left  : in Line_List;
                 Right : in Line)
     return Line_List renames implementation_defined;

   function "&" (Left  : in Line;
                 Right : in Line_List)
     return Line_List renames implementation_defined;

   function "&" (Left  : in Line;
                 Right : in Line)
     return Line_List renames implementation_defined;

end Line_List_Operations;


------------------------------------------------------------------------
-- EXCEPTIONS


Inappropriate_Library : exception;

-- Raised when an operation is given a Library value that is
-- not appropriate for the operation.


Inappropriate_Compilation_Unit : exception;

-- Raised when an operation is given a Compilation_Unit value that is
-- not appropriate.


Inappropriate_Program_Element : exception;

-- Raised when an operation is given an Element value that is
-- not appropriate.


Failed : exception;

-- All operations may raise Failed whenever they cannot normally complete
-- their operation.
-- This is a catch-all exception that may be raised for different reasons
-- in different ASIS implementations.
--
-- The Diagnosis and Status functions are provided in ASIS_Libraries,
```

```
   -- ASIS_Compilation_Units, and ASIS_Elements packages so more information
   -- about the failure can be retrieved.


end ASIS_Ada_Program;
```

```
-------------------------------------------------------------------------
-- ASIS PRELIMINARY REVIEW VERSION 0.4
-- Sept 27, 1991
-------------------------------------------------------------------------


with ASIS_Ada_Program;
package ASIS_Environment is

   ----------------------------------------------------------------------
   -- This package provides operations to initialize and finalize the
   -- ASIS environment.

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

   subtype ASIS_String is ASIS_Ada_Program.ASIS_String;


   Nil_String : ASIS_String renames ASIS_Ada_Program.Nil_String;


   ----------------------------------------------------------------------
   -- EXCEPTIONS

   Failed : exception renames ASIS_Ada_Program.Failed;

   -- Raised when the environment failed to initialize or finalize.
   -- Raised when invalid parameters are given to initialize or finalize.


   ----------------------------------------------------------------------
   -- ENVIRONMENT VERSION
```

```
function Version return ASIS_String;

-- Returns a ASIS_String that identifies the current ASIS version.


-------------------------------------------------------------------
-- ENVIRONMENT INITIALIZATION


function Is_Initialized return Boolean;

-- Returns True if the the environment is initialized.


procedure Initialize(Parameters: in ASIS_String := Nil_String);

-- Performs any necessary initialization activities.
-- This must be invoked once before any other ASIS
-- Services.  Parameter values are vendor dependent.
--
-- Preconditions:
--     The environment is not already initialized,
--     otherwise the call is ignored.
--
-- Raises:
--     Failed if the environment failed to initialize.
--
-- Postconditions:  (if no exception is raised)
--     ASIS services are ready for use.


-------------------------------------------------------------------
-- ENVIRONMENT FINALIZATION


function Is_Finalized return Boolean;

-- Returns True if the environment is finalized.


procedure Finalize(Parameters: in ASIS_String := Nil_String);

-- Performs any necessary termination activities.
-- This must be invoked once following all other ASIS
```

```
   -- Services.  Parameter values are vendor dependent.
   --
   -- Preconditions:
   --     The environment must be initialized, otherwise the
   --     call is ignored.
   --
   -- Raises:
   --     Failed if the environment failed to finalize.
   --
   -- Postconditions:  (if no exception is raised)
   --     Subsequent calls to ASIS services are erroneous.


   ------------------------------------------------------------------------
   -- ERROR INFORMATION

   type ASIS_Environment_Error_Kinds is

          (Not_An_Error,

           Environment_Error,      -- The environment failed to
   -- initialize or finalize.

           Parameter_Error,        -- Invalid parameters were given
   -- to initialize or finalize.

           TBD);


   -- Whenever an error condition is detected and the exception Failed
   -- is raised, an ASIS_Environment_Error_Kinds value is stored.  That value
   -- can be retrieved using the Status function.  The Diagnosis function will
   -- retrieve the ASIS_String diagnostic message describing the failure.
   --
   -- Error information is only kept when the catch-all exception Failed
   -- is raised, and refers to the most recent failure.
   --
   -- Note that Status and Diagnosis values are vendor dependent and will
   -- vary among ASIS implementations.


   function Status return ASIS_Environment_Error_Kinds;

   -- Returns the ASIS_Environment_Error_Kinds value for the the most recent
   -- failure.
```

```
  -- Returns Not_An_Error if the most recent initialize or finalize
  -- operations were successful.


  function Diagnosis return ASIS_Ada_Program.ASIS_String;

  -- Returns an ASIS_String value describing the most recent failure.

  -- Returns a Nil_String if the most recent initialize or finalize
  -- operations were successful.



end ASIS_Environment;
```

```
-------------------------------------------------------------------------
-- ASIS PRELIMINARY REVIEW VERSION 0.4
-- Sept 27, 1991
-------------------------------------------------------------------------


with ASIS_Ada_Program;
package ASIS_Libraries is

  -- LRM Section 10.4

  -- ASIS_Libraries encapsulates a set of operations that implement
  -- the ASIS Ada Program Library abstraction.
  -- It assumes only knowledge of the program library that is
  -- specified in section 10.4 of the Ada LRM.

  subtype Library is ASIS_Ada_Program.Library;

  subtype ASIS_String is ASIS_Ada_Program.ASIS_String;

  Nil_String : ASIS_String renames ASIS_Ada_Program.Nil_String;
```

```
-------------------------------------------------------------------------
-- EXCEPTIONS


Failed : exception renames ASIS_Ada_Program.Failed;

-- All operations may raise Failed whenever they cannot normally complete
-- their operation.
-- This is a catch-all exception that may be raised for different reasons
-- in different ASIS implementations.
--
-- The Diagnostic and Status functions are provided so more information
-- about the failure can be retrieved.


-------------------------------------------------------------------------
-- LIBRARY CONSTRUCTORS


function Default_Name return ASIS_String;

-- Returns the default library name.
-- If there is no default library, a Nil_String is returned.


function Default_Parameters return ASIS_String;

-- Returns the default library parameters.
-- If there are no default parameters, a Nil_String is returned.


procedure Associate
   (Program_Library : in out Library;
    Name            : in     ASIS_String;
    Parameters      : in     ASIS_String := Default_Parameters);

-- Associates Library with the given Name and Parameters values.
-- A library can have at most one set of values associated with
-- it at any time.  Name and Parameters values previously
-- associated are replaced.
--
-- Preconditions:
-- Is_Open( Program_Library ) = False
--
-- Postconditions:
```

```
-- Name( Program_Library ) = Name
-- Parameters( Program_Library ) = Parameters
-- Has_Associations( Program_Library ) = True
-- Is_Open( Program_Library ) = False


procedure Open (Program_Library : in out Library);

-- Opens the Ada Library using the associated Name and
-- Parameters values.
--
-- Preconditions:
-- Has_Associations( Program_Library ) = True
-- Is_Open( Program_Library ) = False
--
-- Postconditions:
-- Is_Open( Program_Library ) = True


procedure Close (Program_Library : in out Library);

-- Closes the Ada Library.
-- Any previous associations to Library are retained, therefore
-- allowing Library to be re-opened.
--
-- Preconditions:
--     Is_Open( Program_Library ) = True
--
-- Postconditions:
-- Is_Open( Program_Library ) = False
-- Has_Associations( Program_Library ) = True
--
--    Any value obtained from a library becomes invalid when that
--    library is closed.
-- Subsequent calls to ASIS services using Compilation_Unit
--    and Element handles obtained from the library are erroneous.

-------------------------------------------------------------------------
-- LIBRARY HANDLE DESTRUCTOR


procedure Dissociate (Program_Library : in out Library);

-- Severs all previous associations to Program_Library.
-- A given library with no associations is returned unchanged.
```

```
--
-- Note that this operation has no affect on the physical Ada Library.
--
-- Preconditions:
-- Is_Open( Program_Library ) = False
--
-- Postconditions:
-- Any storage for Library is deallocated.
-- Has_Associations( Program_Library ) = False


-------------------------------------------------------------------
-- LIBRARY ATTRIBUTES

function Is_Equal (Left, Right : in Library) return Boolean;

-- Returns True if Left and Right designate the same library file.


function Exists (Program_Library : in Library) return Boolean;

-- Returns True if Program_Library designates a library file that exists.


function Is_Open (Program_Library : in Library) return Boolean;

-- Returns True if Program_Library is open.


function Has_Associations ( Program_Library : in Library)
  return Boolean;

-- Returns True if name and parameters values have been
-- associated to Program_Library.


function Name (Program_Library : in Library) return ASIS_String;

-- Returns the name associated with Program_Library.
-- if not Has_Associations( Program_Library ), a Nil_String is returned.


function Parameters (Program_Library : in Library) return ASIS_String;

-- Returns the parameters associated with Program_Library.
```

```
-- if not Has_Associations( Program_Library ), a Nil_String is returned.


-------------------------------------------------------------------------
-- ERROR INFORMATION

type ASIS_Library_Error_Kinds is

  (Not_An_Error,              -- The library has no error condition.

   Capacity_Error,            -- The implementation cannot support a
     -- request to open an additional library.

   Data_Error,                -- The library file specified by Name(Library)
     -- does not contain a valid Ada library.
     -- ie. The contents of the library are the
     -- wrong version or format.

   Initialization_Error,      -- The underlying components have not been
     -- initialized by calling
     -- ASIS_System_Services.Initialize.

   Name_Error,                -- The Name(Library) does not specify a
     -- library file.
     -- ie. The named library does not exist.

   Status_Error,              -- The Library is not in valid state (either
     -- open or closed) to perform a given operation.

   Storage_Error,             -- The system can not perform new allocations.

   Use_Error,                 -- Characteristics of the library file specified
     -- by Name(Library) or Parameters(Library) do
     -- not permit performance of a given operation.

   TBD);


-- Whenever an error condition is detected and the exception Failed
-- is raised, an ASIS_Library_Error_Kinds value is stored for the Library
-- involved in the failure.   That value can be retrieved using the Status
-- function.  The Diagnosis function will retrieve the ASIS_String
-- diagnostic message for the Library_Error_Kinds value.
--
-- Status information is only kept when the catch-all exception Failed
```

```
   -- is raised.
   --
   -- Note that Status and Diagnosis values are vendor dependent and will
   -- vary among ASIS implementations.


   function Status (Program_Library : in Library)
     return ASIS_Library_Error_Kinds;

   -- Returns the ASIS_Library_Error_Kinds value associated with
   -- Program_Library.

   -- Returns Not_An_Error if Program_Library has no error kind associated
   -- with it, or if not Has_Associations(Program_Library).
   -- EG. It has not been involved in a failed operation.


   function Diagnosis (Program_Library : in Library) return ASIS_String;

   -- Returns an ASIS_String value associated with Program_Library.


   procedure Dissociate_Library_Error_Information;

   -- Removes the association of status and diagnosis error information with
   -- all Library handles that have been involved in a failed operation.
   --
   -- This operation does not affect Library handles in any other way.


   ------------------------------------------------------------------------
   -- DEBUGGING

   function Debug_Image (Program_Library : in Library) return ASIS_String;

   -- Returns an ASIS_String value containing implementation defined debug
   -- information associated with Program_Library.


private


end ASIS_Libraries;
```

```
-------------------------------------------------------------------------
-- ASIS PRELIMINARY REVIEW VERSION 0.4
-- Sept 27, 1991
-------------------------------------------------------------------------


with ASIS_Ada_Program;
package ASIS_Compilation_Units is

  -- LRM Chapter 10

  subtype Library is ASIS_Ada_Program.Library;


  subtype Compilation_Unit is ASIS_Ada_Program.Compilation_Unit;

  Nil_Compilation_Unit : Compilation_Unit renames
                           ASIS_Ada_Program.Nil_Compilation_Unit;

  -- A constant Ada Compilation Unit with a void or null value.


  subtype Element is ASIS_Ada_Program.Element;

  subtype Compilation_Unit_List is ASIS_Ada_Program.Compilation_Unit_List;

  Nil_Compilation_Unit_List : Compilation_Unit_List  renames
                                ASIS_Ada_Program.Nil_Compilation_Unit_List;

  -- A constant Ada Compilation Unit List with a void or null value.


  subtype ASIS_String is ASIS_Ada_Program.ASIS_String;

  Nil_String : ASIS_String renames ASIS_Ada_Program.Nil_String;

  subtype ASIS_Time is ASIS_Ada_Program.ASIS_Time;
```

```
-----------------------------------------------------------------------
-- EXCEPTIONS


Inappropriate_Library : exception renames
                        ASIS_Ada_Program.Inappropriate_Library;

-- Raised when an operation is given a Library value that is
-- not appropriate for the operation.
--
-- All ASIS_Compilation_Units operations that take a library parameter
-- will raise this exception when given a library that is not open.


Inappropriate_Compilation_Unit : exception renames
                        ASIS_Ada_Program.Inappropriate_Compilation_Unit;

-- Raised when an operation is given a Compilation_Unit value that is
-- not appropriate.


Failed : exception renames ASIS_Ada_Program.Failed;

-- All operations may raise Failed whenever they are given appropriate
-- input but cannot normally complete their operation.
-- This is a catch-all exception that may be raised for different reasons
-- in different ASIS implementations.
--
-- The Diagnosis and Status functions are provided so more information
-- about the failure can be retrieved.


-----------------------------------------------------------------------
-- 10.1 COMPILATION UNITS - KINDS


type Compilation_Unit_Kinds is

-- The kinds of Ada Program Library compilation units.

   (A_Subprogram_Declaration,
    A_Package_Declaration,
    A_Generic_Declaration,
    A_Generic_Instantiation,
    A_Subprogram_Body,
```

```
    A_Package_Body,
    A_Subunit,

    A_Nonexistent_Library_Unit,   -- A unit that no longer exists but is
  -- mentioned in a with clause of another
  -- unit or is the specification of a
  -- library unit body.

    A_Nonexistent_Secondary_Unit, -- A unit that does not exist but is
  -- mentioned in a body stub of another
  -- unit or is the proper body of a
  -- library unit.

    Unknown,      -- The unit kind is not known.

    Not_A_Compilation_Unit        -- A void compilation unit kind value.
    );


function Kind (A_Compilation_Unit : in Compilation_Unit)
  return Compilation_Unit_Kinds;

-- Returns the kind of the compilation unit.
--
-- Returns Not_A_Compilation_Unit for any inappropriate compilation unit
-- such as a Nil_Compilation_Unit.


------------------------------------------------------------------------
-- COMPILATION_UNIT ASSIGNMENT


procedure Assign (Target : in out Compilation_Unit;
                  Source : in     Compilation_Unit);

-- Assigns the source compilation unit to the target compilation unit.
--
-- Postconditions:  (if no exception is raised)
--   Depending on the ASIS implementation, the target compilation unit
--    may be dissociated before performing the assignment.
--   Is_Equal(Target, Source)


------------------------------------------------------------------------
-- 10.1 COMPILATION UNITS - BELONGING TO A LIBRARY
```

```
-- Compilation units are said to belong to a program library.
-- These queries search a given library for compilation units.
-- Successful searches have the affect of associating the logical
-- compilation unit handles with the corresponding physical compilation
-- unit values within the library.
--
-- Exists(A_Compilation_Unit) = True when a Compilation_Unit is successfully
-- obtained from these queries.
--
--
-- Eligible Compilation Units
--
-- Some ASIS implementation may allow more than one version of a library
-- unit to exist within a library.  If so, then only one of those library
-- units is "eligible" to be used in an executable program.  Other versions
-- of the library unit are "ineligible".  The search rules used to determine
-- library unit eligibility are defined by each ASIS implementation.


function Library_Unit (Name : in String; Program_Library : in Library)
  return Compilation_Unit;

-- Returns an eligible library unit with the distinct name belonging
-- to the library.  See LRM 10.1(3).
--
-- If no such library unit exists, a Nil_Compilation_Unit is returned.
--
-- Raises Inappropriate_Library
-- if not ASIS_Libraries.Is_Open(Program_Library).


function Secondary_Unit (Name : in String; Program_Library : in Library)
  return Compilation_Unit;

-- Returns an eligible secondary unit with the distinct name belonging
-- to the library.  See LRM 10.1(3).
--
-- If no such secondary unit exists, a Nil_Compilation_Unit is returned.
--
-- Raises Inappropriate_Library
-- if not ASIS_Libraries.Is_Open(Program_Library).
```

```
subtype Library_Unit_List is Compilation_Unit_List;

function All_Library_Units (Program_Library : in Library)
  return Library_Unit_List;

-- Returns the complete list of all versions of library units in the
-- library.
--
-- The occurrence and order of units in the list is implementation defined.
--
-- If no library units exist within the library,
-- a Nil_Compilation_Unit_List is returned.
--
-- Raises Inappropriate_Library
-- if not ASIS_Libraries.Is_Open(Program_Library).


function Library_Units (Program_Library : in Library)
  return Library_Unit_List;

-- Returns a list of eligible and distinctly named library units
-- belonging to the library.  See LRM 10.1(3).
--
-- The order of units in the list is implementation defined.
--
-- If no library units exist within the library,
-- a Nil_Compilation_Unit_List is returned.
--
-- Raises Inappropriate_Library
-- if not ASIS_Libraries.Is_Open(Program_Library).


subtype Secondary_Unit_List is Compilation_Unit_List;


function All_Secondary_Units (Program_Library : in Library)
  return Secondary_Unit_List;

-- Returns the complete list of all versions of secondary units in the
-- library.
--
-- The occurrence and order of units in the list is implementation defined.
--
-- If no secondary units exist within the library,
-- a Nil_Compilation_Unit_List is returned.
```

```
--
-- Raises Inappropriate_Library
-- if not ASIS_Libraries.Is_Open(Program_Library).


function Secondary_Units (Program_Library : in Library)
  return Secondary_Unit_List;

-- Returns a list of eligible and distinctly named secondary units
-- belonging to the library.  See LRM 10.1(3).
--
-- The order of units in the list is implementation defined.
--
-- If no secondary units exist within the library,
-- a Nil_Compilation_Unit_List is returned.
--
-- Raises Inappropriate_Library
-- if not ASIS_Libraries.Is_Open(Program_Library).



function All_Compilation_Units (Program_Library : in Library)
  return Compilation_Unit_List;

-- Returns the complete list of all versions of compilation units in the
-- Library.
--
-- If no compilation units exist within the Library, an empty list is
-- returned.
--
-- Raises Inappropriate_Library
-- if not ASIS_Libraries.Is_Open(Program_Library).


function Compilation_Units (Program_Library : in Library)
  return Compilation_Unit_List;

-- Returns a list of eligible and distinct compilation units belonging to
-- the given Library.  See LRM 10.1(3).
--
-- The order of units in the list is implementation defined.
--
-- If no compilation units exist within the Library, an empty list is
-- returned.
--
```

```
-- Raises Inappropriate_Library
-- if not ASIS_Libraries.Is_Open(Program_Library).


-------------------------------------------------------------------------
-- 10.1 ENCLOSING LIBRARY


function Enclosing_Library (A_Compilation_Unit : in Compilation_Unit)
  return Library;

-- Returns the library to which the compilation unit belongs.
-- Compilation units retrieved through any of the search operations
-- keep track of the library in which they were found.
--
-- Since the type Library is a limited private, this function is only
-- intended to be used to supply the library parameter to other operations.
-- This conveniently eliminates the need to make the original library
-- visible at the place of each call where a library parameter is required.
--
-- Raises Inappropriate_Compilation_Unit
-- if not Exists(A_Compilation_Unit)


-------------------------------------------------------------------------
-- 10.1 CORRESPONDING COMPILATION UNITS


function Corresponding_Library_Unit
   (Secondary_Unit  : in Compilation_Unit)
  return Compilation_Unit;

function Corresponding_Library_Unit
   (Secondary_Unit  : in Compilation_Unit;
    Program_Library : in Library)
  return Compilation_Unit;

-- Returns the compilation unit that is the semantically correct
-- corresponding library unit of the given Secondary_Unit.  The library
-- unit returned is the library unit for which the secondary unit is
-- the proper body.
--
-- Returns a Nil_Compilation_Unit for all given Secondary_Unit
-- values that do not have an existing corresponding library unit.
-- The corresponding library unit would have to have existed when the
```

```
-- secondary unit was compiled, however, some ASIS implementations allow
-- compilation units to be removed without affecting other dependent
-- compilation units.
--
-- If a Program_Library is given, that library is used to search for the
-- corresponding compilation unit.
--
-- If a library unit is given, the same compilation unit is returned.
-- If no corresponding library unit exists, a Nil_Compilation_Unit is
-- returned.
--
-- LRM 10.1(6).  A subprogram body given in a compilation unit is
-- interpreted as a secondary unit if the program library already
-- contains a library unit that is a subprogram with the same name;
-- it is otherwise interpreted both as a library unit and as the
-- corresponding library unit body (that is, as a secondary unit).
-- In this case, given a secondary unit handle whose Kind is
-- A_Subprogram_Body, this operation returns the library unit handle
-- whose Kind is A_Subprogram_Declaration.
--
-- Raises Inappropriate_Compilation_Unit
-- if not Exists(Secondary_Unit) or Is_Subunit(Secondary_Unit)
--   or Kind(Secondary_Unit) = Unknown
--
-- Raises Inappropriate_Library
-- if not ASIS_Libraries.Is_Open(Program_Library).


function Corresponding_Secondary_Unit
   (Library_Unit    : in Compilation_Unit)
  return Compilation_Unit;

function Corresponding_Secondary_Unit
   (Library_Unit    : in Compilation_Unit;
    Program_Library : in Library)
  return Compilation_Unit;

-- Returns the compilation unit that is the semantically correct
-- corresponding library unit body (the secondary unit) of the given
-- Library_Unit.
--
-- Returns a Nil_Compilation_Unit for all given Library_Unit
-- values that do not have an existing corresponding secondary unit,
-- for example, the library unit is a package declaration whose body is
-- not required or has not yet been compiled.
```

```
--
-- If a secondary unit is given, the same compilation unit is returned.
-- If no corresponding secondary unit exists, a Nil_Compilation_Unit is
-- returned.
--
-- If a Program_Library is given, that library is used to search for the
-- corresponding compilation unit.
--
-- LRM 10.1(6).  A subprogram body given in a compilation unit is
-- interpreted as a secondary unit if the program library already
-- contains a library unit that is a subprogram with the same name;
-- it is otherwise interpreted both as a library unit and as the
-- corresponding library unit body (that is, as a secondary unit).
-- In this case, given a library unit handle whose Kind is
-- A_Subprogram_Declaration, this operation returns the secondary
-- unit handle whose Kind is A_Subprogram_Body.
--
-- Raises Inappropriate_Compilation_Unit
-- if not Exists(Library_Unit) or Is_Subunit(Library_Unit)
--   or Kind(Library_Unit) = Unknown
--
-- Raises Inappropriate_Library
-- if not ASIS_Libraries.Is_Open(Program_Library).


-------------------------------------------------------------------------
-- 10.1(7) COMPILATION PRAGMAS


subtype Pragma_List is ASIS_Ada_Program.Element_List;

function Compilation_Pragmas
   (A_Compilation_Unit : in Compilation_Unit)
  return Pragma_List;

-- Returns a list of pragmas that apply to the whole of the compilation
-- of the compilation unit.
--
-- Pragmas returned in this list must appear before the first compilation
-- unit of the compilation that contained the given compilation unit.
--
-- Use the Pragmas operation in ASIS_Elements to obtain a list of
-- pragmas in a given context within a compilation unit.
--
-- A Nil_Element_List is returned if the compilation unit has no such
```

```
-- applicable pragmas or if the Kind(A_Compilation_Unit) = Unknown.
--
-- Further analysis:
--   Use ASIS_Elements pragma operations to decompose pragmas.
--
-- Raises Inappropriate_Compilation_Unit
-- if not Exists(A_Compilation_Unit)


-------------------------------------------------------------------------
-- 10.1 COMPILATION UNIT ATTRIBUTES


function Is_Nil (A_Compilation_Unit : in Compilation_Unit) return Boolean;

-- Returns True if the compilation unit has a void or null value.
--
-- All Compilation_Unit kinds are appropriate.


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

-- Returns True if Left and Right represent the same compilation unit.
--
-- All Compilation_Unit kinds are appropriate.


function Name (A_Compilation_Unit : in Compilation_Unit) return String;

-- Returns the Name associated with the given compilation unit.
-- Then name is the Ada identifier that names the compilation unit.
--
-- If a Nil_Compilation_Unit is given then a null string is returned.
--
-- All Compilation_Unit kinds are appropriate.


function Unique_Name (A_Compilation_Unit : in Compilation_Unit)
  return ASIS_String;

-- Returns an ASIS_String that uniquely identifies the given compilation unit.
-- The result may vary depending on the ASIS implementation.  The unique
-- name may include the name of the library, file system paths, sublibrary
-- files, version numbers, kind, or any other information that a vendor
-- may use to uniquely identify the compilation unit.
```

```
--
-- If a Nil_Compilation_Unit is given then a Nil_String is returned.
--
-- All Compilation_Unit kinds are appropriate.


function Exists (A_Compilation_Unit : in Compilation_Unit) return Boolean;

-- Returns True if Compilation_Unit has been successfully compiled into
-- a program library.
--
-- Returns False when given a Nil_Compilation_Unit.
--
-- All Compilation_Unit kinds are appropriate.


function Can_Be_Main_Program (A_Compilation_Unit : in Compilation_Unit)
  return Boolean;

-- Returns True if Compilation_Unit exists and is a subprogram
-- that is a library unit that can be used as a main program.
-- See LRM 10.1(8)
--
-- Results of this function may vary according to the requirements
-- an Ada implementation may impose on a main program.
--
-- Raises Inappropriate_Compilation_Unit
-- if not Exists(A_Compilation_Unit) or Kind(A_Compilation_Unit) = Unknown


function Body_Is_Optional (A_Compilation_Unit : in Compilation_Unit)
  return Boolean;

-- Returns True if Compilation_Unit's body is not required.
-- See LRM 7.1(5)
--
-- Raises Inappropriate_Compilation_Unit
-- if not Exists(A_Compilation_Unit) or Kind(A_Compilation_Unit) = Unknown


function Text_File_Name (A_Compilation_Unit : in Compilation_Unit)
  return ASIS_String;

-- Returns the text file name for Compilation_Unit.
-- The text file name identifies the file or other structure that
```

```
-- contained the text for the given compilation unit when it was
-- submitted to the compiler.
--
-- If not Exists(A_Compilation_Unit) or Kind(A_Compilation_Unit) = Unknown
-- or if the text file name is not available, a Nil_String is returned.


function Text_File_Form (A_Compilation_Unit : in Compilation_Unit)
  return ASIS_String;

-- Returns the text file form for Compilation_Unit.
--
-- If not Exists(A_Compilation_Unit) or Kind(A_Compilation_Unit) = Unknown
-- or if the text file form is not available, a Nil_String is returned.


function Object_File_Name (A_Compilation_Unit : in Compilation_Unit)
  return ASIS_String;

-- Returns the object file name created for Compilation_Unit.
--
-- If not Exists(A_Compilation_Unit) or Kind(A_Compilation_Unit) = Unknown
-- or if the object file name is not available, a Nil_String is returned.


function Object_File_Form (A_Compilation_Unit : in Compilation_Unit)
  return ASIS_String;

-- Returns the object file Form created for Compilation_Unit.
--
-- If not Exists(A_Compilation_Unit) or Kind(A_Compilation_Unit) = Unknown
-- or if the object file form is not available, a Nil_String is returned.


function Compilation_Command_Line_Options
   (A_Compilation_Unit : in Compilation_Unit)
  return ASIS_String;

-- Returns the command line options used to compile Compilation_Unit.
--
-- If not Exists(A_Compilation_Unit) or Kind(A_Compilation_Unit) = Unknown
-- or if compilation command line options are not available,
-- a Nil_String is returned.
```

```
function Time_Of_Last_Update (A_Compilation_Unit : in Compilation_Unit)
  return ASIS_Time;

-- Returns the time that Compilation_Unit was most recently
-- updated in its library.
--
-- If not Exists(A_Compilation_Unit) or Kind(A_Compilation_Unit) = Unknown
-- or if the time is not available, a Nil_time value is returned.


function Compilation_Cpu_Time (A_Compilation_Unit : in Compilation_Unit)
  return ASIS_Time;

-- Returns the CPU time used to compile Compilation_Unit.
--
-- The value is relative to Nil_time as defined below.
--
-- For example:
--
--   A value where Year = 1901, Month = 1, Day = 1, and Seconds = 10.0
--   means the compilation took 10.0 seconds.
--   A value where Year = 1902, Month = 4, Day = 6, and Seconds = 23.0
--   means the compilation took 1 year, 3 months, 5 days, 23.0 seconds.
--
-- If not Exists(A_Compilation_Unit) or Kind(A_Compilation_Unit) = Unknown
-- or if compilation CPU time is not available, a Nil_time value is returned.


------------------------------------------------------------------------
-- 10.1 COMPILATION UNIT ATTRIBUTES - GENERAL OR IMPLEMENTATION DEFINED


function Has_Attribute
   (A_Compilation_Unit : in Compilation_Unit;
    Attribute          : in ASIS_String)
  return Boolean;

-- Returns True if the compilation unit has the given attribute.
--
-- Results of this operation may vary across ASIS implementations.
--
-- Raises Inappropriate_Compilation_Unit
-- if not Exists(A_Compilation_Unit) or Kind(A_Compilation_Unit) = Unknown
```

```
function Attribute_Value_Delimiter return ASIS_String;

-- Returns the ASIS_String used as a delimiter separating
-- individual values within the ASIS_String Attribute_Values of
-- a compilation unit.
--
-- Results of this operation may vary across ASIS implementations.


function Attribute_Values
   (A_Compilation_Unit : in Compilation_Unit;
    Attribute          : in ASIS_String)
  return ASIS_String;

-- Returns the ASIS_String containing zero or more images of values that
-- may be associated with the given attribute.  The Attribute_Value_Delimiter
-- separates individual values in the ASIS_String when more than one value
-- is returned.
--
-- Results of this operation may vary across ASIS implementations.
--
-- If not Exists(A_Compilation_Unit) or Kind(A_Compilation_Unit) = Unknown
-- or the compilation unit does not have the attribute or associated values,
-- a Nil_String is returned.



function Attribute_Time
     (A_Compilation_Unit : in Compilation_Unit;
      Attribute          : in ASIS_String)
  return ASIS_Time;

-- Returns the ASIS_Time value associated with the given attribute.
--
-- Results of this operation may vary across ASIS implementations.
--
-- If not Exists(A_Compilation_Unit) or Kind(A_Compilation_Unit) = Unknown
-- or the compilation unit does not have the attribute or associated value,
-- Nil_Time is returned.



------------------------------------------------------------------------
-- 10.1 COMPILATION UNITS - ELEMENT GATEWAY
```

```
function Enclosing_Compilation_Unit (Program_Element : in Element)
  return Compilation_Unit;


-- Returns the Compilation_Unit that contains the given Element.
--
-- This is a gateway from Element handles to Compilation_Unit handles.
--
-- Raises Inappropriate_Program_Element
-- if ASIS_Elements.Is_Nil(Program_Element).


subtype Declaration is ASIS_Ada_Program.Element;

function Unit_Declaration (A_Compilation_Unit : in Compilation_Unit)
  return Declaration;

-- Returns the declaration of the library unit or secondary unit.
--
-- This is a gateway from Compilation_Unit handles to Element handles.
--
-- If Kind(A_Compilation_Unit) = Unknown an ASIS_Ada_Program.Nil_Element
-- is returned.
--
-- Use ASIS_Declarations operations to further decompose these elements.
--
-- Raises Inappropriate_Compilation_Unit
-- if not Exists(A_Compilation_Unit)


-------------------------------------------------------------------------
-- 10.1.1 CONTEXT CLAUSES - ELEMENT GATEWAY

subtype Context_Clause_Element is ASIS_Ada_Program.Element;

subtype Context_Clause_Element_List is ASIS_Ada_Program.Element_List;

function Context_Clause_Elements
   (A_Compilation_Unit : in Compilation_Unit)
  return Context_Clause_Element_List;

-- Returns a list of with clauses, use clauses, and pragmas that
-- explicitly reside in the context clause of the compilation unit.
--
```

```
-- With clause and use clauses that implicitly apply to a given secondary
-- unit are not returned in this list.  See LRM 10.1.1(4).  Use the
-- compilation unit relationship operations and Relation_Kinds Supporters
-- and Direct_Supporters to find all the supporting library units.
--
-- Pragmas returned in this list must occur immediately after the context
-- clause of a compilation unit before the subsequent library unit or
-- secondary unit.  See LRM Appendix B. Pragma ELABORATE.
--
-- Results of this operation may vary across ASIS implementations.
-- Some implementations normalize all with clauses and use clauses
-- containing multiple names of library units into an equivalent sequence
-- of corresponding single with or use clauses.  Similarly, an
-- implementation may keep a name only once even though that name may appear
-- more than once in the with clauses or use clauses of a compilation unit.
-- See LRM 8.4(9) and LRM 10.1.1(8).
--
-- If not Exists(A_Compilation_Unit) or Kind(A_Compilation_Unit) = Unknown
-- or if the compilation unit has no context clause, a Nil_Element_List is
-- returned
--
-- Further analysis:
--   Use ASIS_Elements.Major_Element_Kind to analyze the kind of elements
--   in a context clause.
--   Use ASIS_Elements pragma operations to decompose pragmas.
--
-- Raises Inappropriate_Compilation_Unit
-- if not Exists(A_Compilation_Unit)


subtype Identifier_Reference_List is ASIS_Ada_Program.Element_List;

function Referenced_Units
   (With_Clause_Or_Use_Clause : in Context_Clause_Element)
  return Identifier_Reference_List;

-- Returns a list of elements of the subtype Identifier_Reference that are
-- mentioned in the with clause or the use clause.
--
-- Results of this operation may vary across ASIS implementations.
-- Depending on the behavior of the function Context_Clause_Elements,
-- this operation will either return a list containing one or more
-- identifiers or will always return a list containing a single identifier.
--
-- Appropriate ASIS_Elements.Major_Element_Kinds:
```

```
--   A_With_Clause
--   A_Use_Clause occurring within a context clause of a compilation unit
--
-- A_Use_Clause that is a declarative item is inappropriate.
-- Use ASIS_Names_And_Expressions.Named_Packages to analyze use clauses that
-- are declarative items.


-------------------------------------------------------------------------
-- 10.2 SUBUNITS OF COMPILATION UNITS


function Is_Subunit
   (A_Compilation_Unit : in Compilation_Unit)
  return Boolean;

-- Returns True if the given Compilation_Unit is a subunit.
--
-- Raises Inappropriate_Compilation_Unit
-- if not Exists(A_Compilation_Unit) or Kind(A_Compilation_Unit) = Unknown


function Subunits
   (Parent_Unit     : in Compilation_Unit)
  return Secondary_Unit_List;

function Subunits
   (Parent_Unit     : in Compilation_Unit;
    Program_Library : in Library)
  return Secondary_Unit_List;

-- Returns a list of subunits corresponding to the
-- the body stubs that appear in the given Parent_Unit.
--
-- Returns a Nil_Compilation_Unit_List if the parent unit does not
-- contain any body stubs.
--
-- The list includes subunits existing in the given library and 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
-- secondary units so their Kind will be A_Nonexistent_Secondary_Unit.
--
-- Subunits can also be obtained through the query
-- Related_Compilation_Units using the Family relation.
--
```

```
-- Raises Inappropriate_Compilation_Unit
-- if not Exists(Parent_Unit) or Kind(Parent_Unit) = Unknown
--
-- Raises Inappropriate_Library
-- if not ASIS_Libraries.Is_Open(Program_Library).


function Subunit_Parent
   (Subunit         : in Compilation_Unit)
  return Compilation_Unit;

function Subunit_Parent
   (Subunit         : in Compilation_Unit;
    Program_Library : in Library)
  return Compilation_Unit;

-- Returns the Compilation_Unit that contains the body stub
-- of the given Subunit.
--
-- Returns a Nil_Compilation_Unit if the subunit parent cannot be
-- found within the library.
--
-- Raises Inappropriate_Compilation_Unit
-- if not Is_Subunit(Subunit)
--
-- Raises Inappropriate_Library
-- if not ASIS_Libraries.Is_Open(Program_Library).



function Subunit_Ancestor
   (Subunit         : in Compilation_Unit)
  return Compilation_Unit;

function Subunit_Ancestor
   (Subunit         : in Compilation_Unit;
    Program_Library : in Library)
  return Compilation_Unit;

-- Returns the Compilation_Unit that is the ancestor library unit
-- of the given Subunit.  The ancestor unit is a parent unit that
-- is not itself a subunit.  See LRM 10.2(5).
--
-- Returns a Nil_Compilation_Unit if the subunit ancestor cannot be
-- found within the library.
```

```
--
-- Raises Inappropriate_Compilation_Unit
-- if not Is_Subunit(Subunit)
--
-- Raises Inappropriate_Library
-- if not ASIS_Libraries.Is_Open(Program_Library).


-------------------------------------------------------------------------
-- 10.3 ORDER OF COMPILATION

-- Compilation Unit Relationships
--
-- Relationship queries provide handles to compilation units that are
-- related in a given manner to one or more given compilation units.
-- Compilation units located by the queries are returned in an ordered
-- list.
--
-- The following describes the semantics of the list of units returned
-- by these queries:
--   Related_Compilation_Units
--   Compilation_Order
--   Recompilation_Order
--   Elaboration_Order
--
-- The list contains four possible sublists:
--
--   1) A sublist of existing ordered units
--   2) A sublist of units that are inconsistent due to the
--      recompilation of one or more related units.
--   3) A sublist of units that have missing (nonexistent) related units.
--   4) A sublist of circularities among the related units.
--
-- Each of the four sublists is delimited by a Nil_Compilation_Unit.
--
--
-- 1) THE ORDERED UNITS SUBLIST
--
-- The semantics of the ordering of units in the first sublist are
-- defined by each of the individual operations.
--
--
-- 2) THE INCONSISTENT UNITS SUBLIST
--
-- The second sublist contains pairs of units; Each pairing contains the
```

```
-- inconsistent unit followed by the unit causing the inconsistency.
-- An inconsistent unit is a Compilation_Unit that exists but is obsolete
-- due to recompilation of any supporters within the given Library.
-- See LRM 10.3
--
-- For example:
--
-- Given a sublist containing the units:  A B A C D E
--
-- It can be determined that:
--   A is inconsistent since its supporter B has been recompiled.
--   A is inconsistent since its supporter C has been recompiled.
--   D is inconsistent since its supporter E has been recompiled.
--
--
--
-- 3) THE MISSING UNITS SUBLIST
--
-- The third sublist contains pairs of units; Each pairing contains the
-- existing unit followed by the missing related unit.
-- A missing unit is a Compilation_Unit that has a valid name, but its
-- Kind is either A_Nonexistent_Library_Unit or A_Nonexistent_Secondary_Unit.
--
-- For example:
--
-- Given a sublist containing the units:  A B A C
--
-- If Kind(B) = A_Nonexistent_Library_Unit
-- and Kind(C) = A_Nonexistent_Secondary_Unit
--
-- It can be determined that:
--   A is missing its supporter B.
--   A is missing a related secondary unit C.
--
--
-- 4) THE CIRCULAR DEPENDENCIES SUBLIST
--
-- Circular dependencies between compilation units are provided in the
-- fourth sublist.  There may be more than one set of circular
-- dependencies.  The ordering of distinct sets in the sublist is
-- implementation defined.
--
-- The semantics of set of units in the circular sublist structures are the
-- same for each operation:
--
```

```
-- The sublist contains pairs of units; Each pairing contains a unit
-- followed by its supporter unit.  A circularity is established when
-- a following supporter unit is equal to the first unit in the sublist
-- (See the unit A in the example below).  The next set of circular
-- dependent units, if any, starts with the next unit in the sublist
-- (the unit D below).
--
-- For example:
--
-- Given a list containing the units:  A B B C C A D E E F F G G D
--
-- 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.
--
--
-- IMPORTANT NOTE:
--
-- For each operation that returns inconsistent, missing, or circular units,
-- the appearance of any of these units in those sublists means that the
-- first sublist containing ordered or related units cannot be guaranteed
-- valid or complete.
--
-- Programs that do not first check for inconsistent, missing, or circular
-- units may be ERRONEOUS.
--
-------------------------------------------------------------------------


type Relation_Kinds is

   -- Defines the kinds of relations between compilation units

 (Direct_Supporters,

  Supporters,

  ---------------------------------------------------------------------
  -- Definition:  SUPPORTERS of a unit
  --
  -- Supporters of a compilation unit are units that must be compiled
  -- before that compilation unit can be compiled.  A compilation unit
```

```
-- cannot be successfully compiled if one or more supporters are
-- missing or obsolete.
--
-- Library units mentioned in with clauses of other compilation
-- units are direct supporters of those compilation units.
-- The property is transitive; Units that support library units withed
-- by other compilation units are indirect supporters of those
-- compilation units.
--
-- The package Standard is a supporter of every unit, but is not
-- considered to be a direct supporter unless it is mentioned in a with
-- clause.
--
-- For example:
-- If A withs B and B withs C.
-- Then C directly supports B,
--      C indirectly supports A, and
--      both B and C are supporters of A.
--
-- A library unit is a direct supporter of its corresponding
-- secondary unit.
-- The parent unit of a subunit is a direct supporter of the subunit.
-------------------------------------------------------------------------

Direct_Dependents,

Dependents,

-------------------------------------------------------------------------
-- Definition:  DEPENDENTS of a unit
--
-- A compilation unit that mentions other library units in its with
-- clauses directly depends on those library units.  The property is
-- transitive;  A unit that depends on a given unit also indirectly
-- depends on the given unit's supporters.
--
-- For example:
-- If A withs B and B withs C
--     then B directly depends on C,
--      A indirectly depends on C, and
--      both A and B are dependents of C.
--
-- Dependents are all the units that must be recompiled if the unit
-- is recompiled.  A secondary unit is a direct dependent of its
-- corresponding library unit.
```

```
--
-- Dependencies between compilation units may also be introduced
-- by inline inclusions, for certain compiler optimizations, and
-- for certain implementations of generic program units.
-- See LRM 10.3(6-9)
-------------------------------------------------------------------

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 spec, body,
-- and subunits (and subunits of subunits).
-------------------------------------------------------------------

Extended_Family

-------------------------------------------------------------------
-- Definition:  EXTENDED FAMILY of a unit
--
-- The extended family of a given unit is defined as the set of
-- compilation units that comprise the given unit's spec, body,
-- subunits and transitive supporters, and the extended family of
-- each transitive supporting unit.
-- The extended family of a unit that is a main program includes
-- all the Ada units ultimately required to form the program.
--
-- The term closure is commonly used to with similar meaning.
--
-- For example:
--  Assume the body of A has a subunit A.S.
-- If A withs B and B withs C and C does not with a library unit.
-- Then the extended family of A is:
--    library unit C
--    secondary unit C    (if required)
--    library unit B
--    secondary unit B    (if required)
--    library unit A
--    secondary unit A
--    secondary unit A.S  (subunit)
--
-------------------------------------------------------------------
```

```
   );


function Related_Compilation_Units
   (Compilation_Units : in Compilation_Unit_List;
    Dependent_Units   : in Compilation_Unit_List;
    Program_Library   : in Library;
    Relation          : in Relation_Kinds )
  return Compilation_Unit_List;

-- Produces the list of compilation units related to
-- the given compilation units by the specified relation.
--
-- Dependent units are only required when the relation is:
--   Direct_Dependents or Dependents.
--
-- The extended families of the dependent units provide a context for
-- the operation to select dependents of the given compilation units.
--
-- The return list contains the sublists described above for the
-- operations on Compilation Unit Relationships
--
-- Raises Inappropriate_Compilation_Unit
-- if any of the given compilation units or dependent units do not exist.
--
-- Raises Inappropriate_Library
-- if not ASIS_Libraries.Is_Open(Program_Library).


function Compilation_Order
   (Compilation_Units : in Compilation_Unit_List;
    Dependent_Units   : in Compilation_Unit_List;
    Program_Library   : in Library;
    Relation          : in Relation_Kinds )
  return Compilation_Unit_List;

-- Produces, in compilation order, the list of compilation units
-- related to the given compilation units by the specified relation.
-- The given compilation units are returned as part of the list.
--
-- Dependent units are only required when the relation is:
--   Direct_Dependents or Dependents.
--
-- The extended families of the dependent units provide a context for
-- the operation to select dependents of the given compilation units.
```

```
--
-- The return list contains the sublists described above for the
-- operations on Compilation Unit Relationships
--
-- Raises Inappropriate_Compilation_Unit
-- if any of the given compilation units or dependent units do not exist.
--
-- Raises Inappropriate_Library
-- if not ASIS_Libraries.Is_Open(Program_Library).


function Recompilation_Order
   (Compilation_Units : in Compilation_Unit_List;
    Dependent_Units   : in Compilation_Unit_List;
    Program_Library   : in Library;
    Relation          : in Relation_Kinds )
  return Compilation_Unit_List;

-- Produces, in compilation order, the list of compilation units
-- related to the given compilation units by the specified relation
-- and which are in need of recompilation.
--
-- The effects of obsolete units are reflected, transitively, in
-- the Ordered_Compilation_Units list produced.
--
-- Dependent units are only required when the relation is:
--   Direct_Dependents or Dependents.
--
-- The extended families of the dependent units provide a context for
-- the operation to select dependents of the given compilation units.
--
-- The return list contains the sublists described above for the
-- operations on Compilation Unit Relationships
--
-- Raises Inappropriate_Compilation_Unit
-- if any of the given compilation units or dependent units do not exist.
--
-- Raises Inappropriate_Library
-- if not ASIS_Libraries.Is_Open(Program_Library).


function Direct_Supporters
   (A_Compilation_Unit : in Compilation_Unit;
    Compilation_Units  : in Compilation_Unit_List)
  return Compilation_Unit_List;
```

```
function Direct_Supporters
   (Program_Library    : in Library;
    A_Compilation_Unit : in Compilation_Unit;
    Compilation_Units  : in Compilation_Unit_List)
  return Compilation_Unit_List;

-- Returns a list of units from the given Compilation_Unit_List
-- that are compiled direct supporters of Compilation_Unit.
-- Indirect (or transitive) supporters are not returned.
--
-- A compilation unit is not a direct supporter of itself.
-- Direct supporters duplicated in the given list are duplicated in
-- the returned list.
--
-- Only compilation units existing in the given library are
-- included in the return list.
--
-- Raises Inappropriate_Compilation_Unit
-- if not Exists(A_Compilation_Unit)
--
-- Raises Inappropriate_Library
-- if not ASIS_Libraries.Is_Open(Program_Library).


function Supporters
   (A_Compilation_Unit : in Compilation_Unit;
    Compilation_Units  : in Compilation_Unit_List)
  return Compilation_Unit_List;

function Supporters
   (Program_Library    : in Library;
    A_Compilation_Unit : in Compilation_Unit;
    Compilation_Units  : in Compilation_Unit_List)
  return Compilation_Unit_List;

-- Returns a list of units from the given Compilation_Unit_List
-- that are compiled supporters of Compilation_Unit.
-- Direct and indirect (or transitive) supporters are included.
--
-- A compilation unit is not a supporter of itself.
-- Supporters duplicated in the given list are duplicated in
-- the returned list.
--
-- Only compilation units existing in the given library are
```

```
-- included in the return list.
--
-- Raises Inappropriate_Compilation_Unit
-- if not Exists(A_Compilation_Unit)
--
-- Raises Inappropriate_Library
-- if not ASIS_Libraries.Is_Open(Program_Library).


function Direct_Dependents
   (A_Compilation_Unit : in Compilation_Unit;
    Compilation_Units  : in Compilation_Unit_List)
  return Compilation_Unit_List;

function Direct_Dependents
   (Program_Library    : in Library;
    A_Compilation_Unit : in Compilation_Unit;
    Compilation_Units  : in Compilation_Unit_List)
  return Compilation_Unit_List;

-- Returns a list of units from the given Compilation_Unit_List
-- that are compiled direct dependents of Compilation_Unit.
-- Indirect (or transitive) dependents are not returned.
--
-- A compilation unit is not a direct dependent of itself.
-- Direct dependents duplicated in the given list are duplicated in
-- the returned list.
--
-- Only compilation units existing in the given library are
-- included in the return list.
--
-- Raises Inappropriate_Compilation_Unit
-- if not Exists(A_Compilation_Unit)
--
-- Raises Inappropriate_Library
-- if not ASIS_Libraries.Is_Open(Program_Library).


function Dependents
   (A_Compilation_Unit : in Compilation_Unit;
    Compilation_Units  : in Compilation_Unit_List)
  return Compilation_Unit_List;

function Dependents
   (Program_Library    : in Library;
```

```
    A_Compilation_Unit : in Compilation_Unit;
    Compilation_Units  : in Compilation_Unit_List)
  return Compilation_Unit_List;

-- Returns a list of units from the given Compilation_Unit_List
-- that are compiled dependents of Compilation_Unit.
-- Direct and indirect (or transitive) dependents are included.
--
-- A compilation unit is not a dependent of itself.
-- Dependents duplicated in the given list are duplicated in
-- the returned list.
--
-- Only existing compilation units are included in the return list.
-- Only compilation units existing in the given library are
-- included in the return list.
--
-- Raises Inappropriate_Compilation_Unit
-- if not Exists(A_Compilation_Unit)
--
-- Raises Inappropriate_Library
-- if not ASIS_Libraries.Is_Open(Program_Library).


-------------------------------------------------------------------------
-- 10.4 THE PROGRAM LIBRARY - COMPILATION UNIT STATUS


function Is_Obsolete
   (A_Compilation_Unit : in Compilation_Unit)
  return Boolean;

function Is_Obsolete
   (A_Compilation_Unit : in Compilation_Unit;
    Program_Library    : in Library)
  return Boolean;

-- Returns True if Compilation_Unit exists but is obsolete due
-- to recompilation or removal of any supporters within the
-- given Library.  See LRM 10.3
--
-- It is always true that within the same library context,
-- a compilation unit is either obsolete or current, but not both.
--
-- Raises Inappropriate_Compilation_Unit
-- if not Exists(A_Compilation_Unit)
```

```
--
-- Raises Inappropriate_Library
-- if not ASIS_Libraries.Is_Open(Program_Library).


function Is_Consistent
   (A_Compilation_Unit : in Compilation_Unit)
  return Boolean;

function Is_Consistent
   (A_Compilation_Unit : in Compilation_Unit;
    Program_Library    : in Library)
  return Boolean;

-- Returns True if Compilation_Unit exists and is not obsolete.
-- See LRM 10.3
--
-- It is always true that within the same library context,
-- a compilation unit is either obsolete or consistent, but not both.
--
-- Raises Inappropriate_Compilation_Unit
-- if not Exists(A_Compilation_Unit)
--
-- Raises Inappropriate_Library
-- if not ASIS_Libraries.Is_Open(Program_Library).


-----------------------------------------------------------------------
-- 10.4 THE PROGRAM LIBRARY - COMPILATION UNIT STATUS

-- The package ASIS_LIBRARY contains the operations for
-- naming and interrogating the Ada Program Library.


-----------------------------------------------------------------------
-- 10.5 ELABORATION OF LIBRARY UNITS


function Elaboration_Order
    (Compilation_Units : in Compilation_Unit_List;
     Program_Library   : in Library)
  return Compilation_Unit_List;

-- Produces, in elaboration order, the list of compilation units
-- required to elaborate the given compilation units.
```

```
   --
   -- The return list contains the sublists described above for the
   -- operations on Compilation Unit Relationships
   --
   -- Use context clause elements to get pragma elaborate elements
   -- for a compilation unit.
   --
   -- Raises Inappropriate_Compilation_Unit
   -- if any of the given compilation units do not exist.
   --
   -- Raises Inappropriate_Library
   -- if not ASIS_Libraries.Is_Open(Program_Library).


   ------------------------------------------------------------------------
   -- COMPILATION UNIT DESTRUCTOR


   procedure Dissociate
      (A_Compilation_Unit : in out Compilation_Unit);

   -- Severs associations made to the Compilation_Unit.
   -- A Nil_Compilation_Unit value is returned.
   --
   -- Postconditions:  (if no exception is raised)
   --   Depending on the ASIS implementation, storage for Compilation_Unit
   --    may be deallocated.


   procedure Dissociate
      (A_Compilation_Unit_List : in out Compilation_Unit_List);

   -- Severs associations made to the Compilation_Unit_List.
   -- A Nil_Compilation_Unit_List value is returned.
   --
   -- Postconditions:  (if no exception is raised)
   --   Depending on the ASIS implementation, storage for each Compilation_Unit
   --    may be deallocated.


   ------------------------------------------------------------------------
   -- ERROR INFORMATION

   type ASIS_Compilation_Unit_Error_Kinds is
```

```
   (Not_An_Error,

    Status_Error,              -- A operation that takes a library parameter
                               -- was given a library that was not open.

    TBD);

  -- Additional Error Kind literals are yet TO BE DETERMINED

  -- Whenever an error condition is detected and the exception Failed
  -- is raised, an ASIS_Compilation_Unit_Error_Kinds value is stored for the
  -- Compilation_Unit involved in the failure.   That value can be retrieved
  -- using the Status function.  The Diagnosis function will retrieve the
  -- ASIS_String diagnostic message for the Compilation_Unit_Error_Kinds value.
  --
  -- Error information is only kept when the catch-all exception Failed
  -- is raised.
  --
  -- Note that Status and Diagnosis values are vendor dependent and will
  -- vary among ASIS implementations.


  function Status (A_Compilation_Unit : in Compilation_Unit)
    return ASIS_Compilation_Unit_Error_Kinds;

  -- Returns the ASIS_Compilation_Unit_Error_Kinds value associated with
  -- the compilation unit.
  --
  -- Returns Not_An_Error if the compilation unit has no error kind associated
  -- with it, or if Is_Nil(A_Compilation_Unit).
  -- EG. It has not been involved in a failed operation.


  function Diagnosis (A_Compilation_Unit : in Compilation_Unit)
    return ASIS_String;

  -- Returns an ASIS_String value containing implementation defined error
  -- information associated with the compilation unit.


  procedure Dissociate_Compilation_Unit_Error_Information;

  -- Removes the association of status and diagnosis error information with
  -- all Compilation_Unit handles that have been involved in a failed
  -- operation.
```

```
  --
  -- This operation does not affect Compilation_Unit handles in any other way.


  ------------------------------------------------------------------------
  -- DEBUGGING

  function Debug_Image (A_Compilation_Unit : in Compilation_Unit)
    return ASIS_String;

  -- Returns an ASIS_String value containing implementation defined debug
  -- information associated with the compilation unit.


private


end ASIS_Compilation_Units;
```

```
------------------------------------------------------------------------
-- ASIS PRELIMINARY REVIEW VERSION 0.4
-- Sept 27, 1991
------------------------------------------------------------------------


with ASIS_Ada_Program;
package ASIS_Elements is

  -- LRM Chapter 2

  subtype Element is ASIS_Ada_Program.Element;

  subtype Element_List is ASIS_Ada_Program.Element_List;

  -- ASIS Elements are handles to the lexical elements that form the text
```

```
-- of compilation units.
--
-- ASIS Elements are either terminal elements or composite elements made
-- up of component elements.  For example, a simple name is a terminal element
-- while a selected component is a composite element consisting of three
-- components: a prefix, a dot, and a selector.  A lexical element must fit
-- on one line of text but a composite element could span many lines.
--
-- ASIS defines a variety of element subtypes that identify classes of
-- elements.  These subtypes classify elements into declarations, statements,
-- expressions, type definitions, and several other classes.
--
-- For each element class, ASIS provides enumeration types that describe the
-- kinds of elements.  For example, statements can be if statements, loop
-- statements, raise statements, or any other kind of statement.
-- Operations are defined to determine the kind for elements of a
-- particular class.
--
-- ASIS offers many operations to decompose the various classes of composite
-- elements into constituent component elements.
--
-- Other ASIS operations return semantically related elements such as the
-- type of a given object or the specification of a given body.
--
-- Each ASIS Element may also have attributes that provide information
-- about that element.  For example, all declaration elements will
-- have a name (identifier) associated with them.  Certain static
-- expressions may have a value associated with them.  Operations
-- are defined to provide this information for specific kinds of
-- elements.
--
-- Some elements "make reference to" other elements.  Declarations,
-- for example, define named elements of the class Identifier_Definition.
-- Other elements such as statements and expressions may make
-- reference to named elements.  These references are handled with
-- elements of the class Identifier_Reference.
--
-- Each ASIS Element has a text image whose value is a series of characters
-- that forms 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 the range of lines.
--

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

-- EXCEPTIONS
```

```
Inappropriate_Program_Element : exception renames
    ASIS_Ada_Program.Inappropriate_Program_Element;

-- All operations may raise Inappropriate_Program_Element whenever they
-- are given input that is not valid.  It is generally inappropriate to
-- mix elements of distinct subtypes.  EG. Passing a statement to an
-- operation expecting a declaration is inappropriate.
-- It is also inappropriate to mix kinds of elements within a subtype when
-- an operation is expecting a specific kind.  EG. Passing a type
-- declaration to an operation looking for a procedure declaration is
-- inappropriate.


Failed : exception renames ASIS_Ada_Program.Failed;

-- All operations may raise Failed whenever they cannot normally complete
-- their operation.
-- This is a catch-all exception that may be raised for different reasons
-- in different ASIS implementations.
--
-- The Diagnostic and Status functions are provided so more information
-- about the failure can be retrieved.


-------------------------------------------------------------------------
-- ELEMENT ASSIGNMENT

procedure Assign (Target : in out Element; Source : in Element);

-- Assigns the source element to the target element.
--
-- Postconditions:  (if no exception is raised)
--   Depending on the ASIS implementation, the target element
--    may be dissociated before performing the assignment.
--   Is_Equal(Target, Source)


-------------------------------------------------------------------
-- ELEMENT ATTRIBUTES

function Is_Nil (Program_Element : in Element) return Boolean;

-- Returns True if the program element has a void or null value.
--
```

```
-- All element kinds are appropriate.


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

-- Returns True if Left and Right represent the same element.
--
-- All element kinds are appropriate.


------------------------------------------------------------------
-- MAJOR ELEMENT KINDS

type Major_Element_Kinds is (

  An_Identifier,                -- LRM 2.3     ASIS_Names_And_Expressions
  A_Pragma,                     -- LRM 2.8     ASIS_Elements
  An_Argument_Association,      -- LRM 2.8     ASIS_Statements
  A_Declaration,                -- LRM 3       ASIS_Declarations
  A_Type_Definition,            -- LRM 3.3.1   ASIS_Type_Definitions
  A_Subtype_Indication,         -- LRM 3.3.2   ASIS_Type_Definitions
  A_Constraint,                 -- LRM 3.3.2   ASIS_Type_Definitions
  A_Discrete_Range,             -- LRM 3.6     ASIS_Type_Definitions
  A_Discriminant_Association,   -- LRM 3.7.2   ASIS_Type_Definitions
  A_Variant_Part,               -- LRM 3.7.3   ASIS_Type_Definitions
  A_Null_Component,             -- LRM 3.7.3   ASIS_Type_Definitions
  A_Variant,                    -- LRM 3.7.3   ASIS_Type_Definitions
  A_Choice,                     -- LRM 3.7.3   ASIS_Type_Definitions
  A_Name_Expression,            -- LRM 4.1     ASIS_Names_And_Expressions
  A_Component_Association,      -- LRM 4.3     ASIS_Names_And_Expressions
  An_Expression,                -- LRM 4.4     ASIS_Names_And_Expressions
  A_Statement,                  -- LRM 5.1     ASIS_Statements
  An_If_Statement_Arm,          -- LRM 5.3     ASIS_Statements
  A_Case_Statement_Alternative, -- LRM 5.4     ASIS_Statements
  A_Parameter_Association,      -- LRM 6.4     ASIS_Statements
  A_Use_Clause,                 -- LRM 8.4     ASIS_Compilation_Units
  A_Select_Statement_Arm,       -- LRM 9.7     ASIS_Statements
  A_Select_Alternative,         -- LRM 9.7.1   ASIS_Statements
  A_With_Clause,                -- LRM 10.1.1  ASIS_Compilation_Units
  An_Exception_Handler,         -- LRM 11.2    ASIS_Statements
  A_Representation_Clause,      -- LRM 13.1    ASIS_Representation_Clauses
  A_Component_Clause,           -- LRM 13.4    ASIS_Representation_Clauses

  Not_A_Major_Element);
```

```
function Major_Element_Kind (Program_Element : in Element)
  return Major_Element_Kinds;

-- Returns the Major_Element_Kind of the element.
-- Returns Not_A_Major_Element for a Nil_Element.
--
-- All element kinds are appropriate.
--
-- Once the Major_Element_Kind of an element is determined,
-- further decomposition or analysis can be done by calling functions
-- in the ASIS package that deals with a specific element kind.


-------------------------------------------------------------------
-- MAJOR ELEMENTS

type Parse_Control is (Continue,
Abandon_Children,
Abandon_Siblings,
Terminate_Immediately);

generic

  with procedure Pre_Operation
   (Ada_Element       : in ASIS_Ada_Program.Element;
    Control           : in out Parse_Control) is <>;

  with procedure Post_Operation
   (Ada_Element       : in ASIS_Ada_Program.Element;
    Control           : in out Parse_Control) is <>;

procedure Parse_Major_Element
 (Ada_Element       : in ASIS_Ada_Program.Element;
  Control           : in out Parse_Control);

-- Parses the element and all its component elements.  For each element,
-- the formal procedure Pre_Operation is called when first visiting the
-- element.  All component elements are then visited and finally the formal
-- procedure Post_Operation is called when returning from visiting all
-- component elements.
--
-- Parsing can be controlled with the Control parameter.
-- The Abandon_Children control prevents parsing of the current element's
```

```
-- child components, but picks up with the next sibling element.
-- The Abandon_Siblings control abandons parsing of the remaining siblings
-- but continues parsing at the parent.
-- The Terminate_Immediately control does just that.
--
-- Abandon_Children in a Post_Operation is the same as Continue as all the
-- children have been parsed.
-- Abandon_Siblings in a Pre_Operation skips the associated
-- Post_Operation.
--
-- Raises Inappropriate_Program_Element
-- if Major_Element_Kind(Ada_Element) = Not_A_Major_Element

-------------------------------------------------------------------
-- ENCLOSING MAJOR ELEMENTS

function Enclosing_Major_Element (Program_Element : in Element)
  return Element;

function Enclosing_Major_Element
   (Program_Element : in Element; Context : in Element)
  return Element;

-- Returns the Major Element that immediately encloses the given element.
-- Some elements know what their enclosing major element is regardless of
-- a given context.  For other elements, the enclosing major element may
-- have to be determined by performing a search.  The context parameter
-- provides the element that is searched and thus can be used to make this
-- operation more efficient.  However, if the Program_Element is not a
-- component element of the Context element and the enclosing element
-- cannot be determined without a search, then a Nil_Element is returned.
--
-- A nil element is also returned if the element is the declaration part
-- of a compilation unit. (EG. The ASIS_Compilation_Units.Unit_Declaration
-- of the compilation unit).
--
-- Use ASIS_Compilation_Units.Enclosing_Compilation_Unit to get the
-- enclosing compilation unit for an element.
--
-- Raises Inappropriate_Program_Element if Is_Nil(Program_Element)
--
-- Examples:
--
--   Given a type declaration in a declare block, returns the block
--   statement element that encloses the type declaration.
```

```
--
--   Given a statement within the sequence of statements of a loop
--   statement, returns the enclosing loop statement.
--
--   Given an Identifier_Reference as the selector in an expanded name,
--   returns the Name_Expression representing the prefix, the dot, and
--   the selector.


------------------------------------------------------------------------
-- PRAGMAS - LRM 2.8

subtype Pragma_List is ASIS_Ada_Program.Element_List;

function Pragmas (Context : in Element) return Pragma_List;

-- Returns the list of pragmas appearing within the given context.
--
-- A Nil_Element_List is returned if there are no pragmas.
--
-- Raises Inappropriate_Program_Element
-- if Major_Element_Kind(Context) = Not_A_Major_Element


subtype Pragma_Element is ASIS_Ada_Program.Element;

function Is_Predefined (A_Pragma : in Pragma_Element) return Boolean;

-- Returns True if the pragma is one of the predefined language pragmas.
-- See the LRM Appendix B.
--
-- Raises Inappropriate_Program_Element
-- if Major_Element_Kind(A_Pragma) /= A_Pragma


function Name (A_Pragma : in Pragma_Element) return String;

-- Returns the identifier name of any pragma.
--
-- Raises Inappropriate_Program_Element
-- if Major_Element_Kind(A_Pragma) /= A_Pragma


subtype Argument_Association_List is ASIS_Ada_Program.Element_List;
```

```
function Argument_Associations (A_Pragma : in Pragma_Element)
  return Argument_Association_List;

-- Returns a list of the argument associations of the pragma.
--
-- Raises Inappropriate_Program_Element
-- if Major_Element_Kind(A_Pragma ) /= A_Pragma
--
-- Further Analysis:
--   Pragma argument associations take the same form as actual parameter
--   associations of a procedure call.
--   The ASIS_Statements operation Formal_Parameter will extract the
--   argument identifier.
--   The ASIS_Statements operation Actual_Parameter will extract the
--   name or expression.



-------------------------------------------------------------------------
-- ELEMENT DESTRUCTOR

procedure Dissociate (Program_Element: in out Element);

-- A Nil_Element value is returned.
--
-- Postconditions:  (if no exception is raised)
--   Depending on the ASIS implementation, storage for the element
--    may be deallocated.
--   Program_Element = Nil_Element



procedure Dissociate
   (Program_Element_List : in out Element_List);

-- Severs associations made to the Element_List.
-- A Nil_Element_List value is returned.
--
-- Postconditions:  (if no exception is raised)
--   Depending on the ASIS implementation, storage for each Element
--    may be deallocated.


-------------------------------------------------------------------
-- ERROR INFORMATION
```

```
type ASIS_Element_Error_Kinds is (Not_An_Error, TBD);
-- Additional Error Kind literals are yet TO BE DETERMINED


-- Whenever an error condition is detected and the exception Failed
-- is raised, an ASIS_Element_Error_Kinds value is stored for the Element
-- involved in the failure.   That value can be retrieved using the Status
-- function.  The Diagnosis function will retrieve the ASIS_String
-- diagnostic message for the Element_Error_Kinds value.
--
-- Error information is only kept when the catch-all exception Failed
-- is raised.
--
-- Note that Status and Diagnosis values are vendor dependent and will
-- vary among ASIS implementations.


function Status (Program_Element : in Element)
  return ASIS_Element_Error_Kinds;

-- Returns the ASIS_Element_Unit_Error_Kinds value associated with
-- the element.
--
-- Returns Not_An_Error if the element has no error kind associated
-- with it, or if Is_Nil(Program_Element)
-- EG. It has not been involved in a failed operation.


function Diagnosis (Program_Element : in Element)
  return ASIS_Ada_Program.ASIS_String;

-- Returns an ASIS_String value containing implementation defined error
-- information associated with the element.


procedure Dissociate_Element_Error_Information;

-- Removes the association of status and diagnosis error information with
-- all Element handles that have been involved in a failed operation.
--
-- This operation does not affect Element handles in any other way.


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

```
  -- DEBUGGING

  function Debug_Image (Program_Element : in Element)
    return ASIS_Ada_Program.ASIS_String;

  -- Returns an ASIS_String value containing implementation defined debug
  -- information associated with the element.


end ASIS_Elements;
```

```
-------------------------------------------------------------------------
-- ASIS PRELIMINARY REVIEW VERSION 0.4
-- Sept 27, 1991
-------------------------------------------------------------------------


with ASIS_Ada_Program;
package ASIS_Declarations is

  -- LRM Chapters 3, 6, 7, 8, 9, 11, 12

  subtype Element is ASIS_Ada_Program.Element;
    -- Use ASIS_Elements operations.

  subtype Declaration is ASIS_Ada_Program.Declaration;
    -- Use ASIS_Declarations operations.

  subtype Expression is ASIS_Ada_Program.Expression;
    -- Use ASIS_Names_And_Expression operations.

  subtype Name_Expression is ASIS_Ada_Program.Name_Expression;
    -- Use ASIS_Names_And_Expression operations.

  subtype Statement is ASIS_Ada_Program.Statement;
    -- Use ASIS_Statements operations.

  subtype Type_Definition is ASIS_Ada_Program.Type_Definition;
```

```
  -- Use ASIS_Type_Definitions operations.

subtype Task_Specification is Type_Definition;
  -- Use operations in this package and in ASIS_Type_Definitions.


subtype Library is ASIS_Ada_Program.Library;

-------------------------------------------------------------
-- DECLARATION KINDS


type Declaration_Kinds is

-- Declaration_Kinds include all declarations and specifications that
-- declare named entities.

--   Declarations with identifier lists can
--   declare one or more identifiers:

   (A_Variable_Declaration,
    A_Component_Declaration,

    A_Constant_Declaration,
    A_Deferred_Constant_Declaration,

    A_Generic_Formal_Object_Declaration,

    A_Discriminant_Specification,

    A_Parameter_Specification,

    An_Integer_Number_Declaration,
    A_Real_Number_Declaration,

    An_Exception_Declaration,


--   Declarations with a single identifier or designator:

    An_Enumeration_Literal_Specification,
    A_Loop_Parameter_Specification,

    A_Full_Type_Declaration,
    An_Incomplete_Type_Declaration,
```

```
A_Private_Type_Declaration,

A_Subtype_Declaration,

A_Package_Declaration,
A_Package_Body_Declaration,

A_Procedure_Declaration,
A_Procedure_Body_Declaration,

A_Function_Declaration,
A_Function_Body_Declaration,

An_Object_Rename_Declaration,
An_Exception_Rename_Declaration,
A_Package_Rename_Declaration,
A_Procedure_Rename_Declaration,
A_Function_Rename_Declaration,

A_Generic_Package_Declaration,
A_Generic_Procedure_Declaration,
A_Generic_Function_Declaration,

A_Package_Instantiation,
A_Procedure_Instantiation,
A_Function_Instantiation,

A_Task_Declaration,
A_Task_Body_Declaration,
A_Task_Type_Declaration,

An_Entry_Declaration,

A_Procedure_Body_Stub,
A_Function_Body_Stub,
A_Package_Body_Stub,
A_Task_Body_Stub,

A_Generic_Formal_Type_Declaration,
A_Generic_Formal_Private_Type_Declaration,
A_Generic_Formal_Procedure_Declaration,
A_Generic_Formal_Function_Declaration,

Not_A_Declaration);
```

```
function Kind (A_Declaration : in Declaration) return Declaration_Kinds;

-- Returns the kind of the given declaration.
--
-- Returns Not_A_Declaration for any inappropriate element such as
-- a Statement, Expression, or Nil_Element.


-------------------------------------------------------------------
-- IDENTIFIER 2.3
--
-- IDENTIFIER LIST 3.2
--   IN A COMPONENT DECLARATION 3.7
--   IN A DEFERRED CONSTANT DECLARATION 7.4
--   IN A DISCRIMINANT SPECIFICATION 3.7.1
--   IN A GENERIC PARAMETER DECLARATION FOR GENERIC FORMAL OBJECTS 3.7.1
--   IN AN OBJECT DECLARATION 3.2
--   IN A NUMBER DECLARATION 3.2
--   IN A PARAMETER SPECIFICATION 6.1


subtype Identifier_Definition_List is ASIS_Ada_Program.Element_List;

function Identifiers (A_Declaration : in Declaration)
  return Identifier_Definition_List;

-- Returns a list of each Identifier_Definition defined by the declaration.
-- For declarations that define a single identifier, the list returned
-- always has a length of one.
--
-- Examples:
--   type Foo is new Integer;
--        --- Returns a list containing one Identifier_Definition Foo.
--
--   One, Uno : constant Foo := 1;
--   ---  --- Returns a list of two Identifier_Definitions One and Uno.
--
-- Results of this operation may vary across ASIS implementations.
-- Some implementations normalize all multiple object declarations into
-- an equivalent sequence of corresponding single object declarations.
-- See LRM 3.2(10).  If this is the case, then this operation will
-- return a list containing a single identifier.
--
-- Raises Inappropriate_Program_Element
```

```
-- if Kind(A_Declaration) = Not_A_Declaration


------------------------------------------------------------
-- OBJECT DECLARATIONS - LRM 3.2.1

function Is_Initialized (Object_Declaration : in Declaration)
  return Boolean;

-- Returns True if the declaration has an explicit initialization.
--
-- Appropriate Declaration Kinds:
--   A_Variable_Declaration,
--   A_Component_Declaration,
--   A_Constant_Declaration,
--   A_Discriminant_Specification,
--   A_Parameter_Specification,
--   An_Integer_Number_Declaration,
--   A_Real_Number_Declaration,
--   A_Generic_Formal_Object_Declaration


function Initial_Value (Object_Declaration : in Declaration)
  return Expression;

-- If Is_Initialized(Object_Decl) then
--   Returns the expression that initializes the declaration.
-- else returns Nil_Element.
--
-- Appropriate Declaration Kinds:
--   All kinds appropriate for Is_Initialized.


function Is_Variable (Object_Declaration : in Declaration) return Boolean;
function Is_Constant (Object_Declaration : in Declaration) return Boolean;

-- Variable, discriminant, and component declarations are always variables.
-- Constants and deferred constants are always constant.
-- A renamed object is a constant if the renamed entity is a constant.
-- A loop parameter specification is always a constant during each
--   iteration of the loop.
-- A number declaration is always a constant.
-- A subprogram formal parameter or a generic formal object is constant
--   if it is mode in.
--
```

```
-- Appropriate Declaration Kinds:
--   A_Variable_Declaration,
--   A_Discriminant_Specification,
--   A_Component_Declaration,
--   A_Constant_Declaration,
--   A_Deferred_Constant_Declaration,
--   An_Object_Rename_Declaration,
--   An_Integer_Number_Declaration,
--   A_Real_Number_Declaration,
--   A_Parameter_Specification,
--   A_Generic_Formal_Object_Declaration



function Object_Declaration_Definition (Object_Declaration : in Declaration)
 return Type_Definition;

-- Returns the subtype indication or the constrained array definition
-- following the colon in the object declaration.
--
-- Appropriate Declaration Kinds:
--   A_Variable_Declaration,
--   A_Component_Declaration,
--   A_Constant_Declaration,


function Corresponding_Constant_Declaration
   (Constant_Declaration : in Declaration)
 return Declaration;

-- Returns the corresponding full constant declaration when given
-- a deferred constant declaration, or;
-- Returns the corresponding deferred constant declaration when given
-- a full constant declaration.
--
-- A Nil_Element is returned when a full constant declaration that has no
-- corresponding deferred constant declaration is given.
--
-- Appropriate Declaration Kinds:
--   A_Constant_Declaration,
--   A_Deferred_Constant_Declaration,


------------------------------------------------------------------
-- TYPE DECLARATIONS - LRM 3.3
```

```
function Is_Type_Declaration (A_Declaration : in Declaration)
  return Boolean;

-- Returns True for declarations that are type or subtype declarations.
--
-- Raises Inappropriate_Program_Element
-- if Kind(A_Declaration) = Not_A_Declaration


function Is_Private (Type_Declaration : in Declaration) return Boolean;

-- Returns True if the type declaration is:
--   A_Private_Type_Declaration,
--   A_Generic_Formal_Private_Type_Declaration,
--   A_Full_Type_Declaration of a type derived from a private type,
--   A_Full_Type_Declaration of a composite type with subcomponents
--     of a private type.
--
-- Raises Inappropriate_Program_Element
-- if not Is_Type_Declaration(Type_Declaration)


function Is_Limited (Type_Declaration : in Declaration) return Boolean;

-- Returns True if the type declaration is:
--   A_Full_Type_Declaration of a type derived from a limited type,
--   A_Full_Type_Declaration of a composite type with subcomponents
--     of a limited type,
--   A_Task_Type_Declaration,
--   A_Private_Type_Declaration that includes the reserved word limited,
--   A_Generic_Formal_Private_Type_Declaration, that includes the
--     reserved word limited.
--
-- Raises Inappropriate_Program_Element
-- if not Is_Type_Declaration(Type_Declaration)


function Is_Discriminated (Type_Declaration : in Declaration)
  return Boolean;

-- Returns True if the type declaration has discriminants.
--
-- Raises Inappropriate_Program_Element
-- if not Is_Type_Declaration(Type_Declaration)
```

```
subtype Discriminant_Specification_List is ASIS_Ada_Program.Element_List;

function Discriminants (Type_Declaration : in Declaration)
  return Discriminant_Specification_List;

-- Returns a list of discriminant specifications in their order of
-- appearance.
--
-- Results of this operation may vary across ASIS implementations.
-- Some implementations normalize all multiple discriminant specifications
-- into an equivalent sequence of corresponding single discriminant
-- specifications.  See LRM 3.7.1(4).
--
-- Further Analysis:
--   Use Identifiers to obtain the identifier list.
--   Use Type_Mark to obtain the parameter type mark.
--   Use Is_Initialized and Initial_Value to query the information
--     related to the presence of the default parameter initialization.
--
-- Appropriate Declaration Kinds:
--   A_Full_Type_Declaration
--   An_Incomplete_Type_Declaration
--   A_Private_Type_Declaration
--   A_Generic_Formal_Private_Type_Declaration


function Type_Declaration_Definition (Type_Declaration : in Declaration)
  return Type_Definition;

-- Returns the type definition following the reserved word is in the
-- type or subtype declaration.
--
-- Returns a Nil_Element if given An_Incomplete_Type_Declaration.
--
-- Appropriate Declaration Kinds:
--   A_Subtype_Declaration
--   A_Full_Type_Declaration
--   An_Incomplete_Type_Declaration
--   A_Private_Type_Declaration
--   A_Task_Type_Declaration
--   A_Generic_Formal_Type_Declaration
--   A_Generic_Formal_Private_Type_Declaration,
```

```
function Corresponding_Type_Declaration
   ( Type_Declaration : in Declaration) return Declaration;

function Corresponding_Type_Declaration
   ( Type_Declaration : in Declaration;
     Program_Library  : in Library)
 return Declaration;

-- Returns the corresponding full type declaration when given a private
-- or incomplete type declaration, or;
-- Returns the corresponding private or incomplete type declaration when
-- given a full type declaration.
--
-- A Nil_Element is returned when a full type declaration that has no
-- corresponding private or incomplete type declaration is given.
--
-- A Nil_Element is returned when a corresponding type declaration
-- does not exist within the given context.
--
-- The Library parameter is necessary whenever the corresponding full
-- type of an incomplete type is in a corresponding package body.
-- See LRM 3.8.1(3).
--
-- Appropriate Declaration Kinds:
--   A_Full_Type_Declaration
--   An_Incomplete_Type_Declaration
--   A_Private_Type_Declaration
--   A_Task_Type_Declaration


------------------------------------------------------------------
-- ENUMERATION LITERALS - LRM 3.5.1

-- Use ASIS_Type_Definitions.Enumeration_Literals to analyze the list of
-- enumeration literals in the type definition part of an enumeration type
-- declaration.

function Is_Character_Literal
   (Enumeration_Literal_Specification : in Declaration)
  return Boolean;

-- Returns True if the enumeration literal is a character literal.
--
-- Appropriate Declaration Kinds:
```

```
--    An_Enumeration_Literal_Specification


-------------------------------------------------------------
-- PROGRAM UNIT DECLARATIONS
-- LRM Chapters 6, 7, 9

-- Program units are package, subprogram, and task specifications
-- and bodies.


function Is_Spec (A_Declaration : in Declaration) return Boolean;

-- Determines whether a package, procedure, function, or task is a
-- specification.
--
-- Returns True if the declaration is:
--    A_Package_Declaration,
--    A_Procedure_Declaration,
--    A_Function_Declaration,
--    A_Package_Rename_Declaration,
--    A_Procedure_Rename_Declaration,
--    A_Function_Rename_Declaration,
--    A_Generic_Package_Declaration,
--    A_Generic_Procedure_Declaration,
--    A_Generic_Function_Declaration,
--    A_Package_Instantiation,
--    A_Procedure_Instantiation,
--    A_Function_Instantiation,
--    A_Task_Declaration,
--    A_Task_Type_Declaration,
--    An_Entry_Declaration,
--    A_Generic_Formal_Procedure_Declaration
--    A_Generic_Formal_Function_Declaration
--
-- Raises Inappropriate_Program_Element
-- if Kind(A_Declaration) = Not_A_Declaration


function Is_Package (A_Declaration : in Declaration) return Boolean;

-- Returns True if the declaration is:
--    A_Package_Declaration,
--    A_Package_Body_Declaration,
--    A_Package_Rename_Declaration,
```

```
--   A_Generic_Package_Declaration,
--   A_Package_Instantiation,
--   A_Package_Body_Stub,
--
-- Raises Inappropriate_Program_Element
-- if Kind(A_Declaration) = Not_A_Declaration


function Is_Task (A_Declaration : in Declaration) return Boolean;

-- Returns True if the declaration is:
--   A_Task_Body_Stub,
--   A_Task_Declaration,
--   A_Task_Body_Declaration,
--   A_Task_Type_Declaration,
--
-- Raises Inappropriate_Program_Element
-- if Kind(A_Declaration) = Not_A_Declaration


function Is_Procedure (A_Declaration : in Declaration) return Boolean;

-- Returns True if the declaration is:
--   A_Procedure_Declaration,
--   A_Procedure_Body_Declaration,
--   A_Procedure_Rename_Declaration,
--   A_Generic_Procedure_Declaration,
--   A_Procedure_Instantiation,
--   A_Procedure_Body_Stub,
--   A_Generic_Formal_Procedure_Declaration
--
-- Raises Inappropriate_Program_Element
-- if Kind(A_Declaration) = Not_A_Declaration


function Is_Function (A_Declaration : in Declaration) return Boolean;

-- Returns True if the declaration is:
--   A_Function_Declaration,
--   A_Function_Body_Declaration,
--   A_Function_Rename_Declaration,
--   A_Generic_Function_Declaration,
--   A_Function_Instantiation,
--   A_Function_Body_Stub,
--   A_Generic_Formal_Function_Declaration
```

```
--
-- Raises Inappropriate_Program_Element
-- if Kind(A_Declaration) = Not_A_Declaration


function Is_Subprogram (A_Declaration : in Declaration) return Boolean;

-- Determines if a declaration is a subprogram declaration regardless
-- if it is generic, an instantiation, rename, spec or body.
--
-- Raises Inappropriate_Program_Element
-- if Kind(A_Declaration) = Not_A_Declaration


function Unit_Specification (Program_Unit_Declaration : in Declaration)
  return Declaration;

-- Returns the corresponding specification of a program unit declaration.
--
-- If a specification declaration is given, the same element is returned.
-- If no explicit specification exists, or the declaration is the proper
-- body of a subunit, a nil element is returned.
--
-- If a generic instantiation is given, the same element is returned.
--
-- Appropriate Declaration Kinds returning a specification:
--   A_Package_Body_Declaration,
--   A_Procedure_Body_Declaration,
--   A_Function_Body_Declaration,
--   A_Task_Body_Declaration,
--   A_Procedure_Body_Stub,
--   A_Function_Body_Stub,
--   A_Package_Body_Stub,
--   A_Task_Body_Stub,
--
-- Appropriate Declaration Kinds returning the same element:
--   Package_Declaration,
--   A_Procedure_Declaration,
--   A_Function_Declaration,
--   A_Package_Rename_Declaration,
--   A_Procedure_Rename_Declaration,
--   A_Function_Rename_Declaration,
--   A_Generic_Package_Declaration,
--   A_Generic_Procedure_Declaration,
--   A_Generic_Function_Declaration,
```

```
--   A_Package_Instantiation,
--   A_Procedure_Instantiation,
--   A_Function_Instantiation,
--   A_Task_Declaration,
--   A_Task_Type_Declaration,



function Unit_Body (Program_Unit_Declaration : in Declaration)
  return Declaration;

function Unit_Body (Program_Unit_Declaration : in Declaration;
                    Program_Library          : in Library)
  return Declaration;

-- Returns the corresponding body for a given program unit declaration.
--
-- If a body declaration is given, the same element is returned.
-- If no body exists, a nil element is returned.
--
-- If a generic instantiation is given, the same element is returned.
--
-- The Library parameter is necessary whenever the unit body is located
-- in a different compilation unit than the declaration of the
-- specification.
--
-- Appropriate Declaration Kinds returning a body:
--   Package_Declaration,
--   A_Procedure_Declaration,
--   A_Function_Declaration,
--   A_Generic_Package_Declaration,
--   A_Generic_Procedure_Declaration,
--   A_Generic_Function_Declaration,
--   A_Task_Declaration,
--   A_Task_Type_Declaration,
--
-- Appropriate Declaration Kinds returning the same element:
--   A_Package_Body_Declaration,
--   A_Procedure_Body_Declaration,
--   A_Function_Body_Declaration,
--   A_Package_Rename_Declaration,
--   A_Procedure_Rename_Declaration,
--   A_Function_Rename_Declaration,
--   A_Package_Instantiation,
--   A_Procedure_Instantiation,
```

```
--    A_Function_Instantiation,
--    A_Task_Body_Declaration,
--    A_Procedure_Body_Stub,
--    A_Function_Body_Stub,
--    A_Package_Body_Stub,
--    A_Task_Body_Stub,



------------------------------------------------------------
-- SUBPROGRAMS
-- LRM Chapter 6

-- PARAMETER_SPECIFICATION - LRM 6.1


subtype Parameter_Specification is Declaration;

type Parameter_Kinds is

   (A_Default_In_Parameter,
    An_In_Parameter,
    An_Out_Parameter,
    An_In_Out_Parameter,
    Not_A_Parameter);

function Parameter_Kind (Parameter : in Parameter_Specification)
            return Parameter_Kinds;

-- Returns the kind of the given parameter specification.
-- This operation also returns the mode for generic formal object
-- declarations.
--
-- Returns Not_A_Parameter for any inappropriate element such as
-- a Statement, Expression, or Nil_Element.
--
-- Results of this operation may vary across ASIS implementations.
-- Some implementations will not differentiate between An_In_Parameter
-- and A_Default_In_Parameter.


subtype Parameter_Specification_List is ASIS_Ada_Program.Element_List;

function Parameters (Subprogram_Or_Entry : in Declaration)
  return Parameter_Specification_List;
```

```
-- Returns a list of parameter specifications in the formal part
-- of the subprogram or entry declaration in the order of appearance.
--
-- A Nil_Element_List is returned if the subprogram or entry has no
-- parameters.
--
-- Results of this operation may vary across ASIS implementations.
-- Some implementations normalize all multiple parameter specifications into
-- an equivalent sequence of corresponding single parameter specifications.
-- See LRM 6.1(4).
--
-- Further Analysis:
--   Use Identifiers to obtain the identifier list.
--   Use Type_Mark to obtain the parameter type mark.
--   Use Is_Initialized and Initial_Value to query the information
--     related to the presence of the default parameter initialization.
--
-- Appropriate Declaration Kinds:
--   Any declaration for which Is_Subprogram returns True, or
--   An_Entry_Declaration,


function Return_Type (Function_Declaration : in Declaration)
  return Name_Expression;

-- Returns the Name_Expression of the return type.
--
-- Appropriate Declaration Kinds:
--   Any declaration for which Is_Function returns True


function Is_Operator_Definition (Function_Declaration : in Declaration)
 return Boolean;

-- Returns True if the function designator is an operator symbol.
--
-- Appropriate Declaration Kinds:
--   Any declaration for which Is_Function returns True


function Subprogram_Body_Block (Subprogram_Body : in Declaration)
  return Statement;

-- Returns the block statement for the body including the declarative
```

```
-- part, sequence of statements, and exception handlers, if any.
-- A Nil_Element is returned if the body is a body stub.
--
-- Appropriate Declaration Kinds:
--   A_Procedure_Body_Declaration,
--   A_Function_Body_Declaration,
--   A_Procedure_Instantiation,
--   A_Function_Instantiation,
--   A_Procedure_Body_Stub,
--   A_Function_Body_Stub,


-------------------------------------------------------------------
-- TYPE_MARK
--   IN A PARAMETER SPECIFICATION 6.1
--   IN A DEFERRED CONSTANT DECLARATIONS 7.4
--   IN A RENAMING_DECLARATION 8.5
--   IN A DISCRIMINANT SPECIFICATION 3.7.1
--   IN A GENERIC PARAMETER DECLARATION 12.1


function Type_Mark (A_Declaration : in Declaration)
  return Name_Expression;

-- Returns the type mark associated with the declaration.
--
-- Appropriate Declaration Kinds:
--   A_Parameter_Specification
--   A_Deferred_Constant_Declaration
--   A_Discriminant_Specification
--   An_Object_Rename_Declaration
--   A_Generic_Formal_Object_Declaration


-----------------------------------------------------------
-- PACKAGES
-- L.M Chapter 7


function Is_Visible (A_Declaration : in Declaration) return Boolean;

-- Returns True for units that are library level package or subprogram
-- specs, and for visible declarative items inside library level package
-- specs.
--
```

```
-- Raises Inappropriate_Program_Element
-- if Kind(A_Declaration) = Not_A_Declaration


function Is_Library_Unit_Declaration (A_Declaration : in Declaration)
  return Boolean;

-- Returns True for declarations that are library units and
-- for declarations of secondary units that are interpreted as
-- library units.  LRM 10.1(6).
--
-- Raises Inappropriate_Program_Element
-- if Kind(A_Declaration) = Not_A_Declaration


function Is_In_Visible_Part (A_Declaration : in Declaration) return Boolean;

-- Returns true if the declaration is a declarative item in the visible
-- part of a library package.
--
-- Raises Inappropriate_Program_Element
-- if Kind(A_Declaration) = Not_A_Declaration


function Is_In_Private_Part (A_Declaration : in Declaration) return Boolean;

-- Returns true if the declaration is a declarative item in the private
-- part of a library package.
--
-- Raises Inappropriate_Program_Element
-- if Kind(A_Declaration) = Not_A_Declaration


subtype Declarative_Item_List is ASIS_Ada_Program.Element_List;

function Visible_Part_Declarative_Items
   (Package_Specification : in Declaration)
  return Declarative_Item_List;

-- Returns a list of all basic declarations, representation specifications,
-- and use clauses in the visible part of a package in the order of
-- appearance.
--
-- Results of this operation may vary across ASIS implementations.
-- Some implementations normalize all multiple object declarations into
```

```
   -- an equivalent sequence of corresponding single object declarations.
   -- See LRM 3.2(10).
   --
   -- Appropriate Declaration Kinds:
   --   A_Package_Declaration,
   --   A_Generic_Package_Declaration,
   --   A_Package_Instantiation,


   function Private_Part_Declarative_Items
      (Package_Specification : in Declaration)
     return Declarative_Item_List;

   -- Returns a list of all basic declarations, representation specifications,
   -- and use clauses in the private part of a package in the order of
   -- appearance.
   --
   -- Results of this operation may vary across ASIS implementations.
   -- Some implementations normalize all multiple object declarations into
   -- an equivalent sequence of corresponding single object declarations.
   -- See LRM 3.2(10).
   --
   -- Appropriate Declaration Kinds:
   --   A_Package_Declaration,
   --   A_Generic_Package_Declaration,
   --   A_Package_Instantiation,


   function Package_Body_Block (Package_Body : in Declaration)
     return Statement;

   -- Returns the block statement for the body including the declarative
   -- part, sequence of statements, and exception handlers, if any.
   --
   -- A Nil_Element is returned if the body is a body stub, or
   -- if the body of a package instantiation does not exist.
   --
   -- Appropriate Declaration Kinds:
   --   A_Package_Body_Declaration,
   --   A_Package_Instantiation,
   --   A_Package_Body_Stub,


   ------------------------------------------------------------------
   -- RENAMING DECLARATIONS
```

```
-- LRM Section 8.5


function Is_Renaming_Declaration (A_Declaration : in Declaration)
  return Boolean;

-- Returns True for declarations that are one of the forms of renaming
-- declarations.
--
-- Raises Inappropriate_Program_Element
-- if Kind(A_Declaration) = Not_A_Declaration


function Renamed_Entity (Renaming_Declaration : in Declaration)
  return Name_Expression;

-- Returns the Name_expression that follows the reserved word renames
-- in the Renaming_Declaration.
--
-- Raises Inappropriate_Program_Element
-- if not Is_Renaming_Declaration(Renaming_Declaration)


function Renamed_Base_Entity (Renaming_Declaration : in Declaration)
  return Name_Expression;

-- Unwinds recursive renamings to return the Name_expression that is
-- not itself a renamed entity.
--
-- Raises Inappropriate_Program_Element
-- if not Is_Renaming_Declaration(Renaming_Declaration)



------------------------------------------------------------
-- TASK DECLARATIONS
-- LRM Chapter 9


function Task_Declaration_Declarative_Items
   (Task_Declaration : in Declaration)
  return Declarative_Item_List;

-- Returns a list of entry declarations, and representation clauses
-- in the task declaration in order of appearance.
```

```
--
-- Further Analysis:
--   Use ASIS_Declarations operations for subprogram declarations to
--   decompose task entries.
--   Use ASIS_Representation_Clauses operations to analyze representation
--   clauses that apply to the entries.
--
-- Appropriate Declaration Kinds:
--   A_Task_Declaration
--   A_Task_Type_Declaration
--
-- Task type declarations can also be analyzed by getting the type_definition
-- part and then calling ASIS_Type_Definitions.Task_Type_Declarative_Items.


function Task_Body_Block (Task_Body : in Declaration) return Statement;

-- Returns the block statement for the body including the declarative part,
-- any elaboration statements, and exception handler if any.
--
-- A Nil_Element is returned if the body is a body stub.
--
-- Appropriate Declaration Kinds:
--   A_Task_Body_Declaration,
--   A_Task_Body_Stub,


-------------------------------------------------------------------
-- ENTRY DECLARATIONS
-- LRM Section 9.5


subtype Entry_Declaration_List is ASIS_Ada_Program.Element_List;

function Entry_Declarations (Task_Declaration : in Declaration)
  return Entry_Declaration_List;

-- Returns a list of entry declarations associated with a task declaration.
--
-- The operations available for decomposing subprogram specifications
-- will also work for entry declarations.
--
-- Appropriate Declaration Kinds:
--   A_Task_Declaration,
--   A_Task_Type_Declaration,
```

```
subtype Discrete_Range is ASIS_Ada_Program.Element;
  -- Use ASIS_Type_Definitions operations to analyze discrete ranges.

function Family_Index (Entry_Family : in Declaration)
  return Discrete_Range;

-- Returns the index discrete range for the entry family.
--
-- If the entry is not a family, a nil element is returned.
--
-- Appropriate Declaration Kinds:
--   An_Entry_Declaration


-------------------------------------------------------------
-- SUBUNITS AND BODY STUBS - LRM 10.2

-- Use ASIS_COMPILATION_UNTIS operations to examine subunits as
-- compilation units.


function Is_Subunit (A_Declaration : in Declaration) return Boolean;

-- Returns True if the declaration is the declaration part of a subunit.
--
-- Raises Inappropriate_Program_Element
-- if Kind(A_Declaration) = Not_A_Declaration


function Subunit (Body_Stub        : in Declaration)
  return Declaration;

function Subunit (Body_Stub        : in Declaration;
                  Program_Library  : in Library)
  return Declaration;

-- Returns the Subunit Declaration corresponding to the body stub.
--
-- A Nil_Element is returned if the subunit does not exist.
--
-- Appropriate Declaration Kinds:
--   Any declaration for which Is_Body_Stub(Body_Stub) = True.
```

```
function Is_Body_Stub (A_Declaration : in Declaration) return Boolean;

-- Returns true if the Declaration is:
--   A_Package_Body_Stub,
--   A_Function_Body_Stub,
--   A_Procedure_Body_Stub,
--   A_Task_Body_Stub,
--
-- Raises Inappropriate_Program_Element
-- if Kind(A_Declaration) = Not_A_Declaration


function Body_Stub (Subunit : in Declaration) return Declaration;

-- Returns the body stub declaration in the subunit's parent unit.
--
-- A Nil_Element is returned if not the body stub does not exist.
--
-- Appropriate Declaration Kinds:
--   Any declaration for which Is_Subunit(Subunit) = True.


------------------------------------------------------------------
-- EXCEPTION DECLARATIONS
-- LRM Chapter 11

-- Use Identifiers to get a list of identifier definitions introduced
-- by an exception declaration.


------------------------------------------------------------------
-- GENERIC PACKAGE and SUBPROGRAM SPECIFICATIONS
-- LRM Chapter 12


function Is_Generic (Package_Or_Subprogram_Declaration : in Declaration)
  return Boolean;

-- Determines if the declaration is a generic specification or body.
--
-- Returns True if the declaration returned by a call to
-- Specification(Package_Or_Subprogram_Decl) is:
--   A_Generic_Package_Declaration,
--   A_Generic_Procedure_Declaration,
```

```
--   A_Generic_Function_Declaration,
--
-- Appropriate Declaration Kinds:
--   Any declaration for which Is_Subprogram returns True
--   Any declaration for which Is_Package returns True


subtype Generic_Formal_Parameter is Declaration;

subtype Generic_Formal_Parameter_List is ASIS_Ada_Program.Element_List;


function Generic_Formal_Parameters (Generic_Declaration : in Declaration)
  return Generic_Formal_Parameter_List;

-- Returns a list of generic formal parameter declarations in order of
-- appearance.
--
-- Results of this operation may vary across ASIS implementations.
-- Some implementations normalize all multiple object declarations into
-- an equivalent sequence of corresponding single object declarations.
-- See LRM 3.2(10).
--
-- Further Analysis:
--   Object parameters can be decomposed with formal parameter
--     and object declaration operations.
--   The mode of a generic formal object can be determined using the
--     Parameter_Kind operation.
--   Array and access type declarations can be decomposed with the
--     operations corresponding to their types.
--   Subprogram parameters can be queried by the following operations and
--     can be further decomposed with subprogram declaration operations.
--
-- Appropriate Declaration Kinds:
--   A_Generic_Package_Declaration,
--   A_Generic_Procedure_Declaration,
--   A_Generic_Function_Declaration,


function Is_Generic_Formal (Program_Element : in Element) return Boolean;

-- Returns true if a subprogram, type or object declaration is a generic
-- formal parameter.
--
-- Raises Inappropriate_Program_Element if the element is a Nil_Element.
```

```
type Generic_Formal_Subprogram_Default_Kinds is

   (A_Box,
    A_Name,
    None,
    Not_A_Generic_Formal_Subprogram_Default);

function Generic_Formal_Subprogram_Default_Kind
   (A_Generic_Formal_Subprogram : in Generic_Formal_Parameter)
  return Generic_Formal_Subprogram_Default_Kinds;

-- Returns the kind of the given generic formal subprogram default.
-- Returns Not_A_Generic_Formal_Subprogram_Default for any inappropriate
-- element such as a Statement, Expression, or Nil_Element.


function Generic_Formal_Subprogram_Default
   (A_Generic_Formal_Subprogram : in Generic_Formal_Parameter)
  return Name_Expression;

-- Returns the name appearing after the reserved word is in the given
-- generic formal subprogram declaration.
--
-- Raises Inappropriate_Program_Element if
-- Generic_Formal_Subprogram_Default_Kind(A_Generic_Formal_Subprogram)
--  /= A_Name


-------------------------------------------------------------
-- GENERIC INSTANTIATION
-- LRM Section 12.3

-- Instantiations can be analyzed in terms of the generic actual parameters
-- supplied with the instantiation.  An instance is a copy of the generic
-- unit, and while there is no explicit specification in the program text,
-- an implicit specification and body, if there is one, with the generic
-- actual parameters is implied.
--
-- To analyze the template specification or body of a generic instantiation:
--   Use Generic_Unit_Name to obtain the name of the generic unit.
--   Then use ASIS_Names_And_Expressions.Named_Declaration to get to the
--     generic declaration.
--   Then use Unit_Body to get to the body of the generic declaration.
```

```
function Is_Generic_Instantiation
   (Package_Or_Subprogram_Declaration : in Declaration)
  return Boolean;

-- Returns True if the declaration is:
--   A_Package_Instantiation,
--   A_Procedure_Instantiation,
--   A_Function_Instantiation,
--
-- Raises Inappropriate_Program_Element
-- if Kind(Package_Or_Subprogram_Declaration) = Not_A_Declaration


function Generic_Unit_Name (Generic_Instantiation : in Declaration)
  return Name_Expression;

-- Returns the name expression 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 the instance.
--
-- Appropriate Declaration Kinds:
--   A_Package_Instantiation,
--   A_Procedure_Instantiation,
--   A_Function_Instantiation,



subtype Generic_Association_List is ASIS_Ada_Program.Element_List;

function Generic_Associations
   (Generic_Instantiation : in Declaration;
    Include_Defaults      : in Boolean := False)
  return Generic_Association_List;

-- Returns an ordered list of generic associations from the generic
-- actual part of an instantiation.
--
-- If Include_Defaults is True:
--   The (unspecified) default associations will also be included and,
--   the list will be ordered to match the declaration order of the
--   corresponding generic formal parameters.
--
-- If Include_Defaults is False:
```

```
   --    Only explicit actual parameters will be included in the order they
   --    appeared in the program text.
   --
   -- An ASIS implementation may choose to always "Include Defaults" its
   -- internal representation.  If that is the case, the operation will
   -- always behave as if Include_Defaults is True.
   --
   -- Appropriate Declaration Kinds:
   --    A_Package_Instantiation,
   --    A_Procedure_Instantiation,
   --    A_Function_Instantiation,
   --
   -- To map generic actual parameters to corresponding generic formal
   -- parameters:
   --
   --    Create an actual parameter list by applying the function
   --    ASIS_Statements.Actual_Parameter to each generic association
   --    returned by Generic_Actual_Parameters with Include_Defaults = True.
   --
   --    Create a formal parameter list by applying the function
   --    ASIS_Declarations.Identifiers to each formal parameter
   --    declaration returned by Generic_Formal_Parameters.
   --
   --    The Expression elements in the actual parameters list will have a
   --    one to one correspondence with the Identifier_Definition elements
   --    in the formal parameter list.


   function Generic_Associations_Include_Defaults return Boolean;

   -- Returns True if this implementation will always "Include Defaults".
   -- If that is the case, the Generic_Associations operation will always
   -- behave as if its parameter Include_Defaults is True.


end ASIS_Declarations;
```

```
-------------------------------------------------------------------------
-- ASIS PRELIMINARY REVIEW VERSION 0.4
-- Sept 27, 1991
-------------------------------------------------------------------------


with ASIS_Ada_Program;
package ASIS_Type_Definitions is

  -- LRM Chapter 3

  subtype Element is ASIS_Ada_Program.Element;
    -- Use ASIS_Elements operations.

  subtype Type_Definition is ASIS_Ada_Program.Type_Definition;
    -- Use ASIS_Type_Definitions operations.

  subtype Declaration is ASIS_Ada_Program.Declaration;
    -- Use ASIS_Declarations operations.

  subtype Expression is ASIS_Ada_Program.Expression;
    -- Use ASIS_Names_And_Expression operations.

  subtype Name_Expression is ASIS_Ada_Program.Name_Expression;
    -- Use ASIS_Names_And_Expression operations.

  subtype Statement is ASIS_Ada_Program.Statement;
    -- Use ASIS_Statements operations.

  subtype Identifier_Reference is ASIS_Ada_Program.Identifier_Reference;
    -- Use ASIS_Names_And_Expression operations.

  subtype Subtype_Indication is Type_Definition;
    -- Use operations in this package and in ASIS_Names_And_Expressions.

  ---------------------------------------------------------------
  -- TYPE DEFINITION KINDS


  type Type_Definition_Kinds is

     (A_Subtype_Definition,    -- Though not an Ada term, this kind is
                               -- distinct from a subtype indication which
      -- is a component of several other
      -- Type_Definition_Kinds
```

```
    An_Enumeration_Type_Definition,
    An_Integer_Type_Definition,
    A_Float_Type_Definition,
    A_Fixed_Type_Definition,
    An_Array_Type_Definition,
    A_Record_Type_Definition,
    An_Access_Type_Definition,
    A_Derived_Type_Definition,
    A_Task_Type_Definition,
    A_Private_Type_Definition,
    A_Limited_Private_Type_Definition,

    A_Generic_Discrete_Subtype_Definition,
    A_Generic_Integer_Subtype_Definition,
    A_Generic_Float_Subtype_Definition,
    A_Generic_Fixed_Subtype_Definition,
    A_Generic_Array_Type_Definition,
    A_Generic_Access_Type_Definition,
    A_Generic_Private_Type_Definition,
    A_Generic_Limited_Private_Type_Definition,

    Not_A_Type_Definition);


function Kind (A_Type_Definition : in Type_Definition)
  return Type_Definition_Kinds;

-- Returns the kind of the Type_Definition.
--
-- Returns Not_A_Type_Definition for any inappropriate element such as
-- a Statement, Declaration, or Nil_Element.


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

function Type_Definition_Declaration (A_Type_Definition : in Type_Definition)
  return Declaration;

-- Returns the type declaration or the object declaration associated
-- with the type definition.
-- A Nil_Element is returned when the type declaration is anonymous.
-- EG. the type_definition is An_Array_Type_Definition from an object
-- declaration.
--
```

```
-- Raises Inappropriate_Program_Element
-- if Kind(A_Type_Definition) = Not_A_Type_Definition


------------------------------------------------------------
-- TYPES AND SUBTYPES - LRM 3.3


function Base_Type (A_Type_Definition : in Type_Definition)
  return Type_Definition;

-- Returns the base type of the specified type definition as per LRM 3.3.
-- All subtypes are constraints applied to some base type.  This function
-- returns that base type.
--
-- This is an identity function if the type definition of a base type
-- is given.
--
-- A Nil_Element is returned when the type definition is from an
-- anonymous type or from a predefined type.

-- Raises Inappropriate_Program_Element
-- if Kind(A_Type_Definition) = Not_A_Type_Definition


function Last_Constraint (A_Type_Definition : in Type_Definition)
  return Type_Definition;

-- This function recursively unwinds subtyping to arrive at a type
-- definition which is either the base_type or a subtype that imposes
-- an explicit constraint.
--
-- This is an identity function if the type definition is not
-- A_Subtype_Definition.
--
-- Raises Inappropriate_Program_Element
-- if Kind(A_Type_Definition) = Not_A_Type_Definition


function Last_Subtype (A_Type_Definition : in Type_Definition)
  return Type_Definition;

-- This function unwinds subtyping one step at a time to return a type
-- definition which is either the base_type or a subtype that imposes
-- constraints (the case where the constraint imposes no restriction is
```

```
-- also included).  LRM 3.3(4).
--
-- This is an identity function if the type definition is not
-- A_Subtype_Definition.
--
-- Raises Inappropriate_Program_Element
-- if Kind(A_Type_Definition) = Not_A_Type_Definition


function Subtype_Definition_Subtype_Indication
   (Subtype_Definition : in Type_Definition)
  return Subtype_Indication;

-- Returns the subtype indication associated with the subtype definition.
--
-- Raises Inappropriate_Program_Element
-- if Kind(Subtype_Definition) /= A_Subtype_Definition.


------------------------------------------------------------
-- DERIVED TYPES - LRM 3.4

function Parent_Subtype (Derived_Type : in Type_Definition)
  return Subtype_Indication;

-- Returns the subtype indication associated with the derived type.
-- The subtype indication after the reserved word new defines the
-- parent subtype.
--
-- Raises Inappropriate_Program_Element
-- if Kind(Derived_Type) /= A_Derived_Type_Definition.


function Parent_Type (Derived_Type : in Type_Definition)
  return Type_Definition;

-- Returns the parent type of the specified derived type definition as
-- per LRM 3.4.
-- The parent type is the base type of the parent subtype.
--
-- Appropriate Type_Definition Kinds:
--   A_Derived_Type_Definition
```

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

function Ground_Type (A_Type_Definition : in Type_Definition)
  return Type_Definition;

-- This function recursively unwinds all type derivations and subtyping
-- to arrive at a type definition which is neither a derived type or a
-- subtype.
--
-- Raises Inappropriate_Program_Element
-- if Kind(A_Type_Definition) = Not_A_Type_Definition



------------------------------------------------------------
--
-- Examples of base type, parent type, and ground type.
--
-- type Number is new Integer range 0 .. 100;
--   A derived type definition defines a new base type whose
--   characteristics are derived from those of a parent type.
--
--   The base type of Number is Number.
--   The parent type of Number is Integer.
--   The ground type of Number is Integer.
--
--
-- subtype Day_Number is Number range 1 .. 31;
--
--   A subtype indication defines a subtype of the base type of the
--   type mark.
--
--   The base type of Day_Number is Number.
--   The parent type of Day_Number is Integer.
--   The ground type of Day_Number is Integer.
--
--
-- type Week_Day is new Day_Number range 1 .. 7;
--
--   The subtype indication after the reserved word new defines the
--   parent subtype. The parent type is the base type of the parent subtype.
--
--   The base type of Week_Day is Number.
--   The parent type of Week_Day is Number.
--   The ground type of Week_Day is Integer.
```

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


function Type_Structure (A_Type_Definition : in Type_Definition)
  return Type_Definition;

-- Returns the type structure from which the specified type definition has
-- been derived.  This function will unwind recursive derivations until the
-- type definition derives a new representation or is no longer derived.
-- This function is different from GROUND_TYPE only for enumeration types or
-- record types that have derivations with rep specs.
--
-- Raises Inappropriate_Program_Element
-- if Kind(A_Type_Definition) = Not_A_Type_Definition


function Is_Predefined (A_Type_Definition : in Type_Definition)
  return Boolean;

-- Returns true if the base type of the type definition is one of:
-- Boolean, Character, String, Integer, Natural, Positive, Float, Duration,
-- or any other type declared in the predefined package Standard.
--
-- TRUE  if given type definition of:
--   subtype Weight is Natural range 0..50;
-- FALSE if given type definition of:
--   type Weight is new Natural is range 1..10;
--
-- Raises Inappropriate_Program_Element
-- if Kind(A_Type_Definition) = Not_A_Type_Definition


-------------------------------------------------------------
-- CONSTRAINTS


subtype Constraint is ASIS_Ada_Program.Element;

type Constraint_Kinds is

   (A_Simple_Range,
    A_Range_Attribute,
    A_Floating_Point_Constraint,
    A_Fixed_Point_Constraint,
```

```
    An_Index_Constraint,
    A_Discriminant_Constraint,
    Not_A_Constraint);

function Constraint_Kind (A_Constraint : in Constraint)
  return Constraint_Kinds;

-- Returns the kind of the Constraint.
--
-- Returns Not_A_Constraint for any inappropriate element such as
-- a Statement, Declaration, or Nil_Element.


subtype Discriminant_Association_List is ASIS_Ada_Program.Element_List;

function Discriminant_Associations
   (Discriminant_Constraint : in Constraint;
    Include_Defaults        : in Boolean := False)
  return Discriminant_Association_List;

-- Returns the list of discriminant associations of a discriminant
-- constraint.
--
-- If Include_Defaults is True:
--   The (unspecified) default associations will also be included and,
--   The list will be ordered to match the declaration order of the
--   identifiers in the corresponding discriminant specifications.
--
-- If Include_Defaults is False:
--   Only explicit discriminant associations will be included in the
--   order they appeared in the program text.
--
-- An ASIS implementation may choose to always "Include Defaults" its
-- internal representation.  If that is the case, the operation will
-- always behave as if Include_Defaults is True.
--
-- Raises Inappropriate_Program_Element
-- if Constraint_Kind(Discriminant_Constraint)
--   /= A_Discriminant_Constraint


function Discriminant_Associations_Include_Defaults return Boolean;

-- Returns True if this implementation will always "Include Defaults".
-- If that is the case, the Discriminant_Associations operation will
```

```
-- always behave as if its parameter Include_Defaults is True.


subtype Discriminant_Association is ASIS_Ada_Program.Element;

subtype Identifier_Reference_List is ASIS_Ada_Program.Element_List;

function Discriminant_Simple_Names
   (A_Discriminant_Association : in Discriminant_Association)
  return Identifier_Reference_List;

-- Returns the list of simple names in the discriminant association.
-- Will be a nil list if positional notation is used.
--
-- Raises Inappropriate_Program_Element
-- if ASIS_Elements.Major_Element_Kind(A_Discriminant_Association) /=
-- A_Discriminant_Association


function Discriminant_Expression
   (A_Discriminant_Association : in Discriminant_Association)
  return Expression;

-- Returns the expression for the discriminant association.
--
-- Raises Inappropriate_Program_Element
-- if ASIS_Elements.Major_Element_Kind(A_Discriminant_Association) /=
-- A_Discriminant_Association



-------------------------------------------------------------------
-- DISCRETE RANGES


subtype Discrete_Range is ASIS_Ada_Program.Element;  -- LRM 3.6

type Discrete_Range_Kinds is

   (A_Simple_Range,
    A_Range_Attribute,
    A_Discrete_Subtype_Indication,
    Not_A_Discrete_Range);

function Discrete_Range_Kind (A_Discrete_Range : in Discrete_Range)
```

```
   return Discrete_Range_Kinds;

-- Returns the kind of the Discrete_Range.
--
-- Returns Not_A_Discrete_Range for any inappropriate element such as
-- a Statement, Declaration, or Nil_Element.


-------------------------------------------------------------------
-- RANGE CONSTRAINT


subtype Range_Constraint is ASIS_Ada_Program.Element;

function Lower_Bound
   (Simple_Range_Constraint : in Range_Constraint)
  return Expression;

-- Returns the simple expression for the lower bound of a simple range.
--
-- Appropriate elements of the subtypes constraint or discrete range:
-- Appropriate Constraint Kinds:
--   A_Simple_Range
-- Appropriate Discrete_Range Kinds:
--   A_Simple_Range


function Upper_Bound
   (Simple_Range_Constraint : in Range_Constraint)
  return Expression;

-- Returns the simple expression for the upper bound of a simple range.
--
-- Appropriate elements of the subtypes constraint or discrete range:
-- Appropriate Constraint Kinds:
--   A_Simple_Range
-- Appropriate Discrete_Range Kinds:
--   A_Simple_Range


function Range_Attribute
   (Range_Attribute_Constraint : in Range_Constraint)
  return Expression;

-- Returns the range attribute expression supplying the range.
```

```
-- Accepts elements of the subtypes constraint or discrete range.
--
-- Further Analysis:
--   Range attributes are expressions and can be analyzed by using
--   the attribute operations in ASIS_Names_And_Expressions.
--
-- Appropriate Constraint Kinds:
--   A_Range_Attribute
-- Appropriate Discrete_Range Kinds:
--   A_Range_Attribute


-------------------------------------------------------------
-- CHOICES


subtype Choice is ASIS_Ada_Program.Element;

type Choice_Kinds is

   (A_Simple_Expression,
    A_Discrete_Range,
    An_Others_Choice,
    A_Simple_Name,
    An_Exception_Name,
    Not_A_Choice);

function Choice_Kind (A_Choice : in Choice) return Choice_Kinds;

-- Returns the kind of the choice.
-- Returns Not_A_Choice for any inappropriate element such as
-- a Statement, Declaration, or Nil_Element.


function Choice_Simple_Expression (A_Choice : in Choice) return Expression;

-- Returns the expression associated with the choice.
--
-- Appropriate Choice Kinds:
--   A_Simple_Expression


function Choice_Discrete_Range (A_Choice : in Choice) return Discrete_Range;

-- Returns the discrete range associated with the choice.
```

```
--
-- Appropriate Choice Kinds:
--   A_Discrete_Range


function Choice_Name (A_Choice : in Choice) return Name_Expression;

-- Returns the simple name or exception name associated with the choice.
--
-- Appropriate Choice Kinds:
--   A_Simple_Name
--   An_Exception_Name


------------------------------------------------------------------
-- SUBTYPE_INDICATIONS AND TYPE_MARKS - LRM 3.3.2


function Type_Mark (A_Subtype_Indication : in Subtype_Indication)
  return Name_Expression;

-- Returns the type mark of a subtype indication or a discrete subtype
-- indication.
--
-- Appropriate Type_Definition Kinds:
--   A_Subtype_Indication
--
-- Appropriate Discrete_Range Kinds:
--   A_Discrete_Subtype_Indication
--
-- Use Index_Subtype_Definitions query to get the type mark(s) from
-- unconstrained array Index_Subtype_Definition(s).


function Subtype_Constraint (A_Subtype_Indication : in Subtype_Indication)
  return Constraint;

-- Returns the constraint applied to the subtype indication.
-- A Nil_Element is returned if no constraint is present.
--
-- Appropriate Type_Definition Kinds:
--   A_Subtype_Indication
--
-- Appropriate Discrete_Range Kinds:
--   A_Discrete_Subtype_Indication
```

```
------------------------------------------------------------
-- ENUMERATION TYPES - LRM 3.5.1

subtype Declaration_List is ASIS_Ada_Program.Element_List;

function Enumeration_Literal_Declarations
   (Enumeration_Type : in Type_Definition)
  return Declaration_List;

-- Returns a list of the literals declared in an enumeration type
-- declaration.  Each of these elements is a Declaration.
--
-- Appropriate Type_Definition Kinds:
--   An_Enumeration_Type_Definition


subtype Identifier_Definition_List is ASIS_Ada_Program.Element_List;

function Enumeration_Literal_Identifiers
   (Enumeration_Type : in Type_Definition)
  return Identifier_Definition_List;

-- Returns a list of the literals declared in an enumeration type
-- declaration.  Each of these elements is an Identifier_Definition.
--
-- Appropriate Type_Definition Kinds:
--   An_Enumeration_Type_Definition


------------------------------------------------------------
-- INTEGER TYPES - LRM 3.5.4


function Integer_Constraint (Integer_Type : in Type_Definition)
  return Range_Constraint;

-- Returns the range constraint on the integer type definition.
--
-- A Nil_Element may be returned by some ASIS implementations if the
-- integer type is a predefined type.
--
-- Appropriate Type_Definition Kinds:
--   An_Integer_Type_Definition
```

```
------------------------------------------------------------
-- REAL TYPES - LRM 3.5.6


function Real_Type_Constraint (Real_Type : in Type_Definition)
  return Constraint;

-- Returns the floating point constraint from a A_Float_Type_Definition
-- or the fixed point constraint from A_Fixed_Type_Definition.
--
-- A Nil_Element may be returned by some ASIS implementations if the
-- real type is a predefined type.
--
-- Appropriate Type_Definition Kinds:
--   A_Float_Type_Definition
--   A_Fixed_Type_Definition

-- Primary

function Floating_Accuracy_Definition
   (Floating_Point_Constraint : in Constraint)
  return Expression;

-- Returns the static simple expression following the reserved word digits
-- in the floating accuracy definition of a floating point constraint.
--
-- Appropriate Constraint Kinds:
--   A_Floating_Point_Constraint


function Floating_Point_Range_Constraint
   (Floating_Point_Constraint : in Constraint)
  return Range_Constraint;

-- Returns the optional range constraint of a floating point constraint.
--
-- A Nil_Element is returned if there is no range constraint.
--
-- Appropriate Constraint Kinds:
--   A_Floating_Point_Constraint


function Fixed_Accuracy_Definition
```

```
   (Fixed_Point_Constraint : in Constraint)
  return Expression;

-- Returns the static simple expression following the reserved word delta
-- in the fixed accuracy definition of a fixed point constraint.
--
-- Appropriate Constraint Kinds:
--   A_Fixed_Point_Constraint


function Fixed_Point_Range_Constraint
   (Fixed_Point_Constraint : in Constraint)
  return Range_Constraint;

-- Returns the optional range constraint of the fixed point type definition.
--
-- A Nil_Element is returned if there is no range constraint.
--
-- Appropriate Constraint Kinds:
--   A_Fixed_Point_Constraint


------------------------------------------------------------
-- ARRAY TYPES - LRM 3.6


function Is_Constrained_Array (Array_Type : in Type_Definition)
  return Boolean;

-- Returns True if the array type definition denotes a constrained array.
--
-- Appropriate Type_Definition Kinds:
--   An_Array_Type_Definition
--   A_Generic_Array_Type_Definition



function Index_Constraint (Constrained_Array_Type : in Type_Definition)
  return Constraint;

-- Returns the index constraint for a constrained array type declaration.
--
-- A constrained array definition has one index constraint which defines
-- one or more discrete ranges.
--
```

```
-- Use the Discrete_Ranges operation to decompose the index constraint.
--
-- Raises Inappropriate_Program_Element
-- if not Is_Constrained_Array(Constrained_Array_Type) = True


subtype Discrete_Range_List is ASIS_Ada_Program.Element_List;

function Discrete_Ranges (Index_Constraint : in Constraint)
  return Discrete_Range_List;

-- Returns the list of Discrete_Range components of an Index_Constraint
--
-- Raises Inappropriate_Program_Element
-- if Constraint_Kind(Index_Constraint) /= An_Index_Constraint


subtype Name_Expression_List is ASIS_Ada_Program.Element_List;

function Index_Subtype_Definitions
   (Unconstrained_Array_Type : in Type_Definition)
  return Name_Expression_List;

-- Returns a list of the Index_Subtype_Definition (Type_Mark range <>)
-- for an unconstrained array type definition.
--
-- The list contains elements of the subtype Name_Expression
-- representing the Type_Mark(s).
--
-- Raises Inappropriate_Program_Element
-- if Is_Constrained_Array(Unconstrained_Array_Type) = True


function Component_Subtype_Indication (Array_Type : in Type_Definition)
  return Subtype_Indication;

-- Returns the component subtype indication of the array type.
--
-- A Nil_Element may be returned by some ASIS implementations if the
-- array type is a predefined type.
--
-- Appropriate Type_Definition Kinds:
--   An_Array_Type_Definition
--   A_Generic_Array_Type_Definition
```

```
-------------------------------------------------------------
-- RECORD TYPES - LRM 3.7


subtype Record_Component is ASIS_Ada_Program.Element;

subtype Variant is ASIS_Ada_Program.Element;


function Is_Discriminated (Record_Type : in Type_Definition) return Boolean;

-- Returns True if the record type definition has discriminants.
--
-- Appropriate Type_Definition Kinds:
--   A_Record_Type_Definition


subtype Discriminant_Specification_List is ASIS_Ada_Program.Element_List;

function Discriminants (Record_Type : in Type_Definition)
  return Discriminant_Specification_List;

-- Returns a list of discriminant specification declarations
-- in their order of appearance.
--
-- A Nil list is returned if the record type definition has no
-- discriminants.
--
-- Further Analysis:
--   Use ASIS_DECLARATIONS operations to decompose discriminant
--   specifications.
--
-- Appropriate Type_Definition Kinds:
--   A_Record_Type_Definition


subtype Record_Component_List is ASIS_Ada_Program.Element_List;

function Record_Components (Record_Type : in Type_Definition)
  return Record_Component_List;

-- Returns a list of the components of the record definition.
--
-- Appropriate Type_Definition Kinds:
```

```
--   A_Record_Type_Definition


type Component_Kinds is

   (A_Null_Component,
    A_Component_Declaration,
    A_Variant_Part,
    Not_A_Component);

function Component_Kind (Component : in Record_Component)
  return Component_Kinds;

-- Returns the kind of the component.
--
-- Returns Not_A_Component for any inappropriate element such as
-- a Statement, Expression, or Nil_Element.

-- Component declarations can be analyzed using the ASIS_DECLARATIONS
-- operations.


function Corresponding_Discriminant_Simple_Name
   (Variant_Part : in Record_Component)
  return Identifier_Reference;

-- Returns the discriminant simple name in the variant part.
--
-- Raises Inappropriate_Program_Element
-- if Component_Kind(Variant_Part) /= A_Variant_Part.


subtype Variant_List is ASIS_Ada_Program.Element_List;

function Variants (Variant_Part : in Record_Component) return Variant_List;

-- Returns a list of variants that make up the record component.
--
-- Raises Inappropriate_Program_Element
-- if Component_Kind(Variant_Part) /= A_Variant_Part.


subtype Choice_List is ASIS_Ada_Program.Element_List;

function Variant_Choices (A_Variant : in Variant) return Choice_List;
```

```
-- Returns a list of the 'WHEN <choice> | <choice>' choices.
-- Use the CHOICES operations to extract further information.
--
-- Raises Inappropriate_Program_Element
-- if ASIS_Elements.Major_Element_Kind(A_Variant) /= A_Variant


subtype Variant_Component_List is ASIS_Ada_Program.Element_List;

function Variant_Components (A_Variant: in Variant)
  return Variant_Component_List;

-- Returns a list of the record components of the variant.
--
-- Further Analysis:
-- Use the Component_Kind operation to analyze the variant components.
--
-- Raises Inappropriate_Program_Element
-- if ASIS_Elements.Major_Element_Kind(A_Variant) /= A_Variant


-------------------------------------------------------------
-- ACCESS TYPES - LRM 3.8

function Access_To (Access_Type : in Type_Definition)
  return Subtype_Indication;

-- Returns the subtype indication following the reserved word access
-- in the access type definition.
--
-- Appropriate Type_Definition Kinds:
--   An_Access_Type_Definition.
--   A_Generic_Access_Type_Definition.


-------------------------------------------------------------
-- TASK SPECIFICATIONS
-- LRM Chapter 9

subtype Declarative_Item_List is ASIS_Ada_Program.Element_List;

function Task_Type_Declarative_Items (Task_Type : in Type_Definition)
  return Declarative_Item_List;
```

```
   -- Returns a list of entry declarations and representation clauses
   -- in the specification of a task type in order of appearance.
   --
   -- Further Analysis:
   --   Use ASIS_Declarations operations for subprogram declarations to
   --   decompose task entries.
   --   Use ASIS_Representation_Clauses operations to analyze representation
   --   clauses that apply to the entries.
   --
   -- Appropriate Type_Definition Kinds:
   --   A_Task_Type_Definition



end ASIS_Type_Definitions;
```

```
-------------------------------------------------------------------------
-- ASIS PRELIMINARY REVIEW VERSION 0.4
-- Sept 27, 1991
-------------------------------------------------------------------------


with ASIS_Ada_Program;
package ASIS_Names_And_Expressions is

   -- LRM Chapter 4

   subtype Element is ASIS_Ada_Program.Element;
     -- Use ASIS_Elements operations.

   subtype Expression is ASIS_Ada_Program.Element;
     -- Use ASIS_Names_And_Expression operations.

   subtype Name_Expression is ASIS_Ada_Program.Element;
     -- Use ASIS_Names_And_Expression operations.

   subtype Declaration is ASIS_Ada_Program.Element;
     -- Use ASIS_Declarations operations.
```

```
subtype Statement is ASIS_Ada_Program.Element;
  -- Use ASIS_Statements operations.

subtype Type_Definition is ASIS_Ada_Program.Element;
  -- Use ASIS_Type_Definitions operations.


-----------------------------------------------------------------
-- NAME AND EXPRESSION KINDS


type Name_And_Expression_Kinds is

   (A_Simple_Name,                          -- LRM 4.1
    An_Operator_Symbol,                     -- LRM 4.1
    A_Character_Literal,                    -- LRM 4.2
    An_Enumeration_Literal,                 -- LRM 4.2

    An_Indexed_Component,                   -- LRM 4.1.1
    A_Slice,                                -- LRM 4.1.2
    A_Selected_Component,                   -- LRM 4.1.3
    An_Attribute,                           -- LRM 4.1.4
    A_Type_Conversion,                      -- LRM 4.6
    A_Qualified_Expression,                 -- LRM 4.7
    A_Function_Call,                        -- LRM 6.4

    A_Null_Literal,                         -- LRM 4.2
    A_String_Literal,                       -- LRM 4.2
    An_Integer_Literal,                     -- LRM 4.2
    A_Real_Literal,                         -- LRM 4.2
    An_Aggregate,                           -- LRM 4.3
    A_Parenthesized_Expression,             -- LRM 4.4
    A_Special_Operation,                    -- LRM 4.4
    An_Allocation_From_Subtype,             -- LRM 4.8
    An_Allocation_From_Qualified_Expression, -- LRM 4.8

    Not_An_Expression);


function Kind (An_Expression : in Expression)
  return Name_And_Expression_Kinds;

-- Returns the kind of the Expression, Name_Expression,
-- Identifier_Definition, or Identifier_Reference.
```

```
--
-- Returns Not_An_Expression for any inappropriate element such as
-- a Statement, Declaration, or Nil_Element.


-------------------------------------------------------------------
-- THE TYPE OF AN EXPRESSION


function Expression_Type (An_Expression : in Expression)
  return Type_Definition;

-- Returns the type definition for the expression.
-- This operation does not "unwind" subtypes or derived types to get to
-- the base type or parent type definitions.  That can be accomplished
-- using ASIS_Type_Definitions operations Parent_Type and Base_Type.
--
-- Raises Inappropriate_Program_Element
-- if Kind(An_Expression) = Not_An_Expression


function Is_Universal (An_Expression : in Expression) return Boolean;

-- Returns True if the expression is of the type Universal_Integer or
-- the type Universal_Real.
--
-- Raises Inappropriate_Program_Element
-- if Kind(An_Expression) = Not_An_Expression


-------------------------------------------------------------------
-- IDENTIFIERS - LRM 2.3


subtype Identifier_Definition is ASIS_Ada_Program.Element;
subtype Identifier_Reference is ASIS_Ada_Program.Element;


type Id_Kinds is

   (An_Identifier_Definition,
    An_Identifier_Reference,
    Not_An_Identifier);

function Id_Kind (An_Identifier : in Element) return Id_Kinds;
```

```
-- Returns the kind of the Identifier_Definition or Identifier_Reference.
--
-- Returns Not_An_Identifier for any inappropriate element such as
-- a Statement, Declaration, or Nil_Element.


function String_Name (An_Identifier : in Element) return String;

-- Returns the name of the identifier.
--
-- Appropriate Id_Kinds:
--   An_Identifier_Definition,
--   An_Identifier_Reference


function Enclosing_Declaration (An_Identifier : in Identifier_Definition)
  return Declaration;

-- Returns the enclosing declaration for the identifier_definition.
--
-- This operations acts as the inverse of the ASIS_Declarations.Identifiers
-- operation when given an Identifier_Defintion.
--
-- Appropriate Kinds:
--   Any element where:
--     Id_Kind(element) = An_Identifier_Definition


-------------------------------------------------------------------
-- REFERENCES OF NAMED ENTITIES


subtype Identifier_List is ASIS_Ada_Program.Element_List;

function References
   ( Named_Entity : in Element;
     Context      : in Element )
  return Identifier_List;

-- Returns all direct references of the given Element within the given
-- context.
--
-- Appropriate Elements:
--   An Identifier_Definition, Identifier_Reference, Name_Expression, or
```

```
--   expression whose ASIS_Names_And_Expressions.Kind is:
--     A_Simple_Name,
--     An_Operator_Symbol,
--     A_Character_Literal,
--     An_Enumeration_Literal,


function Is_Referenced
   (Named_Entity : in Element;
    Context      : in Element )
  return Boolean;

-- Returns true if the Named_Entity is referenced within the given
-- context.
--
-- The search is discontinued after first reference is found.
--
-- Appropriate Elements:
--   An Identifier_Definition, Identifier_Reference, or Name_Expression
--   whose Name_And_Expression_Kind is:
--     A_Simple_Name,
--     An_Operator_Symbol,
--     A_Character_Literal,
--     An_Enumeration_Literal,


subtype Expression_List is ASIS_Ada_Program.Element_List;

function Used_Names (An_Expression : in Expression) return Expression_List;

-- Returns a list of names of objects/types, operators, and literals
-- in an expression.
-- EG. the expression (A + B.D (Q'(4))) would return the list :
--      A   : A_SIMPLE_NAME
--      +   : AN_OPERATOR_SYMBOL
--      B.D : A_SELECTED_COMPONENT
--      Q   : A_SIMPLE_NAME
--      4   : AN_INTEGER_LITERAL
--
-- The list may vary from one ASIS implementation to another
-- due to constant folding or other vendor dependent normalizations.
-- However, the lists are always semantically equivalent.
--
-- Raises Inappropriate_Program_Element
-- if Kind(An_Expression) = Not_An_Expression
```

```
-----------------------------------------------------------------------
-- DEFINITION OF REFERENCED NAME


function Definition (A_Reference : in Element) return Identifier_Definition;

-- Returns the first definition of the given reference.
--
-- A Nil_Element is returned if the element references a declaration that
-- is either predefined or implicit.
--
-- If given an Identifier_Definition, the same element is returned.
--
-- Appropriate Expression Kinds:
--   A_Simple_Name,
--   An_Operator_Symbol,
--   A_Character_Literal,
--   An_Enumeration_Literal,
--   An_Indexed_Component,
--   A_Slice,
--   A_Selected_Component,
--   An_Attribute,


function Enclosing_Name_Expression (An_Identifier : in Identifier_Reference)
  return Name_Expression;

-- Returns the outermost name_expression that contains the
-- identifier_reference.
--
-- Appropriate Kinds:
--   Any element where:
--     Id_Kind(element) = An_Identifier_Reference


-----------------------------------------------------------------------
-- DECLARATIONS OF NAMED ENTITIES

function Is_Predefined (Named_Entity : in Name_Expression)
  return Boolean;

-- Returns true if the named entity denotes an element that is
-- predefined. (EG. STANDARD."+")
```

```
--
-- Appropriate Expression Kinds:
--   A_Simple_Name,
--   An_Operator_Symbol,
--   A_Character_Literal,
--   An_Enumeration_Literal,
--   An_Indexed_Component,
--   A_Slice,
--   An_Selected_Component,
--   An_Attribute
--   A_Type_Conversion
--   A_Qualified_Expression
--   A_Function_Call


function Is_Implicitly_Declared
   (Named_Entity : in Name_Expression)
  return Boolean;

-- Returns true if the named entity has no explicit
-- declaration associated with it. (EG. An implicit declaration of
-- a derived subprogram).
--
-- Appropriate Expression Kinds:
--   A_Simple_Name,
--   An_Operator_Symbol,
--   A_Character_Literal,
--   An_Enumeration_Literal,
--   An_Indexed_Component,
--   A_Slice,
--   An_Selected_Component,
--   An_Attribute
--   A_Type_Conversion
--   A_Qualified_Expression
--   A_Function_Call


function Named_Declaration
   (Named_Entity : in Name_Expression)
  return Declaration;

-- Returns the declaration of the entity denoted by the name.
--
-- A Nil_Element is returned if the element references a declaration that
-- is either predefined or implicit.
```

```
--
-- Appropriate Name_Expression Kinds:
--   A_Simple_Name,
--   An_Operator_Symbol,
--   A_Character_Literal,
--   An_Enumeration_Literal,
--   An_Indexed_Component,
--   An_Selected_Component,
--   A_Slice,
--   An_Attribute


-------------------------------------------------------------------
-- NAMES - LRM 4.1
--
-- Simple_Names and operator symbols are instances of identifier references


function Prefix (Name : in Name_Expression) return Name_Expression;

-- Returns the prefix (the construct to the left of the rightmost
-- left parenthesis in indexed or sliced objects, the rightmost 'dot' for
-- selected components, or the rightmost tick for attributes).
--
-- Appropriate Name_Expression Kinds:
--   An_Indexed_Component,
--   A_Slice,
--   A_Selected_Component,
--   An_Attribute,
--   A_Function_Call


-------------------------------------------------------------------
-- INDEXED COMPONENTS - LRM 4.1.1

function Index_Expressions (Indexed_Component : in Name_Expression)
  return Expression_List;

-- Returns a list of expressions (possibly only one) within the parenthesis.
--
-- Appropriate Name_Expression Kinds:
--   An_Indexed_Component,


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

```
-- SLICES - LRM 4.1.2

subtype Discrete_Range is ASIS_Ada_Program.Element;
  -- Use ASIS_Type_Definitions operations to analyze discrete ranges.

function Slice_Range (Slice : in Name_Expression) return Discrete_Range;

-- Returns the discrete range of the slice.
--
-- Appropriate Name_Expression Kinds:
--   A_Slice,


-------------------------------------------------------------------- ---
-- SELECTED COMPONENTS - LRM 4.1.3

type Selection_Kinds is

   (A_Discriminant,          -- LRM 4.1.3 ( )
    A_Record_Component,      -- LRM 4.1.3 (b)
    A_Task_Entry,            -- LRM 4.1.3 (c)
    An_Access_Object,        -- LRM       (d)
    An_Expanded_Name,        -- LRM       (e,f)
    Not_A_Selection_Kind.

function Selection_Kind (Selected_Component : in Name_Expression)
  return Selection_Kinds;

-- Returns the selection kind of the selected component.
--
-- Returns Not_A_Selection_Kind for any inappropriate element such as
-- a Statement, Declaration, or Nil_Element.


function Selector (Selected_Component : in Name_Expression)
  return Name_Expression;

-- Returns the selector (the construct to the right of the rightmost
-- 'dot' in the selected component).
--
-- Appropriate Name_Expression Kinds:
--   A_Selected_Component whose Selection_Kind is:
--     A_Discriminant,
--     A_Record_Component,
--     A_Task_Entry,
```

```
--     An_Expanded_Name,


function Record_Object
   (Discriminant_Or_Component_Selection : in Name_Expression)
  return Declaration;

-- Returns the record object declaration for the selected object.
--
-- Appropriate Name_Expression Kinds:
--   A_Selected_Component whose Selection_Kind is:
--     A_Discriminant,
--     A_Record_Component,


function Selected_Component
   (Discriminant_Or_Component_Selection : in Name_Expression)
  return Declaration;

-- Returns the declaration of the Discriminant_Specification or
-- Component_Declaration in the record type declaration.
--
-- Operations in the package ASIS_Declarations can be used to
-- manipulate record components.
--
-- Appropriate Name_Expression Kinds:
--   A_Selected_Component whose Selection_Kind is:
--     A_Discriminant,
--     A_Record_Component,


function Selected_Task_Entry (Task_Entry_Selection : in Name_Expression)
  return Declaration;

-- Returns the entry declaration within the task type.
--
-- Appropriate Name_Expression Kinds:
--   A_Selected_Component whose Selection_Kind is:
--     A_Task_Entry,


function Selected_Designated_Subtype
   (Access_Object_Selection : in Name_Expression)
  return Declaration;
```

```
-- Returns the declaration of the designated subtype.  The base type of
-- this subtype is called the designated type.  The designated type is
-- returned in the absence of a subtype.
--
-- Appropriate Name_Expression Kinds:
--   A_Selected_Component whose Selection_Kind is:
--   An_Access_Object


subtype Name_Expression_List is ASIS_Ada_Program.Element_List;

function Named_Packages (A_Use_Clause : in Element)
  return Name_Expression_List;

-- Returns a list containing the Name_Expression of each package
-- mentioned in the use clause.
--
-- Results of this operation may vary across ASIS implementations.
-- Some implementations normalize all use clauses containing multiple
-- package names into an equivalent sequence of corresponding single use
-- clauses.  Similarly, an implementation may keep a name only once even
-- though that name may appear more than once in a use clause.
-- See LRM 8.4(9).
--
-- Appropriate Kinds:
--   Any element whose ASIS_Elements.Major_Element_Kind is A_Use_Clause.


-----------------------------------------------------------------------
-- ATTRIBUTES - LRM 4.1.4

function Attribute_Designator_Name (Attribute : in Name_Expression)
  return Name_Expression;

-- Returns the name expression of the attribute designator.  The name
-- expression may itself be an attribute if the given attribute is of the
-- form T'BASE'FIRST.
--
-- The attribute simple name returned here only has a name if its
-- kind = A_Simple_Name.  It is inappropriate in all other queries.
--
-- Appropriate Name_Expression Kinds:
--   An_Attribute
--
-- Further Analysis:
```

```
--  Use String_Name to get the name as a string.


function Attribute_Designator_Argument (Attribute : in Name_Expression)
  return Expression;

-- Returns the static expression associated with the optional argument
-- of the attribute designator if one exists, Nil_Element otherwise.
--
-- Appropriate Name_Expression Kinds:
--   An_Attribute


-------------------------------------------------------------------
-- LITERALS - LRM 4.2
--
-- The image of literals can be determined by using the Static_Value
-- operation.


function Is_Literal (An_Expression : in Expression) return Boolean;

-- Determines whether the expression is a literal.
--
-- Returns True if the expression is:
--   A_Character_Literal,
--   An_Enumeration_Literal,
--   A_Null_Literal,
--   A_String_Literal,
--   An_Integer_Literal,
--   A_Real_Literal,
--
-- Raises Inappropriate_Program_Element
-- if Kind(An_Expression) = Not_An_Expression


function Position_Number_Image
   (Enumeration_Or_Character_Literal : in Expression)
  return String;

-- Returns the image of the cardinality of the enumeration literal within
-- the base type of the enumeration type. (same as image of "'POS")
--
-- Appropriate Expression Kinds:
--   A_Character_Literal,
```

```
--   An_Enumeration_Literal,


function Representation_Value_Image
   (Enumeration_Or_Character_Literal : in Expression)
  return String;

-- Returns the image of the internal representation of the enumeration
-- literal.
-- (same as image of "'POS" if no rep spec defined for the enumeration type)
--
-- Appropriate Expression Kinds:
--   A_Character_Literal,
--   An_Enumeration_Literal,


function Enumeration_Type_Declaration
   (Enumeration_Or_Character_Literal : in Expression)
  return Declaration;

-- Returns the declaration of the type that contains the literal.
-- Since characters are enumerations, both regular enumeration and
-- character literals have enumeration base type declarations.
--
-- Returns a Nil_Element if
--   Is_Implicitly_Declared(Enumeration_Or_Character_Literal).
--
-- The operations in ASIS_Declarations can be used to further analyze
-- these declarations.
--
-- Appropriate Expression Kinds:
--   A_Character_Literal,
--   An_Enumeration_Literal,


-------------------------------------------------------------------------
-- AGGREGATES - LRM 4.3


subtype Component_Association is ASIS_Ada_Program.Element;

subtype Component_Association_List is ASIS_Ada_Program.Element_List;

function Components
   (Aggregate  : in Expression;
```

```
    Normalized : in Boolean := False)
  return Component_Association_List;

-- Returns a list of the component associations of an aggregate.
-- NOTE THAT NORMALIZED INFO IS NOT AVAILABLE FOR ARRAY AGGREGATES.
-- For record aggregates, the Normalized parameter has the following
-- affect:
--
-- If Normalized is True:
--   A normalized list containing all of the component associations
--   of an aggregate is returned in positional notation.
--
-- If Normalized is False:
--   Only explicit component associations will be included in the
--   order they appeared in the program text.
--
-- An ASIS implementation may choose to always "Normalize" its internal
-- representation.  If that is the case, the operation will always behave
-- as if Normalized is True.
--
-- Appropriate Expression Kinds:
--   An_Aggregate


function Components_Normalized return Boolean;

-- Returns True if this implementation will always "Normalize" record
-- aggregate components as described in the Components operation.
-- If that is the case, the Components operation will always behave as
-- if its parameter Normalized is True.


subtype Choice_List is ASIS_Ada_Program.Element_List;

function Component_Choices (Component : in Component_Association)
  return Choice_List;

-- Returns the list of choices in the aggregate component.
-- The list will be a nil list if positional notation is used.
--
-- Further Analysis:
--   Use the CHOICE operations in ASIS_Type_Definitions
--
-- Raises Inappropriate_Program_Element
-- if ASIS_Elements.Major_Element_Kind(Component) /= A_Component_Association
```

```
function Component_Expression (Component : in Component_Association)
  return Expression;

-- Returns the expression for the component association.
--
-- Raises Inappropriate_Program_Element
-- if ASIS_Elements.Major_Element_Kind(Component) /= A_Component_Association


-------------------------------------------------------------------------
-- PARENTHESIZED EXPRESSIONS - LRM 4.4


function Expression_Parenthesized (Parenthesized_Expression : in Expression)
  return Expression;

-- Returns the expression within the parenthesis.
--
-- Appropriate Expression Kinds:
--   A_Parenthesized_Expression


-------------------------------------------------------------------------
-- FUNCTION CALLS
--   AS A PRIMARY - LRM 4.4
--   OPERATORS    - LRM 4.5


function Is_Prefix_Call (Function_Call : in Expression) return Boolean;

-- Returns true if the function call is in prefix form.
-- EG. - Foo (A, B);   -- Returns TRUE
--       "<" (A, B);   -- Returns TRUE
--       ... A < B ... -- Returns FALSE
--
-- Appropriate Expression Kinds:
--   A_Function_Call


function Is_Prefix_Call_Supported return Boolean;

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

```
function Called_Function (Function_Call : in Expression) return Declaration;

-- Returns the declaration of the called function.
--
-- Returns a Nil_Element if
--   Is_Implicitly_Declared(Function_Call).
--
-- Appropriate Expression Kinds:
--   A_Function_Call


subtype Parameter_Association_List is ASIS_Ada_Program.Element_List;

function Function_Call_Parameters
   (Function_Call    : in Expression;
    Include_Defaults : in Boolean := False)
  return Parameter_Association_List;

-- Returns a list of parameter associations for the call.
--
-- If Include_Defaults is True:
--   The (unspecified) default parameters will also be included and,
--   the list will be ordered to match the declaration order of the
--   corresponding formal parameter specifications.
--
-- If Include_Defaults is False:
--   Only explicit actual parameters will be included in the order they
--   appeared in the program text.
--
-- An ASIS implementation may choose to always "Include Defaults" its
-- internal representation.  If that is the case, the operation will
-- always behave as if Include_Defaults is True.
--
-- Use ASIS_Statements operations to decompose parameter associations.
--
-- To map actual parameters to corresponding formal parameters:
--
--   Create an actual parameter list by applying the function
--   ASIS_Statements.Actual_Parameter to each parameter association
--   returned by Function_Call_Parameters with Include_Defaults = True.
--
--   Create a formal parameter list by applying the function
--   ASIS_Declarations.Identifiers to each formal parameter
```

```
--   specification returned by ASIS_Declarations.Parameters.
--
--   The Expression elements in the actual parameters list will have a
--   one to one correspondence with the Identifier_Definition elements
--   in the formal parameter list.
--
-- Appropriate Expression Kinds:
--   A_Function_Call
--
-- Further Analysis:
--  Use ASIS_Statements operations Formal_Parameter and Actual_Parameter
--  to analyze the parameter associations.


function Function_Call_Parameters_Include_Defaults return Boolean;

-- Returns True if this implementation will always "Include Defaults".
-- If that is the case, the Function_Call_Parameters operation will
-- always behave as if its parameter Include_Defaults is True.


function Is_Parameter_Association (Program_Element : in Element)
  return Boolean;

-- Returns True if the element is a parameter association.
--
-- Raises Inappropriate_Program_Element if the element is a Nil_Element.


-------------------------------------------------------------------------
-- SPECIAL OPERATIONS
--   SHORT CIRCUIT CONTROL FORMS - LRM 4.5.1
--   MEMBERSHIP TESTS            - LRM 4.5.2


type Special_Operation_Kinds is

   (An_In_Range,
    A_Not_In_Range,
    An_In_Type,
    A_Not_In_Type,
    An_And_Then,
    An_Or_Else,
    Not_A_Special_Operation);
```

```
function Special_Operation_Kind (Special_Operation : in Expression)
  return Special_Operation_Kinds;

-- Returns the kind of the special operation.
--
-- Returns Not_A_Special_Operation for any inappropriate element such as
-- a Statement, Declaration, or Nil_Element.


function Special_Operation_Left_Hand_Side (Special_Operation : in Expression)
  return Expression;

-- All special operation left hand sides are expressions.
--
-- Appropriate Expression Kinds:
--   A_Special_Operation


subtype Range_Constraint is ASIS_Ada_Program.Element;

function In_Range_Operation_Right_Hand_Side
   (Special_Operation : in Expression)
  return Range_Constraint;

-- The right hand side for an IN_RANGE operation is a range
-- Further Analysis:
--   Use Constraint_Kind, Lower_Bound, and Upper_Bound operations
--   in ASIS_Type_Definitions.
--
-- Appropriate Special_Operation Kinds:
--   An_In_Range,
--   A_Not_In_Range


function In_Type_Operation_Right_Hand_Side
   (Special_Operation : in Expression)
  return Name_Expression;

-- The right hand side for an IN_TYPE operation is a type mark which is
-- a name and can be further analyzed using this package.
--
-- Appropriate Special_Operation Kinds:
--   An_In_Type,
--   A_Not_In_Type
```

```
function Short_Circuit_Operation_Right_Hand_Side
   (Special_Operation : in Expression)
  return Expression;

-- The right hand side for a short circuit operation can be any
-- expression kind which can be further analyzed using this package.
--
-- Appropriate Special_Operation Kinds:
--   An_And_Then,
--   An_Or_Else


-----------------------------------------------------------------
-- EXPRESSION EVALUATION - LRM 4.5
-- STATIC VALUES


function Is_Static (An_Expression : in Expression) return Boolean;

-- Returns True if the expression has a static value.
--
-- Raises Inappropriate_Program_Element
-- if Kind(An_Expression) = Not_An_Expression


function Static_Value (An_Expression : in Expression) return String;

-- If Is_Static(An_Expression) then
--   Returns the string image of the value of the expression.
-- else returns a null string.
--
-- Note that Static_Value for strings will not return the quotes around
-- a string literal.  However, quotes embedded in a string literal are
-- doubled, just as they appear in the program text.
--
-- Raises Inappropriate_Program_Element
-- if Kind(An_Expression) = Not_An_Expression


-----------------------------------------------------------------
-- TYPE CONVERSIONS and QUALIFIED EXPRESSIONS
-- LRM 4.6, 4.7
```

```
function Type_Mark (Type_Conversion_Or_Qualified_Expression : in Expression)
  return Name_Expression;

-- Returns the type mark of the type to which the expression is
-- being converted or qualified.
--
-- Appropriate Expression Kinds:
--   A_Type_Conversion
--   A_Qualified_Expression


function Converted_Or_Qualified_Expression
   (Type_Conversion_Or_Qualified_Expression : in Expression)
  return Expression;

-- Returns the expression being converted or qualified.
--
-- Appropriate Expression Kinds:
--   A_Type_Conversion
--   A_Qualified_Expression


------------------------------------------------------------------------
-- ALLOCATORS LRM 4.8


subtype Subtype_Indication is ASIS_Ada_Program.Element;
  -- Use operations in this package and in ASIS_Type_Definitions.

function Allocation_Type (Allocator : in Expression)
  return Subtype_Indication;

-- Returns the subtype indication for the object being allocated.
--
-- Appropriate Expression Kinds:
--   An_Allocation_From_Subtype


function Qualified_Object_Expression (Allocator : in Expression)
  return Expression;

-- Returns the qualified expression for the object being allocated.
--
-- Appropriate Expression Kinds:
--   An_Allocation_From_Qualified_Expression
```

```
end ASIS_Names_And_Expressions;
```

```
-------------------------------------------------------------------------
-- ASIS PRELIMINARY REVIEW VERSION 0.4
-- Sept 27, 1991
-------------------------------------------------------------------------

with ASIS_Ada_Program;
package ASIS_Statements is

  -- LRM Chapters 5, 6, 9, 11

  subtype Element is ASIS_Ada_Program.Element;
    -- Use ASIS_Elements operations.

  subtype Statement is ASIS_Ada_Program.Element;
    -- Use ASIS_Statements operations.

  subtype Declaration is ASIS_Ada_Program.Element;
    -- Use ASIS_Declarations operations.

  subtype Expression is ASIS_Ada_Program.Element;
    -- Use ASIS_Names_And_Expression operations.

  subtype Name_Expression is ASIS_Ada_Program.Element;
    -- Use ASIS_Names_And_Expression operations.

  subtype Type_Definition is ASIS_Ada_Program.Element;
    -- Use ASIS_Type_Definitions operations.

  subtype Identifier_Definition is ASIS_Ada_Program.Element;
    -- Use ASIS_Names_And_Expression operations.

  subtype Identifier_Reference is ASIS_Ada_Program.Element;
    -- Use ASIS_Names_And_Expression operations.
```

```
-------------------------------------------------------------------------
-- STATEMENT KINDS


type Statement_Kinds is

   -- Simple statements:

   (A_Null_Statement,
    An_Assignment_Statement,
    A_Procedure_Call_Statement,
    An_Exit_Statement,
    A_Return_Statement,
    A_Goto_Statement,
    An_Entry_Call_Statement,
    A_Delay_Statement,
    An_Abort_Statement,
    A_Raise_Statement,
    A_Code_Statement,

    -- Compound statements:

    An_If_Statement,
    A_Case_Statement,
    A_Loop_Statement,
    A_Block_Statement,
    An_Accept_Statement,
    A_Selective_Wait_Statement,
    A_Conditional_Entry_Call_Statement,
    A_Timed_Entry_Call_Statement,
    Not_A_Statement);


function Kind (A_Statement : in Statement) return Statement_Kinds;

-- Returns the kind of the statement.
--
-- Returns Not_A_Statement for any inappropriate element such as
-- a Type_Definition, Declaration, or Nil_Element.


-------------------------------------------------------------------------
-- LABELED STATEMENTS - LRM 5.1
```

```
function Is_Labeled (A_Statement : in Statement) return Boolean;

-- Returns True if the statement has a label.
--
-- Raises Inappropriate_Program_Element
-- if Kind(A_Statement) = Not_A_Statement


function Label_Names (A_Statement : in Statement) return String;

-- Returns the Label_Name(s) of the Statement.
-- The << and >> delimiters are included in the string.
-- A null string is returned if no label is present
--
-- Raises Inappropriate_Program_Element
-- if Kind(A_Statement) = Not_A_Statement


-------------------------------------------------------------------
-- ASSIGNMENT STATEMENTS - LRM 5.2


function Object_Assigned_To (Assignment_Statement : in Statement)
  return Name_Expression;

-- Returns the name of object to which the assignment is being made.
--
-- Appropriate Statement Kinds:
--   An_Assignment_Statement


function Assignment_Expression (Assignment_Statement : in Statement)
  return Expression;

-- Returns the expression on the right hand side of the assignment.
--
-- Appropriate Statement Kinds:
--   An_Assignment_Statement


-------------------------------------------------------------------
-- IF STATEMENTS - LRM 5.3
```

```
subtype If_Statement_Arm is ASIS_Ada_Program.Element;

type If_Statement_Arm_Kinds is

   (An_If_Arm,
    An_Elsif_Arm,
    An_Else_Arm,
    Not_An_If_Statement_Arm);

function If_Statement_Arm_Kind (Arm : in If_Statement_Arm)
  return If_Statement_Arm_Kinds;

-- Returns the kind of the if statement arm.
--
-- Returns Not_An_If_Statement_Arm for any inappropriate element such as
-- a Type_Definition, Declaration, or Nil_Element.


subtype If_Statement_Arm_List is ASIS_Ada_Program.Element_List;

function If_Statement_Arms (If_Statement : in Statement)
  return If_Statement_Arm_List;

-- Returns a list of the arms of the if statement in their order of
-- appearance.
--
-- Appropriate Statement Kinds:
--   An_If_Statement


function Condition_Expression (Arm : in If_Statement_Arm) return Expression;

-- Returns the condition expression for an if arm or an elsif arm.
--
-- Appropriate If_Statement_Arm Kinds:
--     An_If_Arm
--     An_Elsif_Arm


subtype Statement_List is ASIS_Ada_Program.Element_List;

function Arm_Statements (Arm : in If_Statement_Arm) return Statement_List;

-- Returns a list of the statements in an if statement arm.
```

```
--
-- Raises Inappropriate_Program_Element
-- if If_Statement_Arm_Kind(Arm) = Not_An_If_Statement_Arm


-------------------------------------------------------------------
-- CASE STATEMENTS - LRM 5.4


function Case_Expression (Case_Statement : in Statement) return Expression;

-- Returns the expression of the case statement that determines
-- which alternative will be taken.
--
-- Appropriate Statement Kinds:
--   A_Case_Statement


subtype Case_Statement_Alternative_List is ASIS_Ada_Program.Element_List;

function Case_Statement_Alternatives (Case_Statement : in Statement)
  return Case_Statement_Alternative_List;

-- Return a list of all alternatives of the case statement in their
-- order of appearance.
--
-- Appropriate Statement Kinds:
--   A_Case_Statement


subtype Case_Statement_Alternative is ASIS_Ada_Program.Element;

function Is_When_Others (Case_Alternative : in Case_Statement_Alternative)
  return Boolean;

-- Returns True if the case statement alternative is an others choice.
--
-- Appropriate Major_Element Kinds:
--   A_Case_Statement_Alternative


subtype Choice_List is ASIS_Ada_Program.Element_List;

function Case_Statement_Alternative_Choices
   (Case_Alternative : in Case_Statement_Alternative)
```

```
  return Choice_List;

-- Returns a list of the 'WHEN <choice> | <choice>' choices in their
-- order of appearance.
--
-- Further Analysis:
--   Use ASIS_Type_Definitions Choices queries to extract more information.
--
-- Appropriate Major_Element Kinds:
--   A_Case_Statement_Alternative


function Case_Statement_Alternative_Statements
   (Case_Alternative : in Case_Statement_Alternative)
  return Statement_List;

-- Returns a list of the statements in the case statement alternative.
--
-- Appropriate Major_Element Kinds:
--   A_Case_Statement_Alternative


-------------------------------------------------------------------------
-- LOOP STATEMENTS - LRM 5.5


type Loop_Kinds is

   (A_For_Loop,
    A_While_Loop,
    A_Simple_Loop,
    Not_A_Loop);

function Loop_Kind (Loop_Statement : in Statement) return Loop_Kinds;

-- Returns the kind of the if loop statement.
--
-- Returns Not_A_Loop for any inappropriate element such as
-- a Type_Definition, Declaration, or Nil_Element.


function Loop_Simple_Name (Loop_Statement : in Statement)
  return Identifier_Definition;

-- Returns the Identifier_Definition of the simple name of the loop.
```

```
   -- A Nil_Element is returned if the loop has no name.
   --
   -- Appropriate Statement Kinds:
   --   A_Loop_Statement


   function While_Condition (Loop_Statement : in Statement) return Expression;

   -- Returns the condition expression associated with the while loop.
   --
   -- Appropriate Statement Kinds:
   --   A_Loop_Statement whose Loop_Kind is A_While_Loop


   function For_Loop_Parameter_Specification (Loop_Statement : in Statement)
     return Declaration;

   -- Returns the Declaration of the Loop_Parameter_Specification.
   --
   -- Further Analysis:
   --   Use ASIS_Declarations operation Identifiers to get the
   --     Identifier_Definition of the loop parameter.
   --
   -- Appropriate Statement Kinds:
   --   A_Loop_Statement whose Loop_Kind is A_For_Loop


   subtype Discrete_Range is ASIS_Ada_Program.Element;
     -- Use ASIS_Type_Definitions operations to analyze discrete ranges.

   function Loop_Parameter_Range (Loop_Parameter_Specification : in Declaration)
     return Discrete_Range;

   -- Returns the Discrete_Range of the loop parameter.
   --
   -- Appropriate Declaration Kinds:
   --   A_Loop_Parameter_Specification


   function Is_Reverse_Loop_Parameter
      (Loop_Parameter_Specification : in Declaration)
     return Boolean;

   -- Returns True if the reserved word reverse is present.
   --
```

```
   -- Appropriate Declaration Kinds:
   --   A_Loop_Parameter_Specification


   function Loop_Statements (Loop_Statement : in Statement)
     return Statement_List;

   -- Returns a list of the statements in the loop statement.
   --
   -- Appropriate Statement Kinds:
   --   A_Loop_Statement


   ------------------------------------------------------------------------
   -- BLOCK STATEMENTS - LRM 5.6


   function Block_Simple_Name (Block_Statement : in Statement)
     return Identifier_Definition;

   -- Returns the Identifier_Definition of the simple name of the block.
   -- A Nil_Element is returned if the block has no explicit name.
   --
   -- Appropriate Statement Kinds:
   --   A_Block_Statement


   subtype Declarative_Item_List is ASIS_Ada_Program.Element_List;

   function Declarative_Items (Block_Statement : in Statement)
     return Declarative_Item_List ;

   -- Returns a list of the declarations, representation specifications,
   -- and use clauses in the declarative part of the block.
   -- A empty list indicates that there are no declarative items.
   --
   -- Appropriate Statement Kinds:
   --   A_Block_Statement


   function Block_Body_Statements (Block_Statement : in Statement)
     return Statement_List;

   -- Returns a list of the statements in the block statement.
   --
```

```
   -- Appropriate Statement Kinds:
   --   A_Block_Statement


   subtype Exception_Handler_List is ASIS_Ada_Program.Element_List;

   function Block_Exception_Handlers (Block_Statement : in Statement)
     return Exception_Handler_List;

   -- Returns a list of the exception handlers of the block.
   --
   -- Appropriate Statement Kinds:
   --   A_Block_Statement


   ------------------------------------------------------------------
   -- EXIT STATEMENTS - LRM 5.7


   function Exit_Loop_Name (Exit_Statement : in Statement)
     return Identifier_Reference;

   -- Returns the name of the exited loop.
   -- A Nil_Element is returned if no loop name is present.
   --
   -- Appropriate Statement Kinds:
   --   An_Exit_Statement


   function Exit_Condition (Exit_Statement : in Statement) return Expression;

   -- Returns the when condition of the exit statement.
   -- A Nil_Element is returned if no condition is present.
   --
   -- Appropriate Statement Kinds:
   --   An_Exit_Statement


   function Loop_Exited (Exit_Statement : in Statement) return Statement;

   -- Returns the loop statement exited by the exit statement.
   --
   -- Appropriate Statement Kinds:
   --   An_Exit_Statement
```

```
-----------  ------------------------------------------------------
-- RETURN STATEMENTS - LRM 5.8

function Return_Expression (Return_Statement : in Statement)
  return Expression;

-- Returns the expression in the return statement.
-- A Nil_Element is returned if no expression is present.
--
-- Appropriate Statement Kinds:
--   A_Return_Statement


-------------------------------------------------------------------------
-- GOTO STATEMENTS - LRM 5.9

function Goto_Label (Goto_Statement : in Statement) return String;

-- Returns the name of label specified by the goto statement.
--
-- Appropriate Statement Kinds:
--   A_Goto_Statement


function Destination_Statement (Goto_Statement : in Statement)
  return Statement;

-- Returns the named target statement specified by the goto statement.
--
-- Appropriate Statement Kinds:
--   A_Goto_Statement


-------------------------------------------------------------------------
-- PROCEDURE CALL STATEMENTS - LRM 6.4


function Called_Name (Procedure_Or_Entry_Call_Statement : in Statement)
  return Name_Expression;

-- Returns the name of the called procedure or entry.  The name of an
-- entry of a family takes the form of an indexed component.
--
-- Appropriate Statement Kinds:
```

```
--   A_Procedure_Call_Statement
--   An_Entry_Call_Statement


function Is_Implicitly_Declared
   (Procedure_Or_Entry_Call_Statement : in Statement)
  return Boolean;

-- Returns true if the procedure or entry call statement has no explicit
-- declaration associated with it. (EG. An implicit declaration of
-- a derived subprogram or an entry of a derived task type).


function Called_Procedure (Procedure_Or_Entry_Call_Statement : in Statement)
  return Declaration;

-- Returns the declaration of the called procedure or entry.
--
-- If Is_Implicitly_Declared(Procedure_Or_Entry_Call_Statement) then
-- a Nil_Element is returned.
--
-- Appropriate Statement Kinds:
--   A_Procedure_Call_Statement
--   An_Entry_Call_Statement


------------------------------------------------------------------------
-- PARAMETER ASSOCIATION - LRM 6.4

subtype Parameter_Association is ASIS_Ada_Program.Element;

subtype Parameter_Association_List is ASIS_Ada_Program.Element_List;

function Procedure_Call_Parameters
   (Procedure_Or_Entry_Call_Statement : in Statement;
    Include_Defaults                  : in Boolean := False)
  return Parameter_Association_List;

-- Returns a list of parameter associations for the call.
--
-- If Include_Defaults is True:
--   The (unspecified) default parameters will also be included and,
--   the list will be ordered to match the declaration order of the
--   corresponding formal parameter specifications.
--
```

```
-- If Include_Defaults is False:
--   Only explicit actual parameters will be included in the order they
--   appeared in the program text.
--
-- An ASIS implementation may choose to always "Include Defaults" its
-- internal representation.  If that is the case, the operation will
-- always behave as if Include_Defaults is True.
--
-- Appropriate Statement Kinds:
--   A_Procedure_Call_Statement
--   An_Entry_Call_Statement


function Procedure_Call_Parameters_Include_Defaults return Boolean;

-- Returns True if this implementation will always "Include Defaults".
-- If that is the case, the Procedure_Call_Parameters operation will
-- always behave as if its parameter Include_Defaults is True.


function Formal_Parameter (A_Parameter_Association : in Parameter_Association)
  return Identifier_Reference;

-- Returns the formal parameter simple name.
-- Accepts parameter associations and generic associations.
--
-- A Nil_Element is returned if the parameter association was given in
-- positional notation.
--
-- An ASIS implementation may choose to always normalize named notation
-- to positional notation in procedure and entry calls.
--
-- Appropriate Major_Element Kinds:
--   A_Parameter_Association
--   An_Argument_Association


function Is_Formal_Parameter_Named_Notation_Supported return Boolean;

-- Returns True if this implementation will always normalize named
-- notation to positional notation in subprogram and entry calls.
-- If that is the case, the Formal_Parameter operation will
-- always return a Nil_Element.
```

```
function Actual_Parameter (A_Parameter_Association : in Parameter_Association)
  return Expression;

-- Returns the actual parameter expression.
-- Accepts parameter associations and generic associations.
--
-- Appropriate Major_Element Kinds:
--   A_Parameter_Association
--   An_Argument_Association


-- To map actual parameters to corresponding formal parameters:
--
--   Create an actual parameter list by applying the function
--   Actual_Parameter to each parameter association returned by
--   Procedure_Call_Parameters with Include_Defaults = True.
--
--   Create a formal parameter list by applying the function
--   ASIS_Declarations.Identifiers to each formal parameter
--   specification returned by ASIS_Declarations.parameters.
--
--   The Expression elements in the actual parameters list will have a
--   one to one correspondence with the Identifier_Definition elements
--   in the formal parameter list.


-------------------------------------------------------------------------
-- ENTRY CALL STATEMENTS - LRM 9.5

-- Use PROCEDURE CALL STATEMENTS operations to analyze entry calls.


function Family_Index (Entry_Call_Or_Accept_Statement : in Statement)
  return Expression;

-- Returns the entry index in the entry call statement or accept statement.
-- If the statement has no explicit entry index, a nil element is returned.
--
-- Appropriate Statement Kinds:
--   An_Entry_Call_Statement
--   An_Accept_Statement


-------------------------------------------------------------------------
-- ACCEPT STATEMENTS - LRM 9.5
```

```
function Accepted_Entry (Accept_Statement : in Statement)
  return Declaration;

-- Returns the declaration of the entry accepted in this statement.
--
-- Appropriate Statement Kinds:
--   An_Accept_Statement


function Accept_Entry_Simple_Name (Accept_Statement : in Statement)
  return Identifier_Reference;

-- Returns the simple name of the entry following the reserved word accept.
--
-- Appropriate Statement Kinds:
--   An_Accept_Statement


subtype Parameter_Specification_List is ASIS_Ada_Program.Element_List;

function Accept_Parameters (Accept_Statement : in Statement)
  return Parameter_Specification_List;

-- Returns a list of parameter specifications in the formal part
-- of the accept statement in the order of appearance.
--
-- A Nil_Element_List is returned if the accept statement has no
-- parameters.
--
-- Results of this operation may vary across ASIS implementations.
-- Some implementations normalize all multiple parameter specifications into
-- an equivalent sequence of corresponding single parameter specifications.
-- See LRM 6.1(4).
--
-- Further Analysis:
--   Use Identifiers to obtain the identifier list.
--   Use Type_Mark to obtain the parameter type mark.
--   Use Is_Initialized and Initial_Value to query the information
--     related to the presence of the default parameter initialization.
--
-- Appropriate Statement Kinds:
--   An_Accept_Statement
```

```
function Accept_Body_Statements (Accept_Statement : in Statement)
  return Statement_List;

-- Returns a list of the statements in the accept statement.
--
-- Appropriate Statement Kinds:
--   An_Accept_Statement


------------------------------------------------------------------------
-- DELAY STATEMENTS - LRM 9.6


function Delay_Expression (Delay_Statement : in Statement)
  return Expression;

-- Returns the expression for the duration of the delay
--
-- Appropriate Statement Kinds:
--   A_Delay_Statement


------------------------------------------------------------------------
-- SELECT STATEMENTS       - LRM 9.7
-- SELECTIVE WAIT          - LRM 9.7.1
-- CONDITIONAL ENTRY CALLS - LRM 9.7.2
-- TIMED ENTRY CALLS       - LRM 9.7.3


subtype Select_Statement_Arm is ASIS_Ada_Program.Element;

type Select_Statement_Arm_Kinds is

   (A_Selective_Wait_Select_Arm,
    A_Selective_Wait_Or_Arm,
    A_Conditional_Entry_Call_Select_Arm,
    A_Timed_Entry_Call_Select_Arm,
    A_Timed_Entry_Call_Or_Arm,
    An_Else_Arm,
    Not_A_Select_Statement_Arm);

function Select_Statement_Arm_Kind (Arm : in Select_Statement_Arm)
  return Select_Statement_Arm_Kinds;
```

```
-- Returns the kind of the select statement arm.
--
-- Returns Not_A_Select_Statement_Arm for any inappropriate element such as
-- a Type_Definition, Declaration, or Nil_Element.


subtype Select_Statement_Arm_List is ASIS_Ada_Program.Element_List;

function Select_Statement_Arms (Select_Statement : in Statement)
  return Select_Statement_Arm_List;

-- Returns a list of the arms of the Select statement in their order of
-- appearance.
--
-- Appropriate Statement Kinds:
--   A_Selective_Wait_Statement
--   A_Conditional_Entry_Call_Statement
--   A_Timed_Entry_Call_Statement



subtype Select_Alternative is ASIS_Ada_Program.Element;

type Select_Alternative_Kinds is

   (An_Accept_Alternative,
    A_Delay_Alternative,
    A_Terminate_Alternative,
    Not_A_Select_Alternative);

function Select_Alternative_Kind (Alternative : in Select_Alternative)
  return Select_Alternative_Kinds;

-- Returns the kind of the select alternative.
-- Returns Not_A_Select_Alternative for any inappropriate element such as
-- a Type_Definition, Declaration, or Nil_Element.


function Arm_Select_Alternative (Arm : in Select_Statement_Arm)
  return Select_Alternative;

-- Returns the select alternative in the select statement arm.
--
-- Appropriate Select_Statement_Arm Kinds:
--   A_Selective_Wait_Select_Arm
```

```
--   A_Selective_Wait_Or_Arm


function Is_Guarded (Alternative : in Select_Alternative) return Boolean;

-- Returns true if a select alternative has a guard.
--
-- Appropriate Select_Alternative Kinds:
--   An_Accept_Alternative
--   A_Delay_Alternative
--   A_Terminate_Alternative


function Guard (Alternative : in Select_Alternative) return Expression;

-- Returns the conditional expression guarding the alternative.
-- Returns a nil element if there is no guard.
--
-- Appropriate Select_Alternative Kinds:
--   An_Accept_Alternative
--   A_Delay_Alternative
--   A_Terminate_Alternative


function Select_Alternative_Statements (Alternative : in Select_Alternative)
  return Statement_List;

-- Returns a list of the statements in the the accept or delay alternative
-- including the accept statement and delay statements themselves.
--
-- Appropriate Select_Alternative Kinds:
--   An_Accept_Alternative
--   A_Delay_Alternative


function Or_Statements (Timed_Entry_Call_Or_Arm : in Select_Statement_Arm)
  return Statement_List;

-- Returns a list of statements contained in the or arm of a time_entry_call.
--
-- Appropriate Select_Statement_Arm Kinds:
--   A_Timed_Entry_Call_Or_Arm


function Else_Statemen    Else_Arm : in Select_Statement_Arm)
```

```
  return Statement_List;

-- Returns a list of statements contained in the else arm of a
-- selective_wait or conditional_entry_call.
--
-- If no else part exists, an empty list is returned.
--
-- Appropriate Select_Statement_Arm Kinds:
--   An_Else_Arm


function Entry_Call_Statements (Arm : in Select_Statement_Arm)
  return Statement_List;

-- Returns the statements associated with the conditional or timed entry
-- call, including the actual entry call statement.
--
-- Appropriate Select_Statement_Arm Kinds:
--   A_Conditional_Entry_Call_Select_Arm
--   A_Timed_Entry_Call_Select_Arm


------------------------------------------------------------------------
-- ABORT STATEMENTS - LRM 9.10


subtype Name_Expression_List is ASIS_Ada_Program.Element_List;

function Aborted_Tasks (Abort_Statement : in Statement)
  return Name_Expression_List;

-- Returns a list of the aborted tasks.
--
-- Appropriate Statement Kinds:
--   An_Abort_Statement

------------------------------------------------------------------------
-- EXCEPTION HANDLERS - LRM 11.2


subtype Exception_Handler is ASIS_Ada_Program.Element;

function Is_Others_Handler (Handler : in Exception_Handler) return Boolean;

-- Appropriate Major_Element Kinds:
```

```
--   An_Exception_Handler


function Exception_Choices (Handler : in Exception_Handler)
  return Choice_List;

-- Returns a list of the 'WHEN <choice> | <choice>' choices in their
-- order of appearance.
--
-- Further Analysis:
--   Use the Type_Definitions package's CHOICES queries to extract
--     further information about the <choice>s.
--
-- Appropriate Major_Element Kinds:
--   An_Exception_Handler


function Handler_Statements (Handler : in Exception_Handler)
  return Statement_List;

-- Returns a list of the statements in the exception handler.
--
-- Appropriate Major_Element Kinds:
--   An_Exception_Handler


------------------------------------------------------------------------
-- RAISE STATEMENTS - LRM 11.3

function Raised_Exception (Raise_Statement : in Statement)
  return Name_Expression;

-- Returns the name of the raised exception.
-- A Nil_Element is returned if there is no explicitly named exception.
--
-- Appropriate Statement Kinds:
--   A_Raise_Statement


------------------------------------------------------------------------
-- CODE STATEMENTS - LRM 13.8

function Qualified_Expression (Code_Statement : in Statement)
  return Expression;
```

```
   -- Returns the qualified expression representing the code statement.
   --
   -- Appropriate Statement Kinds:
   --   A_Code_Statement



end ASIS_Statements;
```

```
-------------------------------------------------------------------------------
-- ASIS PRELIMINARY REVIEW VERSION 0.4
-- Sept 27, 1991
-------------------------------------------------------------------------------


with ASIS_Ada_Program;
package ASIS_Representation_Clauses is

  -- LRM Chapter 13

  subtype Element is ASIS_Ada_Program.Element;
    -- Use ASIS_Elements operations.

  subtype Representation_Clause is ASIS_Ada_Program.Element;
    -- Use ASIS_Representation_Clauses operations.

  subtype Declaration is ASIS_Ada_Program.Element;
    -- Use ASIS_Declarations operations.

  subtype Expression is ASIS_Ada_Program.Element;
    -- Use ASIS_Names_And_Expression operations.

  subtype Type_Definition is ASIS_Ada_Program.Element;
    -- Use ASIS_Type_Definitions operations.

  subtype Identifier_Reference is ASIS_Ada_Program.Element;
    -- Use ASIS_Names_And_Expression operations.
```

```
subtype Identifier_Definition is ASIS_Ada_Program.Element;
  -- Use ASIS_Names_And_Expression operations.

-------------------------------------------------------------------
-- REPRESENTATION CLAUSE KINDS


type Representation_Clause_Kinds is

   (A_Length_Clause,
    An_Enumeration_Representation_Clause,
    A_Record_Representation_Clause,
    An_Address_Clause,
    Not_A_Representation_Clause);

function Kind (Clause : in Representation_Clause)
  return Representation_Clause_Kinds;

-- Returns the kind of the Representation_Clause
--
-- Returns Not_A_Representation_Clause for any inappropriate element
-- such as a Statement, Declaration, or Nil_Element.


function Associated_Type (Clause : in Representation_Clause)
  return Type_Definition;

-- Returns the definition of the type specified in the length clause,
-- enumeration representation clause or record representation clause.
--
-- Raises Inappropriate_Program_Element
-- if Kind(Clause) = Not_A_Representation_Clause


-------------------------------------------------------------------
-- LENGTH CLAUSES - 13.2

type Length_Clause_Attribute_Kinds is

   (A_Size_Attribute,                      -- LRM 13.2(a)
    A_Collection_Storage_Size_Attribute,   -- LRM 13.2(b)
    A_Task_Storage_Size_Attribute,         -- LRM 13.2(c)
    A_Small_Attribute,                     -- LRM 13.2(d)
    Not_A_Length_Clause_Attribute);
```

```
function Length_Clause_Attribute_Kind
   (A_Length_Clause : in Representation_Clause)
  return Length_Clause_Attribute_Kinds;

-- Returns the kind of the Length_Clause_Attribute.
--
-- Returns Not_A_Length_Clause_Attribute for any inappropriate element
-- such as a Statement, Declaration, or Nil_Element.


function Associated_Size (A_Type_Definition : in Type_Definition)
  return Expression;

-- Returns the expression that describes the size associated with a type
-- definition if there is a representation clause associated with this type.
--
-- A Nil_Element is returned if no associated representation clause exists.
--
-- Raises Inappropriate_Program_Element
-- if ASIS_Type_Definitions.Kind(A_Type_Definition) = Not_A_Type_Definition


function Associated_Storage_Size (A_Type_Definition : in Type_Definition)
  return Expression;

-- Returns the expression that describes the storage_size associated
-- with a type definition.
--
-- A Nil_Element is returned if no associated representation clause exists.
--
-- Raises Inappropriate_Program_Element
-- if ASIS_Type_Definitions.Kind(A_Type_Definition) = Not_A_Type_Definition


-------------------------------------------------------------------
-- ENUMERATION REPRESENTATION CLAUSES - 13.3


function Associated_Enumeration_Literal_Representation
   (Enumeration_Literal : in Identifier_Definition)
  return Expression;

-- Returns the expression that describes the representation of
-- enumeration literal.
--
```

```
-- Raises Inappropriate_Program_Element
-- if ASIS_Names_And_Expression.Kind(Enumeration_Literal)
--   /= An_Enumeration_Literal


function Enumeration_Type_Simple_Name
   (Enumeration_Clause : in Representation_Clause)
  return Identifier_Reference;

-- Returns the identifier reference of the type simple name in
-- the enumeration representation clause.
--
-- Appropriate Representation_Clause Kinds:
--   An_Enumeration_Representation_Clause


function Representation_Aggregate
   (Enumeration_Clause : in Representation_Clause)
  return Expression;

-- Returns the aggregate of the representation clause.
--
-- Appropriate Representation_Clause Kinds:
--   An_Enumeration_Representation_Clause


------------------------------------------------------------------
-- RECORD REPRESENTATION CLAUSES - 13.4


function Associated_Record_Type_Representation
   (Record_Type : in Type_Definition)
  return Representation_Clause;

-- Returns the record representation clause associated with the
-- record type definition.
--
-- A Nil_Element is returned if there is no associated representation
-- clause.
--
-- Appropriate Type_Definition Kinds:
--   A_Record_Type_Definition


function Associated_Record_Component_Representation
```

```
   (Record_Component : in Declaration)
  return Representation_Clause;

-- Returns the record representation clause associated with the
-- record type that encloses the record component declaration.
--
-- A Nil_Element is returned if there is no associated representation
-- clause.
--
-- Appropriate Declaration Kinds:
--   A_Component_Declaration
--   A_Discriminant_Specification


function Record_Type_Simple_Name (Record_Clause : in Representation_Clause)
  return Identifier_Reference;

-- Returns the identifier reference of the type simple name in the
-- representation clause.
--
-- Appropriate Representation_Clause Kinds:
--   A_Record_Representation_Clause


function Alignment_Expression (Record_Clause : in Representation_Clause)
  return Expression;

-- Returns the alignment expression for the representation clause.
--
-- A Nil_Element is returned if an alignment expression is not present.
--
-- Appropriate Representation_Clause Kinds:
--   A_Record_Representation_Clause


-------------------------------------------------------------------
-- COMPONENT CLAUSES - 13.4


subtype Component_Clause is ASIS_Ada_Program.Element;

subtype Component_Clause_List is ASIS_Ada_Program.Element_List;

function Component_Clauses (Record_Clause : in Representation_Clause)
  return Component_Clause_List;
```

```
-- Returns a list of the component clauses of the record clause in order
-- of appearance.
--
-- A Nil list is returned if the representation clause has no component
-- clauses.
--
-- Appropriate Representation_Clause Kinds:
--   A_Record_Representation_Clause


function Component_Simple_Name
   (Record_Component_Clause : in Component_Clause)
  return Identifier_Reference;

-- Returns the identifier reference of the name of the component.
--
-- Appropriate Major_Element_Kind:
--   A_Component_Clause


function Component_Offset (Record_Component_Clause : in Component_Clause)
 return Expression;

-- Returns the offset expression for the component.
--
-- Appropriate Major_Element_Kind:
--   A_Component_Clause


subtype Range_Constraint is ASIS_Ada_Program.Element;

function Component_Range (Record_Component_Clause : in Component_Clause)
  return Range_Constraint;

-- Returns the range constraint for the component.
--
-- Appropriate Major_Element_Kind:
--   A_Component_Clause


------------------------------------------------------------------------
-- ADDRESS CLAUSES - 13.5
```

```
   function Associated_Address (Identifier : in Identifier_Definition)
     return Representation_Clause;

   -- Returns the address clause associated with the identifier.
   --
   -- A Nil_Element is returned if no associated address clause exists.
   --
   -- Raises Inappropriate_Program_Element if the element is a Nil_Element.


   function Entity_Simple_Name (Address_Clause : in Representation_Clause)
     return Identifier_Reference;

   -- Returns the identifier reference of the entity named in
   -- the address clause.
   --
   -- Appropriate Representation_Clause Kinds:
   --    An_Address_Clause


   function Addressed_Declaration (Address_Clause : in Representation_Clause)
     return Declaration;

   -- Returns the declaration of the entity whose address is specified.
   --
   -- Appropriate Representation_Clause Kinds:
   --    An_Address_Clause


   function Address_Expression (Address_Clause : in Representation_Clause)
     return Expression;

   -- Returns the simple expression for the address of the entity.
   --
   -- Appropriate Representation_Clause Kinds:
   --    An_Address_Clause


end ASIS_Representation_Clauses;
```

```
-------------------------------------------------------------------------
-- ASIS PRELIMINARY REVIEW VERSION 0.4
-- Sept 27, 1991
-------------------------------------------------------------------------


with ASIS_Ada_Program;
package ASIS_Text is


  -----------------------------------------------------------------------
  -- 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 may be a reserved word), a numeric literal, a character
  -- literal, a string literal, or a comment.
  --
  -- Each ASIS Element has a text image whose value is a series of characters
  -- that forms 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 the range of lines.
  --
  -----------------------------------------------------------------------

  subtype Element is ASIS_Ada_Program.Element;


  Maximum_Line_Length : ASIS_Ada_Program.ASIS_Integer
    renames ASIS_Ada_Program.Maximum_Line_Length;
  -- A constant value, Implementation defined.

  Maximum_Line_Number : ASIS_Ada_Program.ASIS_Integer
    renames ASIS_Ada_Program.Maximum_Line_Number;
  -- A constant value, Implementation defined.

  subtype Line_Number is ASIS_Ada_Program.Line_Number;
  -- The range of line numbers within the text of programs.
  -- The value 0 is used to indicate an invalid line number.
```

```
subtype Character_Position is ASIS_Ada_Program.Line_Number range
  0..Maximum_line_Length;
-- The range of character positions within text lines.
-- The value 0 is used to indicate an invalid character position.

subtype Line is ASIS_Ada_Program.Line;
-- A private type

subtype Line_List is ASIS_Ada_Program.Line_List;
-- A list of lines.

Nil_Line_List : Line_List  renames ASIS_Ada_Program.Nil_Line_List;

-------------------------------------------------------------------------
-- EXCEPTIONS

Inappropriate_Line_Number : exception;

-- Raised on attempt to retrieve a line whose number is outside the
-- range of valid text lines.


-------------------------------------------------------------------------
-- LINE NUMBERS


function First_Line_Number (Program_Element : in Element) return Line_Number;

-- Returns the first line number on which the text of the element resides.
-- Returns 0 if a line number is not available for a given element,
-- and for any inappropriate element.


function Last_Line_Number (Program_Element : in Element) return Line_Number;

-- Returns the last line number on which the text of the element resides.
-- Returns 0 if a line number is not available for a given element,
-- and for any inappropriate element.


-------------------------------------------------------------------------
-- TEXT SPANS


type Span     is record
```

```
  First_Line    : Line_Number        := Line_Number'First;
  First_Column  : Character_Position := Character_Position'First;
  Last_Line     : Line_Number        := Line_Number'First;
  Last_Column   : Character_Position := Character_Position'First;
end record;

-- A single text position is identified by a line number and
-- a column representing the character position within the line.
--
-- An implementation may choose to ignore character position values
-- in which case the function Is_Span_Column_Position_Supported returns
-- False;
--
-- The text of an element may span one or more lines.  The Text span
-- of an element identifies the lower and upper bound of a span
-- of text positions.
--
-- Spans and position give client tools the flexibility to access text
-- through this package or through the external text file if any.


function Nil_Span return Span;

-- Returns a span with a void or nil value (the default initialization).


function Element_Span (Program_Element : in Element) return Span;

-- Returns the span of the given element.
--
-- Returns a Nil_Span if not Is_Text_Available(Program_Element),
-- and for any inappropriate element.


function Compilation_Unit_Span (Program_Element : in Element) return Span;

-- Returns the span of the text comprising the enclosing compilation unit
-- of the given element.
--
-- Returns a Nil_Span if text is not available for the enclosing
-- compilation unit, and for any inappropriate element.


function Compilation_Span (Program_Element : in Element) return Span;
```

```
-- 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 text is not available for the enclosing
-- compilation unit, and for any inappropriate element.


-------------------------------------------------------------------
-- LINES


-- Lists can be indexed to get each individual line.
-- The bounds of the lists returned correspond to the line numbers of
-- the text.


function Length (Text : in Line) return Character_Position;

-- Returns the length of the line.


function Lines (Program_Element : in Element) return Line_List;

-- Returns a list of lines covering the span of the
-- given program element.
--
-- Returns a Nil list if not Is_Text_Available(Program_Element),
-- and for any inappropriate element.


function Lines
   (Program_Element : in Element;
    Text_Span : in Span)
  return Line_List;

-- Returns a list of lines covering the given span
-- from the compilation containing the given program element.
--
-- 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.
--
-- Returns a Nil list if not Is_Text_Available(Program_Element),
-- and for any inappropriate element.
```

```
--
-- Raises Inappropriate_Line_Number if the Span defines a Nil_Span or a line
-- whose number is outside the range of text lines that can be accessed
-- through the element.


function Lines
   (Program_Element : in Element;
    First_Line      : in Line_Number;
    Last_Line       : in Line_Number)
  return Line_List;

-- Returns a list of lines covering the given line numbers
-- from the compilation containing the given program 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 list if not Is_Text_Available(Program_Element),
-- and for any inappropriate element.
--
-- Raises Inappropriate_Line_Number if the Span defines a Nil_Span or a line
-- whose number is outside the range of text lines that can be accessed
-- through the element.


-------------------------------------------------------------------------
-- IMAGES

subtype ASIS_Character is ASIS_Ada_Program.ASIS_Character;

-- The component type of the ASIS_String type.


subtype ASIS_String is ASIS_Ada_Program.ASIS_String;

-- ASIS_String values form lexical elements in the text of programs.
-- The ASIS_String type allows implementations to include non-ASCII
-- characters in comments appearing in the text of programs.


Nil_String : ASIS_String renames ASIS_Ada_Program.Nil_String;

-- A constant value
```

```
function Image_Delimiter return ASIS_String;

-- Returns the ASIS_String used as a delimiter separating
-- individual lines of text within the ASIS_String Image of an element.


function Image (Program_Element : in Element) return ASIS_String;

-- Returns an ASIS_String image of the element.
-- The image of an element may span more than one line, in which
-- case the ASIS_Character returned by the function Image_Delimiter
-- separates the individual lines.
--
-- Returns a Nil_String if not Is_Text_Available(Program_Element),
-- and for any inappropriate element.
--
-- NOTE: The image of a large element may exceed the range of an
-- ASIS_String in which case the exception Failed will be raised.
-- Use the Lines operation to operation on the image of large elements.


function Image (Text : in Line) return ASIS_String;

-- Returns an ASIS_String image of the entire line.
-- The image of a single lexical element can be sliced from the
-- ASIS_String value using the first or last column character positions
-- from the span of an Element.


function Non_Comment_Image (Text : in Line) return ASIS_String;

-- Returns an ASIS_String image of the lexical elements of a line
-- up to but excluding any comment on that line.
-- other than a comment.
--
-- The value returned includes any preceding white space characters.


function Comment_Image (Text : in Line) return ASIS_String;

-- Returns an ASIS_string image of any comment on that line excluding
-- any lexical elements preceding the comment.
--
-- The value returned includes the two adjacent hyphens "--".
--
```

```
-- A Nil_String is returned if the line has no comment.
--
-- The bounds of the ASIS_String correspond to the range of
-- Character_Positions the comment occupies in the line.


-------------------------------------------------------------------
-- TEXT AVAILABILITY

function Is_Text_Available (Program_Element : in Element) return Boolean;

-- Returns True if the implementation can return a valid text image for
-- the given element.
--
-- An implementation may choose to make text available only for certain
-- kinds of elements.


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;


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;


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;
```

```
   ------------------------------------------------------------------------
   -- LINE DESTRUCTOR


   procedure Dissociate (A_Line : in out Line);

   -- A Line of length 0 is returned.
   --
   -- Postconditions:  (if no exception is raised)
   --   Depending on the ASIS implementation, storage for Line
   --    may be deallocated.
   --   Length(A_Line) = 0



   procedure Dissociate (A_Line_List : in out Line_List);

   -- Severs associations made to the Line_List.
   -- A Nil_Line_List value is returned.
   --
   -- Postconditions:  (if no exception is raised)
   --   Depending on the ASIS implementation, storage for each Line
   --    may be deallocated.


end ASIS_Text;
```