ASIS Fundamentals

- ASIS Overview
  - Background
  - Structure of asis.ss
  - Operations Packages
- Basic Components
- Writing ASIS Programs

What Is ASIS?

- Gives you programmatic access to semantic information
- Based on LRM (1815A)
- Is an industry standard
- Is compiler independent
  – although some features are implementation dependent
- Is “read only”

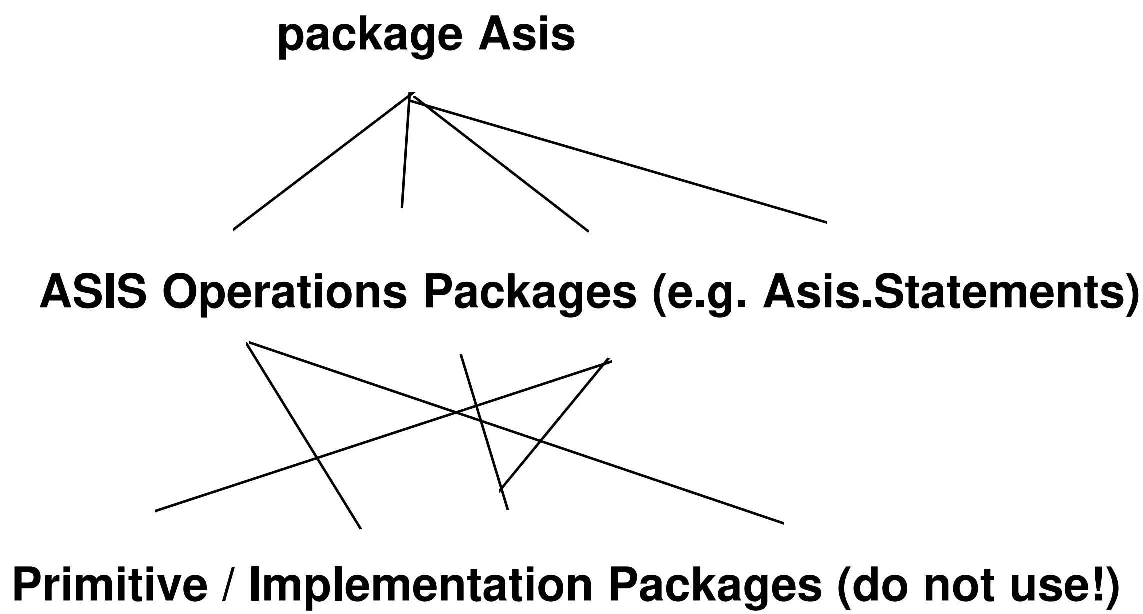
Uses of ASIS

- Lines-of-code counter
- Quality-metrics utility
- Type/Data analyzer
- Call-tree analyzer
- Exception tracker
- Comment extractor
- many more!

Advantages of ASIS

- Provides information much more precisely than string-based utilities could
- Uses terminology similar to the LRM, making it easier to learn
- Makes code analysis tools more portable

Structure of asis.ss



ASIS Operations Packages

- **Setup Packages: gets you to the elements**
  - **Environments, Libraries, Compilation_Units**
- **Query Packages: traverses to the desired elements**
  - **Declarations, Statements, Type_Definitions, Expressions, Representation_Clauses, Exceptions, Compilation_Units**
- **Low-level Packages: retrieves detailed information about an element**
  - **Asis_Elements, Lines, Numerics, Text, Strings, Ids**

Instructor: browse through the subsystem with the class

ASIS Fundamentals

- **ASIS Overview**
- **Basic Components**
  - Libraries
  - Compilation Units
  - Elements
  - Lists
- **Writing ASIS Programs**

ASIS Libraries

- Determine the *closure* of units to be queried
- Must be opened before doing any queries
- Are of type Asis.Library
- In Apex, are named by views or configurations
  - Example: Asis.LIbraries.Associate (The_Lib, “/project/my_sub.ss/my_view.wrk”);

Compilation Units

- **Match the LRM definition in Section 10.1**
- **Are either library units (specs) or secondary units (bodies)**
- **Are of type Asis.Compilation_Unit**
- **In Apex, are specified by uppercase simple name**
  - **Example: Asis.Compilation_Units.Library_Unit ("SYSTEM");**

Basic Steps for All ASIS Applications

1. Asis.Environment.Initialize;
2. Asis.Libraries.Associate ();
3. Asis.Libraries.Open ();
4. obtain compilation unit(s)
5. traverse and query as needed
6. Asis.Libraries.Close ();
7. Asis.Libraries.Dissociate ();
8. Asis.Environment.Finalize;

Exercise #1: display_unit

1. **Create a subsystem (asis_trng.ss) and a view.**
2. **Create a procedure body named display_unit.**
3. **Enter the following program:**

```
with Asis;
with Text_Io;
procedure Display_Unit is
  View_Name : constant String := “<training_loc/asis_trng.ss/rev1.wrk”;
  Unit_Name : constant String := “DISPLAY_UNIT”;
  The_Library : Asis.LIbrary;
begin
  Asis.Environment.Initialize;
  Asis.Libraries.Associate (Library => The_Library,
                Name => Asis.Strings.To_Asis_String (View_Name));
  Asis.Libraries.Open (The_Library);
  declare
    The_Unit : Asis.Compilation_Unit;
  begin
    The_Unit := Asis.Compilation_Units.Library_Unit (Name => Unit_Name,
                        Library => The_Library);
```

Exercise #1: display_unit (contd.)

```
     Text_Io.Put_Line ("Pathmame is " & Asis.Strings.To_Standard_String
          (Asis.Compilation_Units.Unique_Name (The_Unit)));
   end;

   Asis.Libraries.Close (The_Library);
   Asis.Libraries.Dissociate (The_Library);
   Asis.Environment.Finalize;
end Display_Unit;
```

4. Code, link, and execute. Notice the size of the executable.
5. Modify display_unit to display the kind of the compilation unit after the name.

ASIS Elements

- Represent individual semantic entities
- Match concepts described in the Ada LRM
- Often contain other elements (children)
- Are of type Asis.Element
- Are subtyped to provide “weak typing”
- Always have an Element_Kind
  - which determines what queries are available

Queries on Elements

- **Obtain child elements**
  - Example: Asis.Declarations.Subprogram_Formal_Parameters
- **Obtain related elements**
  - Example: Asis.Expressions.Name_Definition
- **Obtain detail on a specific element**
  - Example: Asis.Text.Line_Number
- **Obtain parent element or parent compilation unit**

Instructor: browse through query packages again, in more detail

How ASIS Queries Map to the LRM

From LRM 3.6:

array_type_definition ::= unconstrained_array_definition | constrained_array_definition

constrained_array_definition ::= **array** index_constraint **of** component_subtype_indication

index_constraint ::= **(** discrete_range {, discrete_range } **)**

From Asis.Type_Definitions:

```
function Is_Constrained_Array (...) return Boolean;

function Index_Constraint (...) return Asis.Constraint;

function Discrete_Ranges (...) return Asis.Discrete_Range_List;

function Component_Subtype_Indication (...) return
                                              Asis.Subtype_Indication;
```

Typical Element Query

```
function Index_Constraint (Type_Definition : Asis.Type_Definition)
                           return Asis.Constraint;
---------------------------------------------
-- Returns the index constraint for a constrained array...
--
-- Use the Discrete_Ranges query to decompose the index constraint
--
-- Appropriate Asis.Elements.Element_Kinds:
--    A_Type_Definition
--
-- Appropriate Asis.Type_Definitions.Type_Definition_Kinds:
--    An_Array_Type_Definition
--    A_Generic_Array_Type_Definition
--
-- Returns Asis.Element.Element_Kinds:
--    A_Constraint
--
-- Returns Asis.Type_Definitions.Constraint_Kinds:
--    An_Index_Constraint
```

Typical Element Query (contd.)

- **Comments always tell you what kinds of elements will work for each query**
  - If it uses “appropriate”, failure to comply will raise exception
  - If it uses “expected”, failure will return Nil_Element or False condition
- **Comments also tell you what kinds of elements are returned**
  - This will give you a clue for what queries will work on the result
- **Remember, weak typing implies that programming errors will be detected at run-time, not compile-time!**

ASIS Lists

- **Implemented using unconstrained arrays**
- **Are either an Element_List or Compilation_Unit_List**
  - **other kinds are subtypes, yielding "weak typing"**
- **Typical usage:**

```
declare
  My_List : constant Asis.Element_List := Asis....;
begin
  for I in My_List'range loop
    do_something_with (My_List(I));
  end loop;
end;
```

Example: Compilation Units in a View

```
function Compilation_Units_In_View
                (Library : in Asis.Library; View_Name : in String)
                 return Asis.Compilation_Unit_List is
  All_Units : constant Asis.Compilation_Unit_List :=
          Asis.Compilation_Units.Compilation_Units (Library);
  Units_In_View : Asis.Compilation_Unit_List (All_Units'Range);
  Num_Units_In_View : Natural := 0;
begin
  for I in All_Units'Range loop
    declare
      Unit_Name : constant String := Asis.Strings.To_Standard_String
            (Asis.Compilation_Units.Unique_Name (All_Units (I)));
    begin
      if String_Utilities.Locate (View_Name, Unique_Name) =
              Unique_Name'First then
        Num_Units_In_View := Num_Units_In_View + 1;
        Units_In_View (Asis.Numerics.List_Index (Num_Units_In_View) :=
              All_Units (I);
      end if;
    end;
  end loop;
  return Units_In_View (1 .. Asis.Numerics.List_Index (Num_Units_In_View);
end Compilation_Units_In_View;
```

Exercise #2: display_element

1. Create a new procedure body called display_element.
2. Copy the context clauses, declarations, and body of display_unit into display_element.
3. Insert a block that obtains the unit declaration from the compilation unit, then displays the declaration kind and first line number of the unit declaration.
4. If you finish early: add the code that displays the name of the unit declaration as well.

(Solutions to all exercises are installed on your machine.)

Exercise #3: display_withs

1. Create a new procedure body called display_withs.
2. Copy the context clauses, declarations, and body from display_unit into display_withs.
3. Insert a block that obtains the context clauses from the compilation unit; then for each clause obtains the referenced units.
4. Display the name of each unit withed by the compilation unit.

RATIONAL SOFTWARE CORPORATION

ASIS Fundamentals

- ASIS Overview
- Basic Components
- Writing ASIS Programs
  - General Traversal
  - Specific Traversal
  - Error Handling
  - Hints and Tips

General Traversal: Walking a Tree

- General Traversal is used when you are searching for elements that may appear anywhere within a compilation unit
- The algorithm is similar to walking an n-ary tree, which is how the semantic information is stored
- Traversal requires instantiation of this generic:

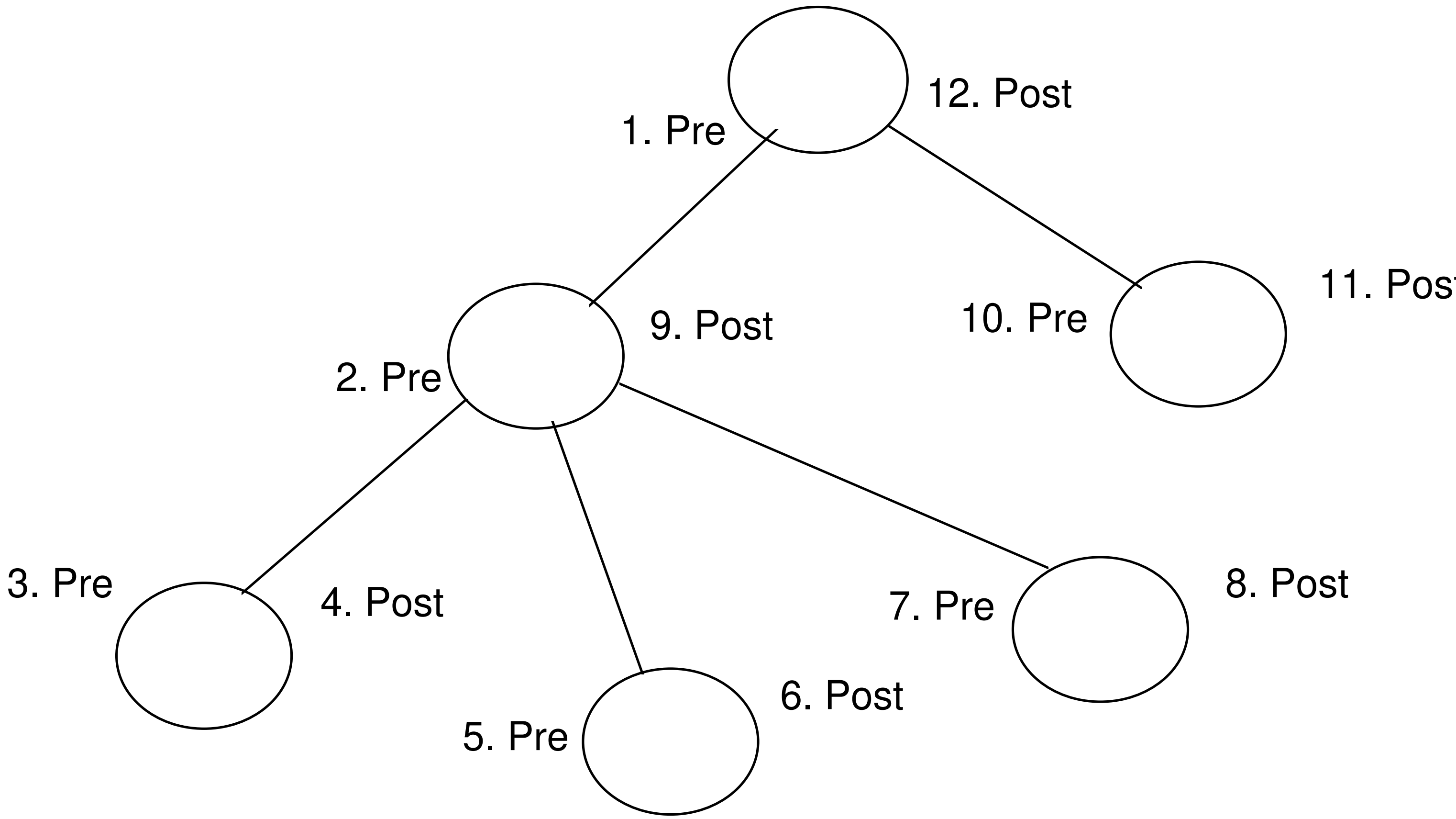
```
generic
  type State_Information is limited private;
  with procedure Pre_Operation (Element : in  Asis.Element;
                                Control : in out Traverse_Control;
                                State : in out State_Information) is <>;
  with procedure Post_Operation (Element : in Asis.Element;
                                 Control : in out Traverse_Control;
                                 State : in out State_Information) is <>;
procedure Traverse_Element (Element : in Asis.Element;
                            Control : in out Traverse_Control;
                            State  : in out State_Information);
```

The Traverse_Element generic

- State_Information can be any type
- Pre_Operation is applied on the way down
- Post_Operation is applied on the way up
- Traverse_Control determines when to stop:

```
type Traverse_Control is
   (Continue,
    Abandon_Children,
    Abandon_Siblings,
    Terminate_Immediately);
```

Depth-First Traversal



General Traveral: Typical Use

1. Obtain an element, usually the unit declaration.
2. Call an instantiation of Asis.Elements.Traversal.
3. In the pre_operation, case on the element kind to determine the elements of interest
4. Apply further queries, process as needed

Example General Traversal: Pre-Op

```
procedure Pre_Operation (Element : in Asis.Element;
              Control : in out Asis.Elements.Traverse_Control;
              State : in out State_Info) is
begin
  case Asis.Elements.Element_Kind (Element) is
    when Asis.Elements.A_Declaration =>
      case Asis.Declarations.Kind (Element) is
        when Asis.Declarations.A_Parameter_Specification =>
          case Asis.Declarations.Parameter_Mode_Kind (Element) is
            when Asis.Declarations.A_Default_In_Mode =>
              Text_Io.Put_Line ("Rule Violation (default in mode) " &
                   Asis.Expressions.Name (Asis.Declarations.Names
                   (Element) (1)) & " in " & Asis.Compilation_Units.Name
                   (Asis.Compilation_Units.Enclosing_Compilation_Unit
                       (Element)));
            when others => null;
          end case;
        when others => null;
      end case;
    when others => null;
  end case;
end Pre_Operation;
```

Example (contd.): Instantiation

```
type State_Info is (Not_Used);

procedure Pre_Operation (Element : in Asis.Element;
              Control : in out Asis.Elements.Traverse_Control;
              State : in out State_Info) is
begin
  [Insert code from previous slide here]
end;

procedure Post_Operation (Element : in Asis.Element;
              Control : in out Asis.Elements.Traverse_Control;
              State : in out State_Info) is
begin
  null;
end;

procedure Find_Violations is new Asis.Elements.Traverse_Element
                (State_Information => State_Info,
                 Pre_Operation => Pre_Operation,
                 Post_Operation => Post_Operation);
```

Example (contd.): Main program

```
with Asis;
with Text_Io;
procedure Display_Code_Violations is
  View_Name : constant String := “[view name here]”;
  The_Library : Asis.Library;

  [instantiation here]

begin
  [initialize the library here]
  declare
    Dont_Care : State_Info := Not_Used;
    Control : Asis.Elements.Traverse_Control := Asis.Elements.Continue;
    All_Units : constant Asis.Compilation_Unit_List :=
        Asis.Compilation_Units.Compilation_Units (The_Library);
  begin
    for I in All_Units’Range loop
      Find_Violations (Element => Asis.Compilation_Units.Unit_Declaration
                                    (All_Units (I)),
                       Control => Control, State => Dont_Care);
    end loop;
  end;
  [finalize the library here]
end Display_Code_Violations;
```

Specific Traversal

- Goes directly to the elements, rather than visiting every one
- Requires much deeper knowledge of the language and the LRM
- Faster than general traversal
- Often used in conjunction with a limited general traversal

Example Specific Traversal

```
function Procedure_Handles_Storage_Error
        (Body_Decl : Asis.Declaration) return Boolean is
  Exception_Handlers : constant Asis.Exception_Handler_List :=
          Asis.Statements.Block_Exception_Handlers
              (Asis.Declarations.Subprogram_Body_Block (Body_Decl));
begin
  for I in Exception_Handlers'Range loop
    if not Asis.Statements.Is_Others_Handler (Exception_Handlers (I)) then
      declare
        Exception_Names : constant Asis.Choice_List :=
              Asis.Statements.Exception_Choices (Exception_Handlers (I));
      begin
        for J in Exception_Names'Range loop
          if Asis.Expressions.Name (Asis.Type_Definitions.Choice_Name
                         (Exception_Names (I))) = "STORAGE_ERROR" then
            return True;
          end if;
        end loop;
      end;
    end if;
  end loop;
  return False;
end Procedure_Handles_Storage_Error;
```

Example: Main program

```
with Asis;
with Text_Io;
procedure Check_Storage_Error is
  View_Name : constant String :=
        "/export/projects/utilities/asis_utilities.ss/training.wrk";
  The_Library : Asis.Library;
  [function Procedure_Handles_Storage_Error here]
begin
  [initialize library here]
  declare
    All_Units : constant Asis.Compilation_Unit_List :=
          Asis.Compilation_Units.Compilation_Units (The_Library);
    Unit_Decl : Asis.Declaration;
  begin
    for I in All_Units'Range loop
      Unit_Decl := Asis.Compilation_Units.Unit_Declaration(All_Units(I));
      case Asis.Declarations.Kind (Unit_Decl) is
        when Asis.Declarations.A_Procedure_Body_Declaration =>
          if not Procedure_Handles_Storage_Error (Unit_Decl) then
            Text_Io.Put_Line (Asis.Compilation_Units.Name
                  (All_Units (I)) & " does not handle storage_error");
          end if;
        when others => null;
   [etc...]
```

Exercise #4: display_all_types

1. Create a new procedure body called display_all_types.
2. Paste in the reusable parts from display_unit.
3. Paste in the declaration of pre_operation from Asis.Elements, and convert it to a procedure body.
4. Implement a pre_operation that outputs the name and compilation unit of every full type declaration and subtype declaration it finds.
5. Instantiate the generic, and complete the program so it displays all the types declared in every unit in the closure of your view.

Exercise #5: display_visible_types

1. Create a new procedure body called display_visible_types.
2. Paste in the reusable parts from display_all_types (but not the pre_operation or instantiation).
3. Implement a procedure that takes a parameter of type Asis.Declaration_List, and loops through that list looking for full type and subtype declarations.
4. In the main procedure, loop through all units, check for unit declarations that are package specs, and pass the visible declarations of the package to the procedure you wrote in step 3.
5. If you finish early: make your local procedure recursive, so it finds types declared in subpackages.

Error Handling

- ASIS defines a series of exceptions, most notably Asis_Failed and Inappropriate_Asis_Element.
- Before raising an exception, ASIS sets a Status and Diagnosis.
- Details on a raised exception can be obtained with calls to Environment.Status and Environment.Diagnosis.
- Utilities written on top of ASIS can follow the same convention by calling Environment.Set_Status before raising an exception.

Catching ASIS Exceptions

Consider including these exception handlers in every ASIS program you write:

```
exception
  when Asis.Asis_Failed | Asis.Asis_Inappropriate_Element =>
    Text_Io.Put_Line ("*** ASIS Exception: " & Asis.Strings.
          To_Standard_String (Asis.Environment.Diagnosis));
  when others =>
    Text_Io.Put_Line ("*** Unknown Exception: " & Debug_Tools.
          Get_Exception_Name);
end;
```

Hints and Tips: The Command Line

- **Including view names and unit names in a program reduces its reusability, forcing you to recompile and relink for each view**
- **Apex includes Posix-compliant interfaces that allow you to pass parameters to a main program via the UNIX command line**
- **Example:**

```
View_Name : constant String :=
     Posix.To_String (Posix.Value
          (Posix_Process_Environment.Argument_List, 2));
-- retrieves the first parameter from the command line
```

Hints and Tips: Reusable Components

- **Create ASIS utilities so they are general enough to be reusable**
- **Build a library of reusable components that make writing new tool easier**
- **A set of utilities has been provided as part of this course, called asis_utilities.ss**
  - **Library_Utilities, for initialization and finalization of libraries, including the handling of relative pathnames**
  - **other utility packages, one package for each Asis package**

Hints and Tips:

- Use the Operations packages locally to get visibility to operators.
- Create a renaming package, with and use it to keep from typing in extra-long package names.
  - The asis_utilities.ss includes one called Asis_Renames
- Combine utilities into single executables, using command-line parameters to distinguish commands.
  - Keeps the number of large executables down
- Be consistent with your names and your style

Final Exercise:

Modify display_visible_types to include:

- **error handling of ASIS exceptions**
- **retrieving the view name from the command line**
- **use of Library_Utilities to handle both relative pathnames, and the limiting of the compilation units to just those in the view specified**
- **if desired, change the longer package names to the shorter ones by using Asis_Renames**