ASIS-for-GNAT Reference Manual

Ada Core Technologies, Inc.

ASIS-for-GNAT Reference Manual

ASIS-for-GNAT Reference Manual

About This Manual

This Manual contains useful information in writing tools and applications using ASIS 95 implementation for the GNAT Ada 95 compiler (ASIS-for-GNAT). It includes information on implementation dependent characteristics and the current implementation limitations of ASIS-for-GNAT.

ASIS has been designed to be a portable basis for many useful Ada code analysis tools. However, since ASIS is designed to be implemented for wide variety of Ada compilers, it also contains a number of implementation specific features to be used in interfacing with the underlying Ada implementation, as well as implementation permissions for particular queries

Note: Any ASIS application that makes use of implementation-dependent features may be non-portable. You should follow good programming practice and isolate and clearly document any sections of your program that make use of these features in a non-portable manner.

What This Manual Contains

This guide contains the following chapters:

- Chapter 1 [ASIS-for-GNAT and the ASIS Standard], page 5 describes the relations between ASIS-for-GNAT and the existing ASIS International Standard;
- Chapter 2 [ASIS Extensions], page 7 describes the ASIS extensions implemented in ASIS-for-GNAT;
- Chapter 3 [Implementation-Specific Features and Implementation Permissions], page 13 goes through the aspects of the ASIS definition which are implementation-specific and describes their implementation in ASIS-for-GNAT;
- Chapter 4 [Debugging Information], page 23 describes the debugging information which can be produced by ASIS-for-GNAT

What You Should Know Before Reading This Manual

This Reference Manual assumes, that you are familiar with Ada 95 language, as described in the International Standard ANSI/ISO/IEC-8652:1995, Jan 1995, and with ASIS 95, as described in the ASIS 95 International Standard ISO/IEC 15291:1999.

This Manual relies on some important facts described in the ASIS-for-GNAT User's Guide and uses the terminology introduced in the User's Guide.

Related Information

For more information, see the following documents:

- GNAT User's Guide;
- ASIS-for-GNAT User's Guide;
- Ada 95 Reference Manual, which contains all reference material for the Ada 95 programming language;

– ASIS 95 definition, which contains all reference material for the Ada Semantic Interface Specification;

1 ASIS-for-GNAT and the ASIS Standard

Now the ASIS definition for Ada 95 (ASIS 95) exists as ISO/IEC 15291:1999 International Standard. ASIS-for-GNAT supports the ASIS 95 interface as it is defined in the ASIS Standard. ASIS-for-GNAT also contains some ASIS extensions (see chapter Chapter 2 [ASIS Extensions], page 7), as allowed by the ASIS Standard, section 1.1.3.1.

Currently some queries in some situations may raise the Asis_Failed exception with Not_Implemented_Error error status. This means, that some part of the functionality of a given query is not implemented yet. If you encounter such a situation, report it as an ordinary ASIS-for-GNAT bug. Our general goal is to have the full implementation of ASIS 95 conforming to the ASIS Standard.

ASIS-for-GNAT contains the specifications of all the ASIS interface packages as defined in the ASIS Standard, except the optional Data Decomposition Annex, with no changes, except adding comments in the very beginning, adding with clauses, adding declarations in package private parts and some reformatting needed to follow the GNAT coding style. The only exception is moving the Is_Dispatching_Operation query from Asis.Expressions into Asis.Declarations.

2 ASIS Extensions

ASIS-for-GNAT provides some additional queries as ASIS extensions. All these queries are defined and documented in the hierarchy headed by package Asis.Extensions. They are referred as "ASIS extensions" or "ASIS extension queries" below.

All the ASIS extensions obey all the general ASIS rules: when using ASIS extensions, you have to follow the required sequiensing of calls, only ASIS-defined exceptions are allowed to propagate outside the ASIS exception queries. If the documentation of an ASIS exception query contains a list of appropriate element kinds, then this query can be applied only to Elements from this list and it raises ASIS_Inappropriate_Element with Value_Error status otherwise. If the documentation of an ASIS exception query contains a list of expected element kinds, then this query can be applied to an Element having any kind, but it returns meaningful result only for Elements from this list.

The current set of ASIS extensions originated from the ASIS implementation needs and from the development of some ASIS tools inside the ASIS-for-GNAT team. As it stands now, Asis.Extensions hierarchy is not supposed to be stable - some queries may be removed from it, some new extension queries may be added. We are very interested in ASIS application developers feedback to implement a compact, but useful set of extension queries in ASIS-for-GNAT.

Currently Asis.Extensions hierarchy contains its top Asis.Extensions package defining a set of various-purpose extensions, and its only child Asis.Extensions.Flat_Kinds, which defines a "flat" Element classification.

Asis.Extensions package contains the following extension types and queries (see the specification of Asis.Extensions in the file named `asis-extensions.ads` for full details):

1. Types for dynamic Elements and Compilation Units lists:
   ```
   type Element_List_Access is access Element_List;
   type Compilation_Unit_List_Access is
      access Compilation_Unit_List;
   ```
   an application may use dynamic allocation to keep the results of queries returning lists (instead of declaring local list variables in block statements)
2. Placeholders for Traverse_Element instantiations:
   ```
   type No_State is (Not_Used);
   --  Placeholder for the State_Information formal type

   procedure No_Op
     (Element : Asis.Element;
      Control : in out Traverse_Control;
      State   : in out No_State);
   --  Placeholder for the formal Post_Operation procedure
   ```
   In many simple cases of instantiating Traverse_Element, the only needed actual is the actual for Pre_Operation, so these placeholders for traversal state and post-operation may be used to get the legal instantiation;
3. Test functions:
   ```
   function Is_Primitive_Operation (Declaration : Asis.Element)
     return Boolean;
   ```

```
--  Checks if its argument is a declaration of a (user-defined)
--  primitive operation of some type (both explicit and implicit
--  declarations are expected).

function Acts_As_Spec (Declaration : Asis.Element)
  return Boolean;

--  Checks if there is no separate subprogram declaration for a
--  given body declaration or body stub.

function Is_Renaming_As_Body (Declaration : Asis.Element)
  return Boolean;
--  Checks if its argument is a renaming-as-body declaration.

function Is_Completed (Declaration : Asis.Element)
  return Boolean;
--  Checks if its argument (which is expected to be a
--  declaration requiring completion) has a completion in its
--  enclosed ASIS Context.

function Is_True_Expression (Expression : Asis.Expression)
  return Boolean;
--  Checks if Expression is an expression in Ada sense, that is
--  if it is an expression as defined in RM 4.4, and the type of
--  this expression can be represented in ASIS. For cases of
--  An_Expression Element for which Is_True_Expression is True,
--  the Corresponding_Expression_Type query should yield non-Nil
--  result

function Is_Static (Expression : Asis.Expression)
  return Boolean;
--  Checks if Expression is static in the GNAT sense, that is if
--  the compiler computes its value during the compilation time.
--  We believe that this notion of a static expression is close
--  to the definition of static expression in RM 95, but we can
--  not guarantee this.
```

4. Modified versions of the "primary" ASIS queries:

```
generic
   type State_Information is limited private;

   with procedure Pre_Operation
                  (Element : in     Asis.Element;
                   Control : in out Traverse_Control;
                   State   : in out State_Information) is <>;

   with procedure Post_Operation
                  (Element : in     Asis.Element;
                   Control : in out Traverse_Control;
```

```
                         State   : in out State_Information) is <>;

procedure Traverse_Unit
  (Unit    : in     Asis.Compilation_Unit;
   Control : in out Traverse_Control;
   State   : in out State_Information);
--  A generalized version of Asis.Iterator.Traverse_Element.
--  Traverses the whole structure of the argument Compilation
--  Unit

function Formal_Subprogram_Default
  (Declaration : in Asis.Generic_Formal_Parameter)
   return Asis.Expression;
--  This is a modified version of the
--  Asis.Declarations.Formal_Subprogram_Default query. It returns
--  Nil_Element instead of raising Asis_Inappropriate_Element if
--  Subprogram_Default_Kinds of its argument is not
--  A_Name_Default

function Primary_Owner
  (Declaration : Asis.Declaration)
   return        Asis.Declaration;
--  In the case that Declaration is Is_Primary_Operation for some
--  tagged type, this function returns the tagged type definition
--  for which it is a primary operation.

function Corresponding_Called_Function_Unwinded
  (Expression : in Asis.Expression)
   return Asis.Declaration;
--  This is the modification of
--  Asis.Expressions.Corresponding_Called_Function which unwinds
--  all the renamings in case if the function name in the
--  argument function call is defined by a renaming declaration.
--  This function returns the declaration of the called function
--  *entity*.

function Corresponding_Called_Entity_Unwinded
  (Statement : in Asis.Statement)
   return Asis.Declaration;
--  This is the modification of
--  Asis.Statements.Corresponding_Called_Entity which unwinds all
--  the renamings in case if the procedure or entry name in the
--  argument call is defined by a renaming declaration. This
--  function returns the declaration of the callable *entity*.
```

5. Extensions of ASIS functionality:

```
function Is_Obsolete (Right : Asis.Compilation_Unit)
  return Boolean;
--  Checks if a unit is obsolete.
```

```
type Source_File_Statuses is
  (No_File_Status, Absent, Older,  Newer, Up_To_Date);

function Source_File_Status
  (Right : Asis.Compilation_Unit)
   return  Source_File_Statuses;
--  Checks the status of the source file for the argument unit.

function Is_Main_Unit_In_Tree
  (Right : Asis.Compilation_Unit)
   return  Boolean;
--  Checks if a unit is a main unit from some compilation which
--  has created a tree within the set of tree files making up the
--  enclosing Context.

function Compilation_Dependencies
  (Main_Unit : Asis.Compilation_Unit)
   return      Asis.Compilation_Unit_List;
--  Provides the full list of units upon which Main_Unit depends
--  in the GNAT compilation system.

function Corresponding_First_Definition
  (Defining_Name : in Asis.Defining_Name)
   return Asis.Defining_Name;
--  In case if there are more then one defining occurrence of an
--  argument Defining_Name representing the same view of the same
-- entity, this function returns the first defining occurrence
--  which actually introduces the entity.

function Corresponding_Body_Parameter_Definition
  (Defining_Name : Asis.Defining_Name)
   return          Asis.Defining_Name;
--  When applying to a defining name which is a name of a formal
--  parameter of a subprogram, this function returns the defining
--  name of this parameter from a subprogram body.

function Element_Span_In_Template
  (Element : Asis.Element)
   return    Asis.Text.Span;
--  If Is_Part_Of_Instance is True for the argument Element, then
--  this function returns the span of the corresponding piece of
--  code in the generic template. Otherwise a Nil_Span is
--  returned. Nil_Span is also returned if Is_Part_Of_Implicit
--  Element is True for Element.

function Element_Image_In_Template
  (Element : Asis.Element)
   return    Program_Text;
--  If Is_Part_Of_Instancce is True for the argument Element,
```

```
--  then this function returns the image of the corresponding
--  piece of code in the generic template. Otherwise a null
--  string is returned. A null string is also returned if
--  Is_Part_Of_Implicit_ELement is true for Element
```

6. General_Purpose Extensions:

```
function Get_Last_Component (E : Asis.Element)
  return Asis.Element;
--  Returns the right-most direct component of its argument.

function Components (E : Asis.Element) return Asis.Element_List;
--  Returns the list of all the first-level components of its
--  argument.
```

Asis.Extensions.Flat_Kinds package contains the definition of the type Flat_Element_Kinds. This type defines a "flat" Element classification which is equivalent to the Element classification hierarchy defined in Asis package: each kind value of the original hierarchy having subordinate kinds is replaced by the corresponding subordinate range. Asis.Extensions.Flat_Kinds package also provides conversion functions from the "flat" classification into the original classification hierarchy.

3 Implementation-Specific Features and Implementation Permissions

There are three kinds of implementation dependencies in ASIS.

First, ASIS subprograms which define an interface between an ASIS implementation and the underlying Ada implementation have implementation-specific parameters. There are three queries of this kind - Asis.Implementation.Initialize, Asis.Implementation.Finalize and Asis.Ada_Enfironments.Associate, all of them have a string parameter named Parameters which has implementation-specific meaning. The meaning of Parameters string in ASIS-for-GNAT is discussed in section Section 3.1 [Interacting with the Underlying Ada Implementation], page 13.

Second, some ASIS areas are considered as making problems for ASIS implementations. For such areas, the Asis Working Draft contains explicit implementation permissions which allow an ASIS implementation not to implement some part of the ASIS functionality or to implement it in a restricted way. Each of these permissions usually affects more, then one particular ASIS query. The ASIS package Asis.Implementation.Permissions contains boolean queries which tell you what choices are made for these implementation permission in a given ASIS implementation. The solutions taken in ASIS-for-GNAT for these general implementation permissions are discussed in section Section 3.2 [Implementation Permissions], page 16.

Third, for some ASIS queries implementation permissions which are specific for a given query are given explicitly in the ASIS Working Draft. For some other queries, the result of a query may be implementation-specific because of the very nature of the query, even though it is not explicitly said so in the ASIS Standard. Such queries are discussed in section Section 3.3 [ASIS Queries Having Specific Implementation Permissions or Implementation-Specific Results], page 17.

3.1 Interacting with the Underlying Ada Implementation

3.1.1 Format of the Parameters string

A Parameters string is a parameter of the three ASIS subprograms: Asis.Implementation. Initialize, Asis.Implementation.Finalize and Asis.Ada_Environments.Associate. The following requirements are common for all these subprograms:

- A Parameters string are considered as consisting on substrings bounded by the beginning of the string, the end of the string, space character, LF or CR and containing no space character, LF or CR inside. These substrings are called parameters (with low-case 'p') below;
- any number of space character, CR and LF are allowed before the first parameter, after the last parameter and between any two parameters;
- each of the queries Asis.Implementation.Initialize, Asis.Implementation.Finalize and Asis.Ada_Enfironments.Associate has its own requirements imposed on its parameters. For all of these queries, if some parameter does not match the rules for a given query, either an error message is generated or Asis_Failed with the Parameter_Error status is raised. In this manual, the situations when Asis_Failed is raised when processing the Parameters string are documented specially.

3.1.2 Parameters of Asis.Implementation.Initialize Procedure

The allowed parameters for Asis.Implementation.Initialize are:

`-d<debug_flag>`
: sets the ASIS-for-GNAT debug flag <debug_flag> ON

`-dall`
: sets all the ASIS-for-GNAT debug flags ON.

`-w<warning_mode>`
: sets the ASIS warning mode (This is under development at the moment).

Now <debug_flag> may be any lower case letter from 'a' to 'z' or any digit from 0 to 9 (not all of the corresponding options are implemented now).

ASIS debug flags are documented in the file `a_debug.adb`. (See also section Section 4.2 [ASIS Debug Flags], page 24 for some advices)

<warning_mode> could be either 's', which means "suppress all the warning messages" or 'e', which means "treat every warning as an error", and in this case every warning are converted in Asis_Failed raising with warning messages as ASIS Diagnosis strings.

If more then one parameter controlling the warning mode is set in the diagnosis string, each of them changes the warning mode in the place where it is given.

See also ASIS-for-GNAT Users' Guide for more information about warnings.

3.1.3 Parameters of Asis.Implementation.Finalize Procedure

No parameter is allowed for Asis.Implementation.Finalize in the current ASIS-for-GNAT version.

When called, Asis.Implementation.Finalize resets all the general ASIS-for-GNAT parameters to the default values (that is, all the debug flags are set off, and the warning mode is set to the default warning mode).

3.1.4 Parameters of Asis.Ada_Environments.Associate Procedure

The following parameters are allowed for this query:

`-C1`
: a Context is made up by a single tree file, this tree file name should be given explicitly in the Parameters string;

`-CN`
: a Context is made up by a set of tree files, the names of the tree files making up the Context should be given explicitly in the Parameters string;

`-CA`
: a Context is made up by all the tree files in the tree search path;

`-FS`
: all the trees considered as making up a given Context are created on the fly, whether or not the corresponding tree file already exists; once created, a tree file may then be reused till the Context remains open;

`-FT`
: only pre-created trees are used, no tree file can be created by ASIS;

`-FM`
: mixed approach: if a needed tree does not exist, the attempt to create it on the fly is made;

`-SA` source files for all the Compilation Units belonging to the Context (except the predefined Standard package) are considered in the consistency check when opening the Context;

`-SE` only existing source files for all the Compilation Units belonging to the Context are considered in the consistency check when opening the Context

`-SN` no source file from the underlying file system is taken into account when checking the consistency of the set of tree files making up a Context;

`-I<dir>` defines the directory to look for a source file;

`-T<dir>` defines the directory to look for a tree file;

`<file_name>`
defines the name of a tree file;

For -I and -T parameters, <dir> should denote an existing directory in the underlying file system, notations as "." and ".." are allowed, as well as relative or absolute directory names. If <dir> does not denote an existing directory, Asis_Failed with Parameter_Error status is raised.

A tree file name given by a <file_name> parameter may or may not contain directory information.

The search path associated with an ASIS Context consists of the directories listed as parameters for the Asis.Ada_Environments.Associate query, in the same order as they are included in the actual Parameters string. The ASIS source search path consists only of the directories following '-I', and the ASIS tree search path consists only of the directories following '-T'. If no source (tree) directory presents in the value of the Parameters string, then the ASIS source (tree) search path consists of the current directory only, otherwise the current directory is included in the ASIS search path if amd only if it is set explicitly as -I. or -T. respectively.

If an ASIS Context is associated with -FS or -FM option, the Context source search path to locate sources of the units to create a tree file for, and to locate other source files needed when called GNAT to create tree files. For example, if we have:

```
Asis.Ada_Environments.Associate
  (My_Cont,
  "My_Cont_Name",
  "-CA -FS -I./dir -I.");
```

then, when processing a call:

```
My_Unit := Asis.Compilation_Units.Library_Unit_Declaration
  ("Foo", My_Cont);
```

ASIS first try to locate the source file foo.ads in ./dir, and if this attempt fails, it tries to locate it in the current directory. If the source file is found (let it be in the current directory), ASIS calls GNAT to create the tree file as

```
gcc -c -gnatc -gnatt -I./dir -I. -I- foo.ads
```

If an ASIS Context is associated with -CA option, then, when this Context is opened, ASIS processes all the tree files located in the tree search path associated with the Context.

The following restrictions are imposed on allowed combinations of parameters in the actual Parameters string of the Asis.Ada_Environments.Associate query:

- -C1 and -CN require FT
- -C1 requires exactly one tree file to be set
- -CN requires at least one tree file to be set
- -CA does not allow a tree file name
- -FS and -FM require -SA

In case if an incompatible combination is set, Asis_Failed with Parameter_Error status is raised

In case if the actual for the Parameters string in a call to the query `Associate` is an empty string, the default parameters are: "-CA -FT -SA"

Note, that the definition of the parameters might change later on.

For the Name parameter of the Asis.Ada_Environments.Associate query any string can be passed as an actual. In the current ASIS-for-GNAT version no verification is performed for an actual and no semantic is associated with the Name parameter of the Asis.Ada_Environments.Associate query.

3.2 Implementation Permissions

3.2.1 Asis.Implementation.Permissions Queries

The Boolean queries defined in the `Asis.Implementation.Permissions` package return the following results under ASIS-for-GNAT:

```
Is_Formal_Parameter_Named_Notation_Supported   True
Default_In_Mode_Supported                      True
Generic_Actual_Part_Normalized                 False
Record_Component_Associations_Normalized       False
Is_Prefix_Call_Supported                       True
Function_Call_Parameters_Normalized            False
Call_Statement_Parameters_Normalized           False
Discriminant_Associations_Normalized           False
Is_Line_Number_Supported                       True
Is_Span_Column_Position_Supported              True
Is_Commentary_Supported                        True
Attributes_Are_Supported                       False
Implicit_Components_Supported                  False +
Object_Declarations_Normalized                 False
Predefined_Operations_Supported                False +
Inherited_Declarations_Supported               False +
Inherited_Subprograms_Supported                False +
Generic_Macro_Expansion_Supported              True
```

+ - See section Section 3.2.2 [Processing Implicit Elements], page 16

3.2.2 Processing Implicit Elements

ASIS Elements represent both explicit and implicit components of Ada programs. There are queries in ASIS which can return implicit Elements (that is, Elements representing implicit Ada constructs) or which can have implicit Elements as parameters.

The ASIS Standard gives the permission to ASIS implementation not to support implicit Elements at all or to support them only partially. If an implementation does not support implicit Element representing a particular kind of Ada implicit constructs, this means, that corresponding ASIS queries return Nil_Element in cases when, according to RM-defined Ada syntax and semantic, they should return implicit Elements representing these implicit constructs.

Now implicit Elements are partially supported by ASIS-for-GNAT. In case of implicitly declared user-defined inherited subprograms, the queries `Asis.Expressions.Corresponding_Name_Definition`, Asis.Expressions.Corresponding_Called_Function and `Asis.Statements.Corresponding_Called_Entity` return non-nil result representing the corresponding entity. In case of implicitly declared predefined operations, `Nil_Element` is returned.

Our final goal is to have all the implicit Ada constructs fully implemented in ASIS-for-GNAT.

3.2.3 Processing Several Contexts at a Time

ASIS Standard says that the number of ASIS Context which can be associated and opened at a time, as well as the number of ASIS Compilation Units which can be processed at a time are implementation-specific. ASIS-for-GNAT does not impose any restrictions on the number of ASIS Context opened at a time in an ASIS application, as well as on the number of ASIS Compilation Units which can be obtained from all the opened Contexts, as long as the application does not go beyond the general system resource limitations.

Note, however, for a Context associated with -FS or -FM option, all the trees create don the fly during obtaining Compilation Units from this Context are placed in the current directory. If the current directory contains also some tree files belonging to another Context, they may be corrupted. To process more then one Context in a safe way, an application should have ot most one Context associated with -FS or -FM option. Moreover, if among Contexts processed at the same time there is a Context which can create trees on the fly, all the other Context should not use tree files located in the current directory.

3.2.4 Implementation-Defined Types and Values

All the implementation-defined types, subtypes and values depend on the subtype Implementation_Defined_Integer_Type and on the Implementation_Defined_Integer_Constant defined in package Asis. ASIS-for-GNAT does not change the definition given in the specification of package Asis as it stands in the ASIS Standard:

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

All the ASIS (sub)types used as list indexes for the ASIS array types have this value of Implementation_Defined_Integer_Constant as an upper bound.

3.3 ASIS Queries Having Specific Implementation Permissions or Implementation-Specific Results

This section contains implementation-specific documentation items for the queries having implementation permissions (given under --|IP sentinel in the ASIS definition) or for the queries whose behavior may be implementation-specific for whatever reason. Such queries are ordered according to the presentation in the ASIS definition. We have preserved the clause and subclause numbers from the ASIS definition, putting these numbers between quotes to avoid confusion with section numbers in this manual.

The results returned by the ASIS Debug_Image queries are discussed in section Section 4.1 [Interpreting Debug Images], page 23.

```
'8'  package Asis.Ada_Environments

'8.1'  function Default_Name
       - null string is returned;

'8.2'  function Default_Parameters
       - null string is returned;

'8.4'  procedure Open
       - what happens during the Context opening is:

         - for a Context associated with -CA option:
           - if -FS is also set, nothing is doing;
           - if -FT or -FM is set, all the tree files (that is,
             files having .adt suffix) in the tree search path
             associated with the Context are processed. For every
             tree file an attempt to read it in is made, and then
             it is checked that this file was created with
             -gnatc option. Tree files which cannot be read in or
             which were not created with -gnatc option are ignored.
             For all the other trees ASIS collects some black-box
             information about Compilation Units represented by
             this tree file, and performs a consistency check for
             every unit it encounters in the tree (see ASIS-for-GNAT
             Users' Guide for discussion of the consistency
             problem). If any consistency check fails, Asis_Failed
             is raises and the Context remains closed.

         - for a Context associated with -C1 or -CN option: ASIS
           process all the tree files associated with the Context,
           collecting black-box information and doing consistency
           check for all the encountered Compilation Units. If,
           because of any reason, a tree file cannot be
           successfully read in, for a Context associated with -C1
           option Asis_Failed is raised and the Context remains
           closed; and for a Context associated with -CN option,
           ASIS warning is generated and Context opening goes
           further. If any consistency check fails, Asis_Failed is
           raised and the Context remains closed.

'10' package Asis.Compilation_Units
```

```
'10.3'  function Unit_Origin
        - A_Predefined_Unit origin is returned for compilation units
          listed in RM95, Annex A (2), and only for these units;

        - An_Implementation_Unit origin is returned for compilation
          units which are the components of the GNAT Run-Time
          Library, but which are not mentioned in RM95, Annex A (2);

        - An_Application_Unit origin is returned for all the other
          compilation units;


'10.6'  function Library_Unit_Declaration
'10.7'  function Compilation_Unit_Body
        - when processing a Context associated with -FS or -FM
          option, for both of these queries, if ASIS cannot find a
          needed unit in the tree files which have been already
          processed, it tries to create the needed tree by locating
          the source of the unit and compiling it on the fly. If
          this attempt fails by any reason, Nil_Compilation_Unit
          is returned;

'10.13' function Corresponding_Declaration:
        - ASIS-for-GNAT does not make use of ASIS Compilation Units
          of An_Unknown_Unit kind;
        - if an argument is of A_Public_Declaration_And_Body class,
          Nil_Compilation_Unit is returned;

'10.14' function Corresponding_Body
        - ASIS-for-GNAT does not make use of ASIS Compilation Units
          of An_Unknown_Unit kind;

'10.22' function Can_Be_Main_Program
        - for GNAT, any parameterless library procedure and any
          parameterless library function returning the result of
          (any) integer type is classified by this query as a
          (possible) main subprogram for partition;

        - If for such a library subprogram both spec and body exist
          as ASIS Compilation Units retrievable form a given ASIS
          Context, both of them are considered as
          Can_Be_Main_Program

'10.24' function Text_Name
        - this function returns the name of the source file
          containing the source of Compilation_Unit. This name may
          or may not contain a prefix indicating the directory in
          the underlying file system, the directory indication may
          be given in absolute or the relative form, depending on
```

```
                the command line options which were used for the call
                to GNAT that created the corresponding tree file;
              - this function does not check the existence of the
                corresponding source file in the underlying file system,
                it just reflects the situation which was in effect when
                the corresponding tree file was created. Thus, if you
                delete or move the corresponding source file after
                creating the tree, the full file name returned by this
                function will be incorrect;
              - in case of inconsistency (that is, when several versions
                of the unit's source were used for creating the tree files
                making up the given ASIS Context), the name of the latest
                version of the source is returned.

'10.25' function Text_Form
              - In the GNAT compilation model all source files are
                ordinary text files in the underlying file system.
                Therefore this function always returns a Nil_Asis_String
                to indicate that Text_IO.Open uses the default options for
                manipulating Ada sources.
'10.29' function Has_Attribute
              - Returns False. ASIS-for-GNAT does not provide any
                additional attributes for Compilation Units.

'10.30' function Attribute_Value_Delimiter
              - Returns (wide) string of the length one and containing LF
                (wide) character.

'10.31' function Attribute_Values
              - a null string is returned


'11' package Asis.Compilation_Units.Times

'11.2'  function Time_Of_Last_Update
              - This function returns the time stamp of the corresponding
                source file. The corresponding source file is the source
                file whose name is returned by Asis.Compilation_Units.
                Text_Name. That is to say, this function returns the time
                of the latest change to the sources for the unit.

'13' package Asis.Elements

'13.3'  function Context_Clause_Elements
              - this function returns exactly those clauses and pragmas
                which are in the source for the unit.
              - returns Nil_Element_List for Nonexistent or Unknown unit.
              - returns Nil_Element_List for the predefined package
                Standard. For all other predefined Ada compilation units,
```

```
          returns their context clauses as they appear in the
          sources held in the GNAT Run-Time Library.

'13.5'  function Compilation_Pragmas
        - this function returns exactly those pragmas which are in
          the source for the unit, both in the context clause and
          after the end of the text of the compilation unit proper.
        - returns Nil_Element_List for Nonexistent or Unknown unit.

        - returns Nil_Element_List for the predefined package
          Standard. For all the other predefined Ada compilation
          units, returns the pragmas found in the corresponding
          sources in the GNAT Run-Time Library.

'13.36' function Enclosing_Element
        - ASIS-for-GNAT does not actually require the
          Element_Context parameter. The Enclosing_Element function
          with two parameters just calls the Enclosing_Element
          function with one parameter for its Element parameter.


'15' package Asis.Declarations

'15.24' function Body_Block_Statement
        - If the body passed as the actual has no declarative items
          on its own, Asis.Statements.Is_Declare_Block returns FALSE
          for the result of this function

'17' package Asis.Expressions

'17.6'  function Corresponding_Name_Definition
        - if an argument Reference name is declared implicitly, then
          if it denotes a user-defined inherited subprogram, the
          corresponding non-nil defining name Element is returned.
          Otherwise the function returns Nil_Element.

'17.8'  function Corresponding_Name_Declaration
        - if an argument Reference name is declared implicitly, then
          if it denotes a user-defined inherited subprogram, the
          corresponding non-nil declaration Element is returned.
          Otherwise the function returns Nil_Element.

'17.29' function Corresponding_Called_Function
        - For implicitly declared functions, if the actual is an
          expression that represents a call to an predefined
          operation, Nil_Element is returned. If it is a call to an
          implicitly declared user-defined inherited operation, the
          corresponding (implicit) function declaration is returned;
```

```
'18' package Asis.Statements

'18.14' function Is_Declare_Block
        - If the argument represents the dummy block statement
          created by Asis.Declarations.Body_Block_Statement
          function, the result will be True if and only if the
          corresponding body has declarative items on its own.


'20' package Asis.Text

'20.1'  type Line
        - Lines in ASIS-for-GNAT do not contain any character which
          signify end of line according to RM95, 2.2(2)

'20.22' function Delimiter_Image
        - Returns (wide) string of the length one and containing LF
         (wide) character.
```

3.4 Restrictions Specific to -FS and -FM Context options

The following queries from Asis.Compilation_Units are not implemented for a Context associated with -FS or -FM Context option:

```
Library_Unit_Declarations
Compilation_Unit_Bodies
Compilation_Units
Corresponding_Children
Corresponding_Body (formally, this function is implemented, but
                    it may give wrong results)
Subunits
```

4 Debugging Information

There are two kinds of the debugging information available in ASIS-for-GNAT - debug images returned by the ASIS queries Debug_Image for Contexts, Compilation Units and Elements, and debug output generated by ASIS queries when the corresponding implementation debug flag is set ON during ASIS initialization (see Section 3.1.2 [Parameters of AsisImplementationInitialize Procedure], page 14).

4.1 Interpreting Debug Images

It is easy to interpret the debug images generated for the main ASIS abstractions, because most of the information being generated directly corresponds to ASIS notions. The following details of debug images are implementation-specific:

- Context

  `Context Id`
  : this is the internal Context Id used in the internal implementation data structures. This Id is assigned to a Context when it is associated for the first time, and it remains unchanged and unique unlit ASIS is finalized.

  `All tree files`
  : the number of the tree files making up the given Context

- Compilation_Unit

  `Unit Id`
  : this is the internal Compilation_Unit Id used in the internal implementation data structures. This Id remains unchanged and unique unlit the unit's enclosed Context is closed.

  `Is consistent`
  : TRUE if the same version of the unit's source was used for all the tree files making up the enclosed unit's context, and FALSE otherwise

- Element

  `Node, R_Node, Node_Field_1`
  : tree nodes on which the internal representation of a given Element is based. They are meaningful only in the tree file indicated in the Enclosing_Tree field of the debug image

  `Special Case`
  : implementation-specific indication of the cases when the element needs some special processing

  `"obtained from the tree"`
  : the Id and the name of the tree file from which the tree-specific fields of the internal representation of given Element were obtained

  `Rel_Sloc`
  : indicates the (relative) position of the source text of the Element, counting from the beginning of the source of its enclosing compilation unit. Makes sense for implicit Elements also.

4.2 ASIS Debug Flags

ASIS provides several internal debug flags which are described in a_debug.adb. Their setting generates (to stdout) useful internal debugging information. This information is not always user-oriented, but the following debug flags may be useful for the ASIS user:

`-dc` outputs the content of the internal data structures for a Context, when the Context is closed and dissociated. By analyzing this information, you may map other debug information onto unit and tree Ids.

`-di` turns off including the location of an Element into its Debug_Image. It may be useful if an ASIS program crashes because of some problem with ASIS structural queries (structural queries are used by Element's Debug_Image query to compute the source location of the argument).

`-do` when the Context is opened, lists the tree files being processed, and the ones selected to represent a given Context

`-dt` outputs a message every time when a tree file is read in. This information may be useful to analyze the "tree swapping profile of your application to cut down the number of tree swapping.