European Component Oriented Architecture (ECOA®) Collaboration Programme: Architecture Specification Part 10: Ada Language Binding

BAE Ref No: IAWG-ECOA-TR-003
Dassault Ref No: DGT 144476-E

Issue: 5

Prepared by
BAE Systems (Operations) Limited and Dassault Aviation

Note: *This specification represents the output of a research programme and contains mature high-level concepts, though low-level mechanisms and interfaces remain under development and are subject to change. This version of documentation is recommended as appropriate for limited lab-based evaluation only. Product development should rely on the DefStan or BNAE publications of the ECOA standard.*

Contents

Figures

Tables

0 Introduction

This Architecture Specification provides the specification for creating ECOA®-based systems. It describes the standardised programming interfaces and data-model that allow a developer to construct an ECOA®-based system. It uses terms defined in the Definitions (Architecture Specification Part 2). The details of the other documents comprising the rest of this Architecture Specification can be found in Section 3.

This document is Part 10 of the Architecture Specification, and describes the Ada 95 (reference ISO/IEC 8652:1995(E) with COR.1:2000) language binding for the Module and Container APIs that facilitate communication between the Module Instances and their Container in an ECOA® system.

The document is structured as follows:

- Section 6 describes the Module to Language Mapping;
- Section 7 describes the method of passing parameters;
- Section 8 describes the Module Context;
- Section 9 describes the basic types that are provided and the types that can be derived from them;
- Section 10 describes the Module Interface;
- Section 11 describes the Container Interface;
- Section 12 describes the Container Types;
- Section 13 describes the External Interface;
- Section 14 provides a reference Ada specification for the ECOA® package, usable in any Ada binding implementation;

1 Scope

This Architecture Specification specifies a uniform method for design, development and integration of software systems using a component oriented approach.

2 Warning

This specification represents the output of a research programme and contains mature high-level concepts, though low-level mechanisms and interfaces remain under development and are subject to change. This standard of documentation is recommended as appropriate for limited lab-based evaluation only. Product development based on this standard of documentation is not recommended.

3 Normative References

Architecture Specification Part 1	IAWG-ECOA-TR-001 / DGT 144474 Issue 5 Architecture Specification Part 1 – Concepts
Architecture Specification Part 2	IAWG-ECOA-TR-012 / DGT 144487 Issue 5 Architecture Specification Part 2 – Definitions
Architecture Specification Part 3	IAWG-ECOA-TR-007 / DGT 144482 Issue 5 Architecture Specification Part 3 – Mechanisms
Architecture Specification Part 4	IAWG-ECOA-TR-010 / DGT 144485 Issue 5 Architecture Specification Part 4 – Software Interface
Architecture Specification Part 5	IAWG-ECOA-TR-008 / DGT 144483 Issue 5 Architecture Specification Part 5 – High Level Platform Requirements
Architecture Specification Part 6	IAWG-ECOA-TR-006 / DGT 144481 Issue 5 Architecture Specification Part 6 – ECOA® Logical Interface
Architecture Specification Part 7	IAWG-ECOA-TR-011 / DGT 144486 Issue 5 Architecture Specification Part 7 – Metamodel
Architecture Specification Part 8	IAWG-ECOA-TR-004 / DGT 144477 Issue 5 Architecture Specification Part 8 – C Language Binding
Architecture Specification Part 9	IAWG-ECOA-TR-005 / DGT 144478 Issue 5 Architecture Specification Part 9 – C++ Language Binding

Architecture Specification Part 10	IAWG-ECOA-TR-003 / DGT 144476 Issue 5 Architecture Specification Part 10 – Ada Language Binding
Architecture Specification Part 11	IAWG-ECOA-TR-031 / DGT 154934 Issue 5 Architecture Specification Part 11 – High Integrity Ada Language Binding
ISO/IEC 8652:1995(E) with COR.1:2000	Ada95 Reference Manual Issue 1
ISO/IEC 9899:1999(E)	Programming Languages – C
ISO/IEC 14882:2003(E)	Programming Languages C++
SPARK_LRM	The SPADE Ada Kernel (including RavenSPARK) Issue 7.3

4 Definitions

For the purpose of this standard, the definitions given in Architecture Specification Part 2 apply.

5 Abbreviations

API	Application Programming Interface
ECOA	European Component Oriented Architecture. ECOA® is a registered trademark.
PINFO	Persistent Information
UK	United Kingdom
UTC	Coordinated Universal Time
XML	eXtensible Markup Language

6 Module to Language Mapping

This section gives an overview of the Module and Container APIs, in terms of filename and the overall structure of the files.

The Ada 95 language allows tagged types (which allow object-oriented behaviour), however the Ada bindings will not use tagged types. This corresponds to traditional use within the avionics industry in the UK. Therefore the mapping is similar to C, apart from support for proper namespacing using Packages. The filename mapping is specified in Table 1.

The Module Interface will be composed of a set of procedures corresponding to each entry-point of the Module Implementation. The declaration of these procedures will be accessible in a package spec file called `#module_impl_name#.ads`.

The Container Interface will be composed of a set of procedures corresponding to the required operations. The declaration of these procedures will be accessible in a package spec file called `#module_impl_name#_Container.ads`.

The Container Types will be composed of the types which the Module Implementation needs in order to declare, use and store various handles. The declaration of these types will be accessible in a package spec file called `#module_impl_name#_Container_Types.ads`.

A dedicated structure named Context_Type, and called Module Context structure in the rest of the document will be generated by the ECOA toolchain in the Module Container specification (`#module_impl_name#_Container.ads`) and shall be extended by the Module implementer to contain all the user variables of the Module. This structure will be allocated by the Container before Module Instance start-up and passed to the Module Instance in each activation entry-point (i.e. received events, received request-response and asynchronous request-response sent call-back).

Figure 1 shows the relationship between the Ada files mentioned above, whilst Table 1 shows the filename mappings.

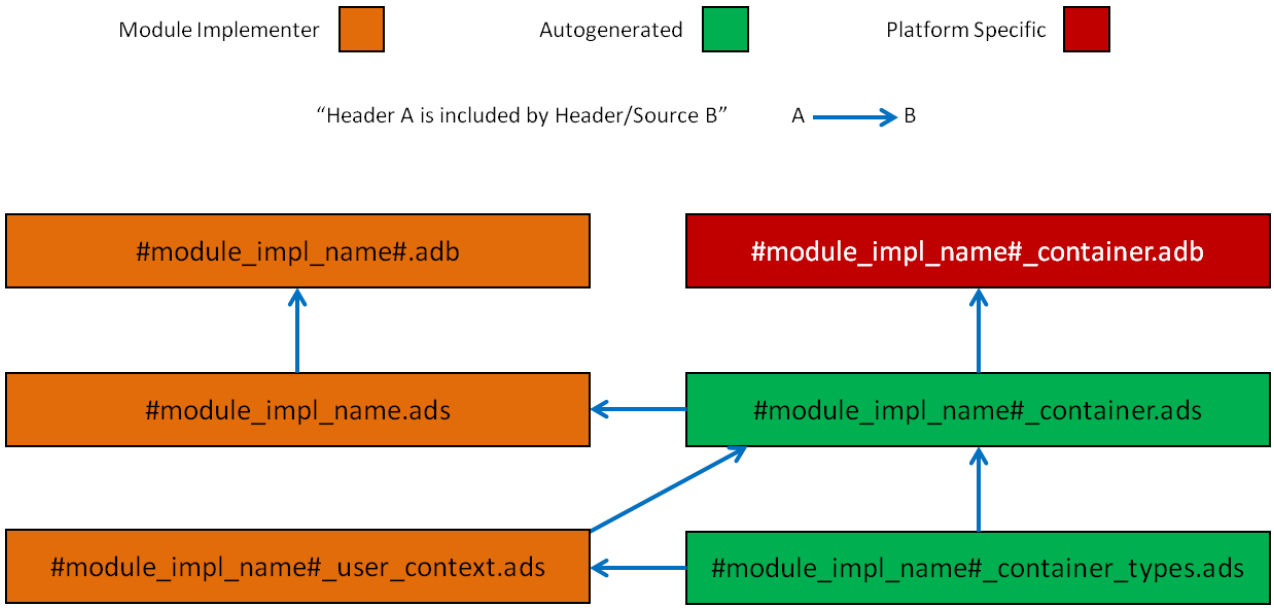

Figure 1 Ada Files Organization

Table 1 Filename Mapping for Ada 95

Filename	Use
#module_impl_name#.ads	Package **#module_impl_name#** specifies the Module interface.
#module_impl_name#.adb	Package body **#module_impl_name#** implements the Module interface.
#module_impl_name#_Container.{ads|adb}	Package **#module_impl_name#**_Container specifies and implements the Container Interface (functions provided by the Container and callable by the Module). It also specifies the standard Module context information. The Container may actually be a collection of source files depending upon the platform implementation.
#module_impl_name#_Container_Types.ads	Package **#module_impl_name#**_Container_Types specifies Container Types declaration (Container-level data types usable by the Module). These types are related to the Container for a Module Implementation and are functionally related to the **#module_impl_name#_Container** namespace, however the Ada language requires the types to be declared in a package that matches the filename i.e. **#module_impl_name#_Container_Types**.
#module_impl_name#_User_Context.ads	Extensions to Module Context. These types are related to the Module Implementation and are functionally related to the **#module_impl_name#** namespace, however the Ada language requires the types to be declared in a package that matches the filename i.e. **#module_impl_name#**_User_Context.

Templates for the files in Table 1 are provided below:

6.1 Module Interface Template

```
-------------------------------------------------------------------
-- @file #module_impl_name#.ads
-- Module Interface package specification for Module #module_impl_name#
-- Generated automatically from specification; do not modify here
-------------------------------------------------------------------

-- Standard ECOA Types
with ECOA;
-- Additionally Created Types
with #additionally_created_types#;
-- Include Container
```

```
with #module_impl_name#_Container;
-- Include Container Types
with #module_impl_name#_Container_Types;
-- Include User Context
with #module_impl_name#_User_Context;

package #module_impl_name# is

   procedure INITIALIZE_Received
    (Context : in out #module_impl_name#_Container.Context_Type);

   procedure START_Received
     (Context : in out #module_impl_name#_Container.Context_Type);

   procedure STOP_Received
     (Context : in out #module_impl_name#_Container.Context_Type);

   procedure SHUTDOWN_Received
     (Context : in out #module_impl_name#_Container.Context_Type);

   procedure REINITIALIZE_Received
     (Context : in out #module_impl_name#_Container.Context_Type);

   -- Event operation handlers specifications
   #list_of_event_operations_specifications#

   -- Request-Response operation handlers specifications
   #list_of_request_response_operations_specifications#

   -- Versioned Data Notifying operation handlers specifications
   #list_of_versioned_data_notifying_operations_specifications#

   -- If this is a Supervision module then additional APIs are declared
   -- Supervision Module API for lifecycle operations (one set per
   -- non-supervision module instance, following the order that the module
   -- instances are defined in the XML)
   --   lifecycle_notification__#module_instance_name#
   --   error_notification__#module_instance_name#

   -- Supervision Module API for lifecycle operations (one per trigger
   -- instance
   --   lifecycle_notification__#module_instance_name#

   -- Supervision Module API for lifecycle operations (one per dynamic
```

```
   -- trigger instance)
   --   lifecycle_notification__#module_instance_name#

   -- Availability Changed API call specification(s) if this is a supervision
   -- module and the component requires at least one service
   #list_of_service_availability_changed_call_specifications#

   -- Provider Changed API call specification(s) if this is a supervision
   -- module and the component requires at least one service
   #list_of_service_provider_changed_call_specifications#

   -- Error notification handlers specifications for supervised modules if this
   -- module is a supervision module
   #list_of_error_notification_operations_specifications#

   -- Error notification handler specification if this module is a Fault
   -- Handler
   #error_notification_operation_specification#

end #module_impl_name#;
```

```
-------------------------------------------------------------------
-- @file #module_impl_name#.adb
-- Module Interface package for Module #module_impl_name#
-- Generated automatically from specification; do not modify here
-- autogenerated by the ECOA toolset and filled in by the module
-- developer.
-------------------------------------------------------------------

-- Standard ECOA Types
with ECOA;
-- Additionally Created Types
with #additionally_created_types#;
-- Include Container Types
with #module_impl_name#_Container_Types;
-- Include Container
with #module_impl_name#_Container;
-- Additional children or other packages implementing the module
with #additional_with_clauses#;

package body #module_impl_name# is

   -- Event operation handlers
```

```
   #list_of_event_operations#

   -- Request-Response operation handlers
   #list_of_request_response_operations#

   -- Lifecycle operation handlers
   #list_of_lifecycle_operations#

   -- Error notification handlers specifications for supervised modules if this
   -- module is a supervision module
   #list_of_error_notification_operations_specifications#

   -- Error notification handler specification if this module is a Fault
   -- Handler
   #error_notification_operation_specification#

end module_impl_name#;
```

6.2 Container Interface Template

```
--------------------------------------------------------------------
-- @file #module_impl_name#_Container.ads
-- Container Interface package specification for Module #module_impl_name#
-- Generated automatically from specification; do not modify here
--------------------------------------------------------------------

-- Standard ECOA Types
with ECOA;
-- Additionally Created Types
with #additionally_created_types#;
-- Include Container Types
with #module_impl_name#_Container_Types;
-- Include module user context
with #module_impl_name#_User_Context;

package #module_impl_name#_Container is

   -- Module Implementation Context data type is specified here. This enables a
   -- module instance to hold its own private data in a non-OO fashion.

   type Context_Type is record
      -- Standard Container context information
      Operation_Timestamp : ECOA.Timestamp_Type;
```

```
      -- A hook to implementation dependant private data
      Platform_Hook : System.Address;

      -- Information that is private to a module implementation
      User_Context        : #module_impl_name#_User_Context.User_Context_Type;

      Warm_Start_Context  :
        #module_impl_name#_User_Context.Warm_Start_Context_Type;

   end record;

   procedure Log_Trace
     (Context : in out Context_Type;
      Log     : in     ECOA.Log_Type);

   procedure Log_Debug
     (Context : in out Context_Type;
      Log     : in     ECOA.Log_Type);

   procedure Log_Info
     (Context : in out Context_Type;
      Log     : in     ECOA.Log_Type);

   procedure Log_Warning
     (Context : in out Context_Type;
      Log     : in     ECOA.Log_Type);

   procedure Raise_Error
     (Context : in out Context_Type;
      Log     : in     ECOA.Log_Type);

   procedure Raise_Fatal_Error
     (Context : in out Context_Type;
     Log      : in     ECOA.Log_Type);

   procedure Get_Relative_Local_Time
     (Context             : in out Context_Type;
      Relative_Local_Time :    out ECOA.HR_Time_Type);

   procedure Get_UTC_Time
     (Context  : in out Context_Type;
      UTC_Time :    out ECOA.Global_Time_Type;
      Status   :    out ECOA.Return_Status_Type);
```

```
   procedure Get_Absolute_System_Time
     (Context              : in out Context_Type;
      Absolute_System_Time :    out ECOA.Global_Time_Type;
      Status               :    out ECOA.Return_Status_Type);

   procedure Get_Relative_Local_Time_Resolution
     (Context                         : in out Context_Type;
      Relative_Local_Time_Resolution :    out ECOA.Duration);

   procedure Get_UTC_Time_Resolution
     (Context             : in out Context_Type;
      UTC_Time_Resolution :    out ECOA.Duration);

   procedure Get_Absolute_System_Time_Resolution
     (Context                          : in out Context_Type;
      Absolute_System_Time_Resolution :    out ECOA.Duration);

   -- Event operation call specifications
   #event_operation_call_specifications#

   -- Request-response call specifications
   #request_response_call_specifications#

   -- Versioned data call specifications
   #versioned_data_call_specifications#

   -- Functional parameters call specifications
   #properties_call_specifications#

   -- Recovery action service API call specification if this is a Fault Handler
   -- module
   #recovery_action_call_specification#

   -- Get Service Availability API call specification(s)if this is a
   -- supervision module and the component requires at least one service
   #list_of_get_service_availability_call_specifications#

   -- Set Service Availability API call specification(s) if this is a
   -- supervision module and the component provides at least one service
   #list_of_set_service_availability_call_specifications#

   -- If this is a Supervision module then additional APIs are declared
   -- Supervision Module API for lifecycle operations (one set per non-
   -- supervision module instance, following the order that the module
```

```
   -- instances are defined in the XML, then trigger instance, then
   -- dynamic trigger instance)
   --  get_lifecycle_state_#module_instance_name#
   --  STOP__#module_instance_name#
   --  START__#module_instance_name#
   --  INITIALIZE__#module_instance_name#
   --  SHUTDOWN__#module_instance_name#

   -- Persistent Information management operations
   #PINFO_read_call_specifications#
   #PINFO_write_call_specifications#
   #PINFO_seek_call_specifications#

   -- Context management operation
   #Save_Warm_Start_Context_operation#

end #module_impl_name#_Container;
```

6.3 Container Types Template

```
--------------------------------------------------------------------
-- @file #module_impl_name#_Container_Types.ads
-- Container Types package specification for Module #module_impl_name#
-- Generated automatically from specification; do not modify here
--------------------------------------------------------------------

-- Standard ECOA Types
with ECOA;

package #module_impl_name#_Container_Types is

   -- The following describes the data types generated with regard to APIs:
   -- For any Versioned Data Read Access: data_handle
   -- For any Versioned Data Write Access: data_handle

   -- If this is a Supervision module then additional data types are declared:
   -- Service Availability API:
     -- if the component has at least one required service
      -- reference_id enumeration
     -- if the component has at least one provided service
      -- service_id enumeration

end #module_impl_name#_Container_Types;
```

6.4 User Module Context Template

```
-------------------------------------------------------------------
-- @file #module_impl_name#_User_Context.ads
-- This is the module implementation private user context data type
-- that is included in the module context.
-------------------------------------------------------------------

-- Standard ECOA Types
with ECOA;
-- Additionally Created Types
with #additionally_created_types#;
-- Include Container Types
with #module_impl_name#_Container_Types;

package #module_impl_name#_User_Context is

   type User_Context_Type is record
       -- Declare the User Module Context "local" data here.

   end record;

   type Warm_Start_Context_Type is record
       -- Declare the Module Warm Start Context "local" data here.

   end record;

end module_impl_name#_User_Context;
```

Data declared within the Module User Context and the Module Warm Start Context can be of any type.

7 Parameters

In the Ada programming language, the manner in which parameters are passed is specified as '**in**', '**out**' or '**in out**'. '**in'** Parameters are only passed into a procedure; '**out**' parameters are only passed out from a procedure; and **'in out**' parameters are passed in, modified and passed out from a procedure. The compiler then makes an appropriate choice as to whether to pass-by-value or pass-by-reference.

Table 2 Parameter Typing

	Input parameter	*Output parameter*	*Input and Output parameter*
Simple type	`in`	`out`	`in out`
Complex type	`in`	`out`	`in out`

8 Module Context

In the Ada language binding, the Module Context is a structure which holds both the user local data (called "User Module Context" and "Warm Start Context") and Infrastructure-level technical data (which is implementation dependant). The structure is defined in the Container Interface.

The following shows the Ada syntax for the Module Context:

```
-------------------------------------------------------------------
-- @file "#module_impl_name#_Container.ads"
-- Container package specification for Module #module_impl_name#
-- Generated automatically from specification; do not modify here
-------------------------------------------------------------------
with System;

-- Standard ECOA Types
with ECOA;
-- Include Container Types
with #module_impl_name#_Container_Types;
-- Additionally Created Types
with #additionally_created_types#;
-- Include module user context
with #module_impl_name#_User_Context;

package #module_impl_name#_Container is

   -- Module Implementation Context data type is specified here. This enables a
   -- module instance to hold its own private data in a non-OO fashion.

   type Context_Type is record
      -- Standard context information
      Operation_Timestamp : ECOA.Timestamp_Type;

      -- A hook to implementation dependant private data
      Platform_Hook : System.Address;

      -- Information that is private to a module implementation
      User_Context : #module_impl_name#_User_Context.User_Context_Type;

      Warm_Start_Context  :
        #module_impl_name#_User_Context.Warm_Start_Context_Type;

   end record;

end #module_impl_name#_Container;
```

8.1 User Module Context

The Ada syntax for the user context is shown below (including an example data item; `My_Counter`) and the Module Warm Start Context (including an example data item My_Data and validity flag Warm_Start_Valid:

```
-------------------------------------------------------------------
-- @file "#module_impl_name#_User_Context.ads"
-- This is the module implementation private user context data type
-- that is included in the module context.
-------------------------------------------------------------------

-- Standard ECOA Types
with ECOA;
-- Include Container Types
with #module_impl_name#_Container_Types;
-- Additionally Created Types
with #additionally_created_types#;

package #module_impl_name#_User_Context is

   type User_Context_Type is record
      -- Example user context
   My_Counter : Integer;
   end record;

   type Warm_Start_Context_Type is record
      -- Example warm start context
      Warm_Start_Valid : ECOA.Boolean_8_Type; -- example of validity flag
   My_Data : Unsigned_Long;

   end record;

end module_impl_name#_User_Context;
```

EXAMPLE The following illustrates the usage of the Module context in the entry-point corresponding to an event-received:

```
-------------------------------------------------------------------
-- @file "#module_impl_name#.adb"
-- Generic operation implementation example
-------------------------------------------------------------------

-- Standard ECOA Types
with ECOA;
-- Additionally Created Types
with #additionally_created_types#;
-- Include Container Types
```

```
with #module_impl_name#_Container_Types;
-- Include Container
with #module_impl_name#_Container;
-- Additional children or other packages implementing the module
with #additional_with_clauses#;

package body #module_impl_name# is

   procedure #operation_name#_Received
     (Context : in out #module_impl_name#_Container.Context_Type;
      #event_parameters#)
   is

   begin

      -- To be implemented by the module.

      -- Increments a local user defined counter.
      Context.User_Context.My_Counter := Context.User_Context.My_Counter + 1;

   end #operation_name#_Received;

end module_impl_name#;
```

The user extensions to Module Context need to be known by the Container in order to allocate the required memory area. This means that the component supplier is requested to provide the associated header file. If the supplier does not want to divulge the original contents of the header file, then:

- It may be replaced by an array with a size equivalent to the original data; or
- Memory management may be dealt with internally to the code, using memory allocation functions, however the current Architecture Specification does not specify any memory allocation function. So, this case may lead to non-portable code.

To extend the Module Context structure, the Module implementer shall define the User Module Context structure, named `#module_impl_name#_User_Context`, in a package spec file called `#module_impl_name#_User_Context.ads`. All the private data of the Module Implementation shall be added as members of this record, and will be accessible within the “User_Context” field of the Module Context.

The Module Context structure will be passed by the Container to the Module as the first parameter for each operation (i.e. received events, received request-response and asynchronous request-response sent call-back). The Module Context defines the instance of the Module being invoked by the operation. This structure shall be passed by the Module to all Container interface API functions it can call.

The Module Context will also be used by the Container to automatically timestamp operations on the emitter/requester side using an ECOA-provided attribute called `operation_timestamp`. The Container also provides a utility function to retrieve this from the Module Instance Context. The way this structure is populated by the ECOA infrastructure is detailed in reference ISO/IEC 8652:1995(E) with COR.1:2000.

9 Types

This section describes the convention for creating namespaces, and how the ECOA basic types and derived types are represented in Ada.

9.1 Filenames and Namespace

The type definitons are contained within one or more namespaces: all types for specific namespace defined in `#namespace1#[__#namespacen#].types.xml` shall be placed in a file called `#namespace1#[-#namespacen#].ads`.

Below is an example of a simple type being defined within a nested namespace in Ada.

```
--
-- @file #namespace1#[-#namespacen#].ads
-- Data-type declaration file
-- Generated automatically from specification; do not modify here
--

package #namespace1#[.#namespacen#] is

   type #simple_type_name# is new #basic_type_name# range #min# .. #max#;

end #namespace1#[.#namespacen#];
```

9.2 Basic Types

Basic types in Ada 95, shown in Table 3, shall be located in the “ECOA” namespace and hence in `ECOA.ads`.

Table 3 Ada 95 Basic Types

ECOA Basic Type	Ada 95 Type
ECOA:boolean8	ECOA.Boolean_8_Type
ECOA:int8	ECOA.Signed_8_Type
ECOA:char8	ECOA.Character_8_Type
ECOA:byte	ECOA.Byte_Type
ECOA:int16	ECOA.Signed_16_Type
ECOA:int32	ECOA.Signed_32_Type
ECOA:int64	ECOA.Signed_64_Type
ECOA:uint8	ECOA.Unsigned_8_Type
ECOA:uint16	ECOA.Unsigned_16_Type
ECOA:uint32	ECOA.Unsigned_32_Type
ECOA:uint64	ECOA.Unsigned_64_Type
ECOA:float32	ECOA.Float_32_Type
ECOA:double64	ECOA.Float_64_Type

Ada provides the 'First and 'Last attributes, so there is no requirement to refer to explicit constants for the maximum and minimum values of the type range.

All basic types shall be specified with a representation clause to ensure they occupy the correct number of bits, and have the correct alignment. Derived Types

9.2.1 Simple Types

The Ada syntax for a Simple Type called "#simple_type_name#" with an optional restricted range, which is derived from a Basic Type is:

```
type #simple_type_name# is new #basic_type_name# range #min# .. #max#;
```

9.2.2 Constants

The syntax for declaring a constant called "#constant_name#" of type #type_name# in Ada is as follows:

```
#constant_name# : constant #type_name# := #constant_value#;
```

Where #constant_value# is either an integer or a floating-point value, compatible with the type.

9.2.3 Enumerations

For an enumerated type named #enum_type_name#, a set of constants named from #enum_value_name_1# to #enum_value_name_n# are defined with a set of optional values named #enum_value_value_1# to #enum_value_value_n#. The syntax is defined below.

The order of fields in the type shall follow the order of fields in the XML definition.

```
type #enum_type_name# is new #base_type_name#;
#enum_type_name#_#enum_value_name_1# : constant #enum_type_name# :=
  #enum_value_value_1#;
#enum_type_name#_#enum_value_name_2# : constant #enum_type_name# :=
  #enum_value_value_2#;
--…
#enum_type_name#_#enum_value_name_n# : constant #enum_type_name# :=
  #enum_value_value_n#;
```

Where:

- `#enum_value_name_X#` is the name of a label
- `#enum_value_value_X#` is the optional value of the label. If not set, this value is computed from the previous label value, by adding 1 (or set to 0 if it is the first label of the enumeration).

9.2.4 Records

The Ada syntax for a record type named `#record_type_name#` with a set of fields named `#field_name1#` to `#field_namen#` of given types `#data_type_1#` to `#data_type_n#` is given below.

The order of fields in the Ada record shall follow the order of fields in the XML definition.

```
type #record_type_name# is
   record
      #field_name1# : #data_type_1#;
      #field_name2# : #data_type_2#;
      --…
      #field_namen# : #data_type_n#;
   end record;
```

9.2.5 Variant Records

The syntax for a variant record named `#variant_record_type_name#` containing:

- a set of fields (named `#field_name1#` to `#field_namen#`) of given types `#data_type_1#` to `#data_type_n#`
- optional fields (named `#optional_field_name1#` to `#optional_field_namen#`) of type (`#optional_type_name1#` to `#optional_type_namen#`) with selector `#selector_name#` of type `#selector_type_name#`

is given below.

The order of fields in the Ada record shall follow the order of fields in the XML definition.

```
-- #selector_type_name# can be of any simple basic type, or an enumeration

type #variant_record_type_name# (#selector_name# : #selector_type_name#) is
   record
      #field_name1# : #data_type_1#;
```

```
      #field_name2# : #data_type_2#;
      --…
      #field_namen# : #data_type_n#;
      case #selector_name# is
         when #selector_value_constant1# =>
           #optional_field_name1# : #optional_type_name1#;
         when #selector_value_constant2# =>
           #optional_field_name2# : #optional_type_name2#;
         --…
         when #selector_value_constantn# =>
           #optional_field_namen# : #optional_type_namen#;
      end case;
   end record;
```

9.2.6 Fixed Arrays

The Ada syntax for a fixed array named `#array_type_name#` of `#max_number#` elements with index range `0 to #max_number#-1`, and with elements of type `#data_type_name#` is given below. The index to an array must be specified as a distinct type.

```
type #array_type_name#_Index is new ECOA.Unsigned_32_Type range
  0..#max_number#-1;
type #array_type_name# is array (#array_type_name#_Index) of #data_type_name#;
```

9.2.7 Variable Arrays

The Ada syntax for a variable array (named `#var_array_type_name#`) of `#max_number#` elements with index range `0 to #max_number#-1`, and with elements of type `#data_type_name#` and a current size of `Current_Size` is given below.

```
type #var_array_type_name#_Size is new ECOA.Unsigned_32_Type range
  0..#max_number#;
subtype #var_array_type_name#_Index is #var_array_type_name#_Size range
  0..#max_number#-1;
type #var_array_type_name#_Data is array (#var_array_type_name#_Index) of
  #data_type_name#;

type #var_array_type_name# is
   record
      Current_Size : #var_array_type_name#_Size;
      Data         : #var_array_type_name#_Data;
   end record;
```

9.3 Predefined Types

The data types described in the following sections are also defined in the ECOA namespace.

9.3.1 ECOA:return_status

In Ada `ECOA:return_status` translates to `ECOA.Return_Status_Type`, with the enumerated values shown below:

```
package ECOA is
   …
   type Return_Status_Type is new Unsigned_32_Type;
   Return_Status_Type_OK                        : constant Return_Status_Type
     := 0;
   Return_Status_Type_INVALID_HANDLE            : constant Return_Status_Type
     := 1;
   Return_Status_Type_DATA_NOT_INITIALIZED      : constant Return_Status_Type
     := 2;
   Return_Status_Type_NO_DATA                   : constant Return_Status_Type
     := 3;
   Return_Status_Type_INVALID_IDENTIFIER        : constant Return_Status_Type
     := 4;
   Return_Status_Type_NO_RESPONSE               : constant Return_Status_Type
     := 5;
   Return_Status_Type_OPERATION_ALREADY_PENDING : constant Return_Status_Type
     := 6;
   Return_Status_Type_INVALID_SERVICE_ID        : constant Return_Status_Type
     := 7;
   Return_Status_Type_CLOCK_UNSYNCHRONIZED      : constant Return_Status_Type
     := 8;
   Return_Status_Type_INVALID_TRANSITION        : constant Return_Status_Type
     := 9;
   Return_Status_Type_RESOURCE_NOT_AVAILABLE    : constant Return_Status_Type
     := 10;
   Return_Status_Type_OPERATION_NOT_AVAILABLE   : constant Return_Status_Type
     := 11;
   Return_Status_Type_PENDING_STATE_TRANSITION  : constant Return_Status_Type
     := 12;
   Return_Status_Type_INVALID_PARAMETER         : constant Return_Status_Type
     := 13;
   …
end ECOA;
```

9.3.2 ECOA:hr_time

The binding for hr_time makes use of ECOA:Seconds and ECOA:Nanoseconds types (section 9.3.16), and is defined as:

```
package ECOA is
   …
```

```
   type HR_Time_Type is
      record
         Seconds     : Seconds_Type;
         Nanoseconds : Nanoseconds_Type;
      end record;
   for HR_Time_Type'size use 64;
   for HR_Time_Type'Alignment use 4;
   …
end ECOA;
```

9.3.3 ECOA:global_time

The binding for global_time makes use of ECOA:Seconds and ECOA:Nanoseconds types (section 9.3.16), and is defined as:

```
package ECOA is
   …
   type Global_Time_Type is
      record
         Seconds     : Seconds_Type;
         Nanoseconds : Nanoseconds_Type;
      end record;
   for Global_Time_Type'size use 64;
   for Global_Time_Type'Alignment use 4;
   …
end ECOA;
```

9.3.4 ECOA:duration

The binding for duration makes use of ECOA:Seconds and ECOA:Nanoseconds types (section 9.3.16), and is defined as:

```
package ECOA is
   …
   type Duration_Type is
      record
         Seconds     : Seconds_Type;
         Nanoseconds : Nanoseconds_Type;
      end record;
   for Duration_Type'size use 64;
   for Duration_Type'Alignment use 4;
   …
end ECOA;
```

9.3.5 ECOA:timestamp

The binding for timestamp makes use of ECOA:Seconds and ECOA:Nanoseconds types (section 9.3.16), and is defined as:

```
package ECOA is
   …
   type Timestamp_Type is
      record
         Seconds     : Seconds_Type;
         Nanoseconds : Nanoseconds_Type;
      end record;
   for Timestamp_Type'size use 64;
   for Timestamp_Type'Alignment use 4;
   …
end ECOA;
```

9.3.6 ECOA:log

The syntax for a log is:

```
package ECOA is
   …
   type Log_Elements_Size_Type is range 0..256;
   for Log_Elements_Size'size use 32;
   for Log_Elements_Size'Alignment use 4;
   subtype Log_Elements_Index_Type is Log_Elements_Size_Type range 0..255;

   type Log_Elements_Type is array (Log_Elements_Index_Type) of
     ECOA.Character_8_Type;
   for Log_Elements_Type'size use 2048;
   for Log_Elements_Type'Alignment use 4;

   type Log_Type is
      record
         Current_Size : Log_Elements_Size_Type;
         Data         : Log_Elements_Type;
      end record;
   for Log_Type'size use 2080;
   for Log_Type'Alignment use 4;
   …
end ECOA;
```

9.3.7 ECOA:module_states_type

In Ada `ECOA:module_states_type` translates to `ECOA.Module_States_Type`, with the enumerated values shown below:

```
package ECOA is
   …
   type Module_States_Type is new Unsigned_32_Type;
   Module_States_Type_IDLE    : constant Module_States_Type := 0;
   Module_States_Type_READY   : constant Module_States_Type := 1;
   Module_States_Type_RUNNING : constant Module_States_Type := 2;
   …
end ECOA;
```

9.3.8 ECOA:module_error_type

In Ada `ECOA:module_error_type` translates to `ECOA.Module_Error_Type`, with the enumerated values shown below:

```
package ECOA is
   …
   type Module_Error_Type is new Unsigned_32_Type;
   Module_Error_Type_ERROR       : constant Module_Error_Type := 0;
   Module_Error_Type_FATAL_ERROR : constant Module_Error_Type := 1;
   …
end ECOA;
```

9.3.9 ECOA:error_id

In Ada the syntax for an `ECOA:error_id` is:

```
package ECOA is
   …
   type Error_Id_Type is new Unsigned_32_Type;
   …
end ECOA;
```

9.3.10 ECOA:asset_id

In Ada the syntax for an `ECOA:asset_id` is:

```
package ECOA is
   …
   type Asset_Id_Type is new Unsigned_32_Type;
   …
end ECOA;
```

In Ada the `ECOA:asset_id` definitions will be generated in a file named ECOA-FaultHandler.ads using the following syntax:

```
-- File ECOA-FaultHandler.ads

package ECOA.FaultHandler is

   type IDs_Type is new ECOA.Unsigned_32_Type;

   IDs_Type_CMP_#component_instance_name1# : constant IDs_Type := #CMP_ID1#;
   IDs_Type_CMP_#component_instance_name2# : constant IDs_Type := #CMP_ID2#;
   IDs_Type_CMP_#component_instance_nameN# : constant IDs_Type := #CMP_IDN#;

   IDs_Type_PD_#protection_domain_name1# : constant IDs_Type := #PD_ID1#;
   IDs_Type_PD_#protection_domain_name2# : constant IDs_Type := #PD_ID2#;
   IDs_Type_PD_#protection_domain_nameN# : constant IDs_Type := #PD_IDN#;

   IDs_Type_NOD_#computing_node_name1# : constant IDs_Type := #NOD_ID1#;
   IDs_Type_NOD_#computing_node_name2# : constant IDs_Type := #NOD_ID2#;
   IDs_Type_NOD_#computing_node_nameN# : constant IDs_Type := #NOD_IDN#;

   IDs_Type_PF_#computing_platform_name1# : constant IDs_Type := #PF_ID1#;
   IDs_Type_PF_#computing_platform_name2# : constant IDs_Type := #PF_ID2#;
   IDs_Type_PF_#computing_platform_nameN# : constant IDs_Type := #PF_IDN#;

   IDs_Type_SOP_#service_operation_name1# : constant IDs_Type := #ELI_UID#;
   IDs_Type_SOP_#service_operation_name2# : constant IDs_Type := #ELI_UID#;
   IDs_Type_SOP_#service_operation_nameN# : constant IDs_Type := #ELI_UID#;

   IDs_Type_DEP_#deployment_name1# : constant IDs_Type := #DEP_ID1#;
   IDs_Type_DEP_#deployment_name2# : constant IDs_Type := #DEP_ID2#;
   IDs_Type_DEP_#deployment_nameN# : constant IDs_Type := #DEP_IDN#;

end ECOA.ErrorHandler;
```

9.3.11 ECOA:asset_type

In Ada `ECOA:asset_type` translates to `ECOA.Asset_Type`, with the enumerated values shown below:

```
package ECOA is
   …
   type Asset_Type is new Unsigned_32_Type;
   Asset_Type_COMPONENT          : constant Asset_Type := 0;
```

```
   Asset_Type_PROTECTION_DOMAIN : constant Asset_Type := 1;
   Asset_Type_NODE              : constant Asset_Type := 2;
   Asset_Type_PLATFORM          : constant Asset_Type := 3;
   Asset_Type_SERVICE           : constant Asset_Type := 4;
   Asset_Type_DEPLOYMENT        : constant Asset_Type := 5;
   …
end ECOA;
```

9.3.12 ECOA:error_type

In Ada `ECOA:error_type` translates to `ECOA.Error_Type`, with the enumerated values shown below:

```
package ECOA is
   …
   type Error_Type is new Unsigned_32_Type;;
   Error_Type_RESOURCE_NOT_AVAILABLE : constant Error_Type := 0;
   Error_Type_UNAVAILABLE            : constant Error_Type := 1;
   Error_Type_MEMORY_VIOLATION       : constant Error_Type := 2;
   Error_Type_NUMERICAL_ERROR        : constant Error_Type := 3;
   Error_Type_ILLEGAL_INSTRUCTION    : constant Error_Type := 4;
   Error_Type_STACK_OVERFLOW         : constant Error_Type := 5;
   Error_Type_DEADLINE_VIOLATION     : constant Error_Type := 6;
   Error_Type_OVERFLOW               : constant Error_Type := 7;
   Error_Type_UNDERFLOW              : constant Error_Type := 8;
   Error_Type_ILLEGAL_INPUT_ARGS     : constant Error_Type := 9;
   Error_Type_ILLEGAL_OUTPUT_ARGS    : constant Error_Type := 10;
   Error_Type_ERROR                  : constant Error_Type := 11;
   Error_Type_FATAL_ERROR            : constant Error_Type := 12;
   Error_Type_HARDWARE_FAULT         : constant Error_Type := 13;
   Error_Type_POWER_FAIL             : constant Error_Type := 14;
   Error_Type_COMMUNICATION_ERROR    : constant Error_Type := 15;
   Error_Type_INVALID_CONFIG         : constant Error_Type := 16;
   Error_Type_INITIALISATION_PROBLEM : constant Error_Type := 17;
   Error_Type_CLOCK_UNSYNCHRONIZED   : constant Error_Type := 18;
   Error_Type_UNKNOWN_OPERATION      : constant Error_Type := 19;
   Error_Type_OPERATION_OVERRATED    : constant Error_Type := 20;
   Error_Type_OPERATION_UNDERRATED   : constant Error_Type := 21;
   …
end ECOA;
```

9.3.13 ECOA:recovery_action_type

In Ada `ECOA:recovery_action_type` translates to `ECOA.Recovery_Action_Type`, with the enumerated values shown below:

```
package ECOA is
```

```
   …
   type Recovery_Action_Type is new Unsigned_32_Type;
   Recovery_Action_Type_SHUTDOWN          : constant Recovery_Action_Type := 0;
   Recovery_Action_Type_COLD_RESTART      : constant Recovery_Action_Type := 1;
   Recovery_Action_Type_WARM_RESTART      : constant Recovery_Action_Type := 2;
   Recovery_Action_Type_CHANGE_DEPLOYMENT : constant Recovery_Action_Type := 3;
   …
end ECOA;
```

9.3.14 ECOA:pinfo_filename

The syntax for a log is:

```
package ECOA is
   …
   type Pinfo_Filename_Elements_Size_Type is range 0..256;
   for Pinfo_Filename_Elements_Size_Type'size use 32;
   for Pinfo_Filename_Elements_Size_Type'Alignment use 4;
   subtype Pinfo_Filename_Elements_Index_Type is
Pinfo_Filename_Elements_Size_Type range 0..255;

   type Pinfo_Filename_Elements_Type is array
     (Pinfo_Filename_Elements_Index_Type) of ECOA.Character_8_Type;
   for Pinfo_Filename_Elements_Type'size use 2048;
   for Pinfo_Filename_Elements_Type'Alignment use 4;

   type Pinfo_Filename_Type is
      record
         Current_Size : Pinfo_Filename_Elements_Size_Type;
         Data         : Pinfo_Filename_Elements_Type;
      end record;
   for Pinfo_Filename_Type'size use 2080;
   for Pinfo_Filename_Type'Alignment use 4;
   …
end ECOA;
```

9.3.15 ECOA:seek_whence_type

In Ada `ECOA:seek_whence_type` translates to `ECOA.Seek_Whence_Type`, with the enumerated values shown below:

```
package ECOA is
   …
   type Seek_Whence_Type is new Unsigned_32_Type;
   Seek_Whence_Type_SEEK_SET : constant Seek_Whence_Type := 0;
   Seek_Whence_Type_SEEK_CUR : constant Seek_Whence_Type := 1;
```

```
   Seek_Whence_Type_SEEK_END  : constant Seek_Whence_Type := 2;
   …
end ECOA;
```

9.3.16 **ECOA:seconds and ECOA:nanoseconds**

Seconds and Nanosecond types[1] are defined as follows:

```
package ECOA is
   …
   type Seconds_Type is mod 2 ** 32;
   for Seconds_Type'Size use 32;
   for Seconds_Type'Alignment use 4;

   type Nanoseconds_Type is range 0 .. 10 ** 9 - 1;
   for Nanoseconds_Type'Size use 32;
   for Nanoseconds_Type'Alignment use 4;
   …
end ECOA;
```

9.3.17 **ECOA:request_response_id_type**

In Ada, the Request Response ID type is defined as follows:

```
package ECOA is
   …
   type Request_Response_ID_Type is new Unsigned_32_Type;
   …
end ECOA;
```

9.3.18 **ECOA:pinfo_size_type**

In Ada, the PINFO Size type is defined as follows:

```
package ECOA is
   …
   type PINFO_Size_Type is new Unsigned_32_Type;
   …
end ECOA;
```

[1] With the difference of C and C++ bindings, the Ada binding defines new types suitable for time management by limiting the possible values of the considered temporal units.

9.3.19 **ECOA:pinfo_offset_type**

In Ada, the PINFO Offset type is defined as follows:

```
package ECOA is
   …
   type PINFO_Offset_Type is new Signed_32_Type;
   …
end ECOA;
```

9.3.20 **ECOA:pinfo_position_type**

In Ada, the PINFO Position type is defined as follows:

```
package ECOA is
   …
   type PINFO_Position_Type is new Unsigned_32_Type;
   …
end ECOA;
```

10 Module Interface

10.1 Operations

This section contains details of the operations that comprise the Module API i.e. the operations that can invoked by the Container on a Module.

10.1.1 Request-response

10.1.1.1 Request Received

The following is the Ada syntax for invoking a request received by a Module Instance, where `#module_impl_name#` is the name of the Module Implementation providing the service and `#operation_name#` is the operation name. The same syntax is applicable for both synchronous and asynchronous request-response operations.

```
package #module_impl_name# is

   procedure #operation_name#_Request_Received
     (Context : in out #module_impl_name#_Container.Context_Type;
      ID      : in     ECOA.Request_Response_ID_Type;
      #request_parameters#);

end #module_impl_name#;
```

10.1.1.2 Response received

The following is the Ada syntax for an operation used by the Container to send a response to an asynchronous request response operation to the Module Instance that originally issued the request, where `#module_impl_name#` is the name of the Module Implementation providing the service and `#operation_name#` is the operation name. (The reply to a synchronous request response is provided by the return of the response).

```
package #module_impl_name# is

   procedure #operation_name#_Response_Received
     (Context : in out #module_impl_name#_Container.Context_Type;
      ID      : in     ECOA.Request_Response_ID_Type;
      Status  : in     ECOA.Return_Status_Type;
      #response_parameters#);

end #module_impl_name#;
```

10.1.2 Versioned Data Updated

The following is the Ada syntax that is used by the Container to inform a Module Instance that reads an item of versioned data that new data has been written.

```
-- Include Container Types
with #module_impl_name#_Container_Types;

package #module_impl_name# is

   procedure #operation_name#_Updated
     (Context     : in out #module_impl_name#_Container.Context_Type;
      Status      : in     ECOA.Return_Status_Type;
      Data_Handle : in
        #module_impl_name#_Container_Types.#operation_name#_Handle_Type);

end #module_impl_name#;
```

10.1.3 Event Received

The following is the Ada syntax for an event received by a Module Instance.

```
package #module_impl_name# is

   procedure #operation_name#_Received
     (Context : in out #module_impl_name#_Container.Context_Type;
      #event_parameters#);

end #module_impl_name#;
```

10.2 Module Lifecycle

10.2.1 Generic Module API

The following operations are applicable to supervision, non-supervision, Trigger and Dynamic-Trigger Module Instances.

10.2.1.1 Initialize_Received

The Ada syntax for a procedure to initialise a Module Instance is:

```
package #module_impl_name# is

   procedure INITIALIZE_Received
     (Context : in out #module_impl_name#_Container.Context_Type);

end #module_impl_name#;
```

10.2.1.2 Start_Received

The Ada syntax for a procedure to start a Module Instance is:

```
package #module_impl_name# is

   procedure START_Received
     (Context : in out #module_impl_name#_Container.Context_Type);

end #module_impl_name#;
```

10.2.1.3 Stop_Received

The Ada syntax for a procedure to stop a Module Instance is:

```
package #module_impl_name# is

   procedure STOP_Received
     (Context : in out #module_impl_name#_Container.Context_Type);

end #module_impl_name#;
```

10.2.1.4 Shutdown_Received

The Ada syntax for a procedure to shutdown a Module Instance is:

```
package #module_impl_name# is

   procedure SHUTDOWN_Received
     (Context : in out #module_impl_name#_Container.Context_Type);

```

```
end #module_impl_name#;
```

10.2.1.5 Reinitialize_Received

The Ada syntax for a procedure to reinitialise a Module Instance is:

```
package #module_impl_name# is

   procedure REINITIALIZE_Received
     (Context : in out #module_impl_name#_Container.Context_Type);

end #module_impl_name#;
```

10.2.2 Supervision Module API

The Ada syntax for an operation that is used by the Container to notify the supervision Module that a Module/Trigger/Dynamic Trigger has changed state is:

```
package #supervision_module_impl_name#

   procedure Lifecycle_Notification_#module_instance_name#
     (Context        : in out
        #supervision_module_impl_name#_Container.Context_Type;
      Previous_State : in     ECOA.Module_States_Type;
      New_State      : in     ECOA.Module_States_Type);

end #supervision_module_impl_name#;
```

The supervision Module API will contain a Lifecycle Notification procedure for every Module/Trigger/Dynamic Trigger in the Component i.e. the above API will be duplicated for every #module_instance_name# Module/Trigger/Dynamic Trigger in the Component.

`ECOA.Module_States_Type` is an enumerated type that contains all of the possible lifecycle states of the Module Instance: see section 9.3.7.

10.3 Service Availability

10.3.1 Service Availability Changed

The following is the Ada syntax for an operation used by the Container to invoke a service availability changed operation to a supervision Module Instance. The operation will only be available if the component has one or more required services. The `Reference_ID_Type` is an enumeration type defined in the Container Interface (Section 12.1.3).

```
-- Include Container Types
with #supervision_module_impl_name#_Container_Types;

package #supervision_module_impl_name# is
```

```
   procedure Service_Availability_Changed
     (Context   : in out #supervision_module_impl_name#_Container.Context_Type;
      Instance  : in
        #supervision_module_impl_name#_Container_Types.Reference_ID_Type;
      Available : in     ECOA.Boolean_8_Type);

end #supervision_module_impl_name#;
```

10.3.2 Service Provider Changed

The following is the Ada syntax for an operation used by the Container to invoke a service provider changed operation to a supervision Module Instance. The operation will only be available if the component has one or more required services. The `Reference_ID_Type` is an enumeration type defined in the Container Interface (Section 12.1.3).

```
-- Include Container Types
with #supervision_module_impl_name#_Container_Types;

package #supervision_module_impl_name# is

   procedure Service_Provider_Changed
     (Context  : in out #supervision_module_impl_name#_Container.Context_Type;
      Instance : in
        #supervision_module_impl_name#_Container_Types.Reference_ID_Type);

end #supervision_module_impl_name#;
```

10.4 Error_notification at application level

The Ada syntax for the Container to report an error to the supervision Module Instance is:

```
package #supervision_module_impl_name# is

   procedure Error_Notification_#module_instance_name#
     (Context      : in out
        #supervision_module_impl_name#_Container.Context_Type;
      Module_Error : in     ECOA.Module_Error_Type);

end #module_impl_name#;
```

10.5 Error_notification at Fault Handler level

The Ada syntax for the Container to report an error to a Fault Handler is:

```
package #fault_handler_impl_name# is
```

```
   procedure Error_Notification
     (Context    : in out #fault_handler_impl_name#_Container.Context_Type;
      Error_Id   : in     ECOA.Error_Id_Type;
      Timestamp  : in     ECOA.Timestamp_Type;
      Asset_Id   : in     ECOA.Asset_Id_Type;
      Asset_Type : in     ECOA.Asset_Type;
      Error_Type : in     ECOA.Error_Type);

end #fault_handler_impl_name#;
```

11 Container Interface

This section contains details of the operations that comprise the Container API i.e. the operations that can be called by a Module.

11.1 Operations

11.1.1 Request Response

11.1.1.1 Response Send

The Ada syntax, applicable to both synchronous and asynchronous request response operations, for sending a reply is:

```
package #module_impl_name#_Container is

   procedure #operation_name#_Response_Send
     (Context : in out Context_Type;
      ID      : in     ECOA.Request_Response_ID_Type;
      #response_parameters#;
      Status  :    out ECOA.Return_Status_Type);

end #module_impl_name#_Container;
```

The `ID` parameter is that which was passed in during the invocation of the request received operation.

11.1.1.2 Synchronous Request

The Ada syntax for a Module Instance to perform a synchronous request response operation is:

```
package #module_impl_name#_Container is

   procedure #operation_name#_Request_Sync
     (Context : in out Context_Type;
      #request_parameters#;
      #response_parameters#;
      Status  :    out ECOA.Return_Status_Type);
```

```
end #module_impl_name#_Container;
```

11.1.1.3 Asynchronous Request

The Ada syntax for a Module Instance to perform an asynchronous request response operation is:

```
package #module_impl_name#_Container is

   procedure #operation_name#_Request_Async
     (Context : in out Context_Type;
      ID      :    out ECOA.Request_Response_ID_Type;
      #request_parameters#;
      Status  :    out ECOA.Return_Status_Type);

end #module_impl_name#_Container;
```

11.1.2 Versioned Data

This section contains the Ada syntax for versioned data operations, which allow a Module Instance to:

- Get (request) Read Access
- Release Read Access
- Get (request) Write Access
- Cancel Write Access (without writing new data)
- Publish (write) new data (automatically releases write access)

- Note: the definition of versioned data handles involved in all #operation_name# is done in the Container Types ads file, as specified in Section 12.1.1.

11.1.2.1 Get Read Access

```
-- Include Container Types
with #module_impl_name#_Container_Types;

package #module_impl_name#_Container is

   procedure #operation_name#_Get_Read_Access
     (Context     : in out Context_Type;
      Data_Handle :    out
        #module_impl_name#_Container_Types.#operation_name#_Handle_Type;
      Status      :    out ECOA.Return_Status_Type);

end #module_impl_name#_Container;
```

11.1.2.2 Release Read Access

```
-- Include Container Types
with #module_impl_name#_Container_Types;

package #module_impl_name#_Container is

   procedure #operation_name#_Release_Read_Access
     (Context     : in out Context_Type;
      Data_Handle : in
        #module_impl_name#_Container_Types.#operation_name#_Handle_Type;
      Status      :    out ECOA.Return_Status_Type);

end #module_impl_name#_Container;
```

11.1.2.3 Get Write Access

```
-- Include Container Types
with #module_impl_name#_Container_Types;

package #module_impl_name#_Container is

   procedure #operation_name#_Get_Write_Access
     (Context     : in out Context_Type;
      Data_Handle :    out
        #module_impl_name#_Container_Types.#operation_name#_Handle_Type;
      Status      :    out ECOA.Return_Status_Type);

end #module_impl_name#_Container;
```

11.1.2.4 Cancel Write Access

```
-- Include Container Types
with #module_impl_name#_Container_Types;

package #module_impl_name#_Container is

   procedure #operation_name#_Cancel_Write_Access
     (Context     : in out Context_Type;
      Data_Handle : in
        #module_impl_name#_Container_Types.#operation_name#_Handle_Type;
      Status      :    out ECOA.Return_Status_Type);

end #module_impl_name#_Container;
```

11.1.2.5 Publish Write Access

```
-- Include Container Types
with #module_impl_name#_Container_Types;

package #module_impl_name#_Container is

   procedure #operation_name#_Publish_Write_Access
     (Context     : in out Context_Type;
      Data_Handle : in
        #module_impl_name#_Container_Types.#operation_name#_Handle_Type;
      Status      :    out ECOA.Return_Status_Type);

end #module_impl_name#_Container;
```

11.1.3 Events

11.1.3.1 Send

The Ada syntax for a Module Instance to perform an event send operation is:

```
package #module_impl_name#_Container is

   procedure #operation_name#_Send
     (Context : in out Context_Type;
      #event_parameters#);

end #module_impl_name#_Container;
```

11.2 Properties

This section describes the syntax for the Get_Value operation to request the Module properties.

11.2.1 Get Value

The syntax for Get_Value is shown below where:

- `#property_name#` is the name of the property used in the component definition.
- `#property_type_name#` is the name of the data-type of the property.

```
package #module_impl_name#_Container is

   procedure Get_#property_name#_Value
     (Context : in out Context_Type;
      Value   :    out #property_type_name#);

end #module_impl_name#_Container;
```

11.2.2 Expressing Property Values

Not applicable to the Ada Binding.

11.2.3 Example of Defining and Using Properties

Not applicable to the Ada Binding.

11.3 Module Lifecycle

11.3.1 Non-Supervision Container API

Container operations are only available to supervision Modules to allow them to manage the Module lifecycle of non-supervision Modules.

11.3.2 Supervision Container API

The Ada Syntax for the procedures that are called by the supervision to request the Container to command a Module/Trigger/Dynamic Trigger instance to change (lifecycle) state is:

```
package #module_impl_name#_Container is

   procedure Get_Lifecycle_State_#module_instance_name#
     (Context       : in out Context_Type;
      Current_State :    out ECOA.Module_States_Type);

   procedure Stop_#module_instance_name#
     (Context : in out Context_Type;
      Status  :    out ECOA.Return_Status_Type);

   procedure Start_#module_instance_name#
     (Context : in out Context_Type;
      Status  :    out ECOA.Return_Status_Type);

   procedure Initialize_#module_instance_name#
     (Context : in out Context_Type;
      Status  :    out ECOA.Return_Status_Type);

   procedure Shutdown_#module_instance_name#
     (Context : in out Context_Type;
      Status  :    out ECOA.Return_Status_Type);

end #module_impl_name#_Container;
```

An instance of each of the above operations is created for each Module/Trigger/Dynamic Trigger instance in the Component, where `#module_instance_name#` above represents the name of the Module/Trigger/Dynamic Trigger Instance.

11.4 Service Availability

11.4.1 Set Service Availability (Server Side)

The following is the Ada syntax for invoking the set service availability operation by a supervision Module Instance. The operation will only be available if the component has one or more provided services. The service instance is identified by the enumeration type `service_id` defined in the Container Interface (Section 12.1.2).

```
-- Include Container Types
with #supervision_module_impl_name#_Container_Types;

package #supervision_module_impl_name#_Container is

   procedure Set_Service_Availability
     (Context   : in out Context_Type;
      Instance  : in
        #supervision_module_impl_name#_Container_Types.Service_ID_Type;
      Available : in     ECOA.Boolean_8_Type;
      Status    :    out ECOA.Return_Status_Type);

end #supervision_module_impl_name#_Container;
```

11.4.2 Get Service Availability (Client Side)

The following is the Ada syntax for invoking the get service availability operation by a supervision Module Instance. The operation will only be available if the component has one or more required services. The service instance is identified by the enumeration type `reference_id` defined in the Container Interface (Section 12.1.3).

```
-- Include Container Types
with #supervision_module_impl_name#_Container_Types;

package #supervision_module_impl_name#_Container is

   procedure Get_Service_Availability
     (Context   : in out Context_Type;
      Instance  : in
        #supervision_module_impl_name#_Container_Types.Reference_ID_Type;
      Available :    out ECOA.Boolean_8_Type;
      Status    :    out ECOA.Return_Status_Type);

end #supervision_module_impl_name#_Container;
```

11.5 Logging and Fault Management

This section describes the Ada syntax for the logging and fault management procedures provided by the Container. There are six procedures:

- Trace: a detailed runtime trace to assist with debugging
- Debug: debug information
- Info: to log runtime events that are of interest e.g. changes of Module state
- Warning: to report and log warnings
- Raise_Error: to report an error from which the application may be able to recover
- Raise_Fatal_Error: to raise a severe error from which the application cannot recover.

11.5.1 Log_Trace

```
package #module_impl_name#_Container is

   procedure Log_Trace
     (Context : in out Context_Type;
      Log     : in     ECOA.Log_Type);

end #module_impl_name#_Container;
```

11.5.2 Log_Debug

```
package #module_impl_name#_Container is

   procedure Log_Debug
     (Context : in out Context_Type;
      Log     : in     ECOA.Log_Type);

end #module_impl_name#_Container;
```

11.5.3 Log_Info

```
-- Include Container Types
with #module_impl_name#_Container_Types;

package #module_impl_name#_Container is

   procedure Log_Info
     (Context : in out Context_Type;
      Log     : in     ECOA.Log_Type);

end #module_impl_name#_Container;
```

11.5.4 Log_Warning

```
package #module_impl_name#_Container is

```

```
   procedure Log_Warning
     (Context : in out Context_Type;
      Log     : in     ECOA.Log_Type);

end #module_impl_name#_Container;
```

11.5.5 Raise_Error

```
package #module_impl_name#_Container is

   procedure Raise_Error
     (Context : in out Context_Type;
      Log     : in     ECOA.Log_Type);

end #module_impl_name#_Container;
```

11.5.6 Raise_Fatal_Error

```
package #module_impl_name#_Container is

   procedure Raise_Fatal_Error
     (Context : in out Context_Type;
     Log      : in     ECOA.Log_Type);

end #module_impl_name#_Container;
```

11.6 Time Services

11.6.1 Get_Relative_Local_Time

```
package #module_impl_name#_Container is

   procedure Get_Relative_Local_Time
     (Context             : in out Context_Type;
      Relative_Local_Time :    out ECOA.HR_Time_Type);

end #module_impl_name#_Container;
```

11.6.2 Get_UTC_Time

```
package #module_impl_name#_Container is

   procedure Get_UTC_Time
     (Context  : in out Context_Type;
```

```
      UTC_Time :     out ECOA.Global_Time_Type;
      Status   :     out ECOA.Return_Status_Type);

end #module_impl_name#_Container;
```

11.6.3 Get_Absolute_System_Time

```
package #module_impl_name#_Container is

   procedure Get_Absolute_System_Time
     (Context              : in out Context_Type;
      Absolute_System_Time :     out ECOA.Global_Time_Type;
      Status               :     out ECOA.Return_Status_Type);

end #module_impl_name#_Container;
```

11.6.4 Get_Relative_Local_Time_Resolution

```
package #module_impl_name#_Container is

   procedure Get_Relative_Local_Time_Resolution
     (Context                          : in out Context_Type;
      Relative_Local_Time_Resolution :     out ECOA.Duration);

end #module_impl_name#_Container;
```

11.6.5 Get_UTC_Time_Resolution

```
package #module_impl_name#_Container is

   procedure Get_UTC_Time_Resolution
     (Context             : in out Context_Type;
      UTC_Time_Resolution :     out ECOA.Duration);

end #module_impl_name#_Container;
```

11.6.6 Get_Absolute_System_Time_Resolution

```
package #module_impl_name#_Container is

   procedure Get_Absolute_System_Time_Resolution
     (Context                          : in out Context_Type;
      Absolute_System_Time_Resolution :     out ECOA.Duration);
```

```
end #module_impl_name#_Container;
```

11.7 Persistent Information management (PINFO)

11.7.1 PINFO read

The Ada syntax for a Module Instance to read persistent data (PINFO) is:

```
package #module_impl_name#_Container is

   procedure Read_#PINFOname#
     (Context         : in out Context_Type;
      Memory_Address  : in     System.Address;
      In_Size         : in     ECOA.PINFO_Size_Type;
      Out_Size        :    out ECOA.PINFO_Size_Type;
      Status          :    out ECOA.Return_Status_Type);

end #module_impl_name#_Container;
```

11.7.2 PINFO write

The Ada syntax for a Module Instance to write persistent data (PINFO) is:

```
package #module_impl_name#_Container is

   procedure Write_#PINFOname#
     (Context         : in out Context_Type;
      Memory_Address  : in     System.Address;
      In_Size         : in     ECOA.PINFO_Size_Type;
      Status          :    out ECOA.Return_Status_Type);

end #module_impl_name#_Container;
```

11.7.3 PINFO seek

The Ada syntax for a Module Instance to seek within persistent data (PINFO) is:

```
package #module_impl_name#_Container is

   procedure Seek_#PINFOname#
     (Context      : in out Context_Type;
      Offset       : in     ECOA.PINFO_Offset_Type;
      Whence       : in     ECOA.Seek_Whence_Type;
      New_Position :    out ECOA.PINFO_Position_Type;
      Status       :    out ECOA.Return_Status_Type);
```

```
end #module_impl_name#_Container;
```

11.7.4 Example of Defining Private PINFO

Not applicable to the Ada Binding.

11.7.5 Example of Defining Public PINFO

Not applicable to the Ada Binding.

11.8 Recovery Action

This section contains the Ada syntax for the recovery action service provided to Fault Handlers by the Container.

```
package #fault_handler_impl_name#_Container is

   procedure Recovery_Action
     (Context         : in out Context_Type;
      Recovery_Action : in     ECOA.Recovery_Action_Type;
      Asset_Id        : in     ECOA.Asset_Id_Type;
      Asset_Type      : in     ECOA.Asset_Type;
      Status          :    out ECOA.Return_Status_Type);

end #fault_handler_impl_name#_Container;
```

11.9 Save Warm Start Context

The Ada syntax for a Module Instance to save its warm start (non-volatile) context is:

```
package #module_impl_name#_Container is

   procedure Save_Warm_Start_Context
     (Context : in Context_Type);

end #module_impl_name#_Container;
```

12 Container Types

This section contains details of the data types that comprise the Container API i.e. the data types that can be used by a Module.

12.1.1 Versioned Data Handles

This section contains the Ada syntax in order to define data handles for versioned data operations defined in the Container Interface.

```
package #module_impl_name#_Container_Types is

   #operation_name#_Handle_Platform_Hook_Size : constant := 32;
   type #operation_name#_Handle_Platform_Hook_Type is array
     (0..#operation_name#_Handle_Platform_Hook_Size-1) of ECOA.Byte_Type;

   --
   -- The following is the data handle structure associated to the data
   -- operation called #operation_name# of data-type #type_name#
   --

   type #operation_name#_Data_Access_Type is access all #type_name#;

   type #operation_name#_Handle_Type is
     record
       Data_Access     : #operation_name#_Data_Access_Type;
       Timestamp       : ECOA.Timestamp_Type;
       Platform_Hook   : #operation_name#_Handle_Platform_Hook_Type;
     end record;

end #module_impl_name#_Container_Types;
```

12.1.2 Service ID Enumeration

In Ada `service_id` translates to `Service_ID_Type`.

This enumeration has a value for each element *<service/>* defined in the file `.componentType`, whose name is given by its attribute *name* and the numeric value is the position (starting at 0).

The `service_id` enumeration is only available if the component provides one or more services.

```
package #supervision_module_impl_name#_Container_Types is

   type Service_ID_Type is new ECOA.Unsigned_32_Type;
   Service_ID_Type_#service_instance_name1# : constant Service_ID_Type := 0;
   --…
   Service_ID_Type_#service_instance_namen# : constant Service_ID_Type := valn;

end #supervision_module_impl_name#_Container_Types;
```

12.1.3 Reference ID Enumeration

In Ada `reference_id` translates to `Reference_ID_Type`.

This enumeration has a value for each element *<reference/>* defined in the file `.componentType`, whose name is given by its attribute *name* and the numeric value is the position (starting at 0).

The `reference_id` enumeration is only available if the component requires one or more services.

```
package #supervision_module_impl_name#_Container_Types is

   type Reference_ID_Type is new ECOA.Unsigned_32_Type;
   Reference_ID_Type_#reference_instance_name1# : constant Reference_ID_Type
     := 0;
   --…
   Reference_ID_Type_#reference_instance_namen# : constant Reference_ID_Type
     := valn;

end #supervision_module_impl_name#_Container_Types;
```

13 External Interface

This section contains the Ada syntax for the ECOA external interface provided to non-ECOA software by the Container.

Note: the choice of the language for generating external APIs is made separately from the choice of the language for generating ECOA Modules APIs. The choice of supported languages is made depending on needs that are to be taken into account in platform procurement requirements.

```
-- @file "#component_impl_name#_External_Interface.ads"
-- External Interface specification for Component
-- Implementation #component_impl_name#
-- Generated automatically from specification; do not modify here

package #component_impl_name#_External_Interface is

   procedure #external_operation_name#(#event_parameters#);

end #component_impl_name#_External_Interface;
```

14 Reference Ada Specification

```
package ECOA is

   type Boolean_8_Type is new Boolean;
   for Boolean_8_Type'Size use 8;
   type Character_8_Type is new Character;
   for Character_8_Type'Size use 8;
```

```
   type Signed_8_Type is range -127 .. 127;
   for Signed_8_Type'Size use 8;
   type Signed_16_Type is range -32767 .. 32767;
   for Signed_16_Type'Size use 16;
   type Signed_32_Type is range -2147483647 .. 2147483647;
   for Signed_32_Type'Size use 32;
   type Signed_64_Type is range -9223372036854775807 .. 9223372036854775807;
   for Signed_64_Type'Size use 64;
   type Unsigned_8_Type is mod 2 ** 8;
   for Unsigned_8_Type'Size use 8;
   type Unsigned_16_Type is mod 2 ** 16;
   for Unsigned_16_Type'Size use 16;
   type Unsigned_32_Type is mod 2 ** 32;
   for Unsigned_32_Type'Size use 32;
   type Unsigned_64_Type is mod 2 ** 64;
   for Unsigned_64_Type'Size use 64;
   type Float_32_Type is digits 6 range -3.402823466e+38 .. 3.402823466e+38;
   for Float_32_Type'Size use 32;
   type Float_64_Type is digits 15 range -1.7976931348623157e+308 ..
1.7976931348623157e+308;
   for Float_64_Type'Size use 64;
   type Byte_Type is mod 2 ** 8;
   for Byte_Type'Size use 8;

   type Return_Status_Type is new Unsigned_32_Type;
   Return_Status_Type_OK                          : constant Return_Status_Type
      := 0;
   Return_Status_Type_INVALID_HANDLE              : constant Return_Status_Type
     := 1;
   Return_Status_Type_DATA_NOT_INITIALIZED        : constant Return_Status_Type
     := 2;
   Return_Status_Type_NO_DATA                     : constant Return_Status_Type
     := 3;
   Return_Status_Type_INVALID_IDENTIFIER          : constant Return_Status_Type
     := 4;
   Return_Status_Type_NO_RESPONSE                 : constant Return_Status_Type
     := 5;
   Return_Status_Type_OPERATION_ALREADY_PENDING : constant Return_Status_Type
     := 6;
   Return_Status_Type_INVALID_SERVICE_ID          : constant Return_Status_Type
     := 7;
   Return_Status_Type_CLOCK_UNSYNCHRONIZED        : constant Return_Status_Type
     := 8;
   Return_Status_Type_INVALID_TRANSITION          : constant Return_Status_Type
```

```
   := 9;
Return_Status_Type_RESOURCE_NOT_AVAILABLE    : constant Return_Status_Type
   := 10;
Return_Status_Type_OPERATION_NOT_AVAILABLE   : constant Return_Status_Type
   := 11;
Return_Status_Type_PENDING_STATE_TRANSITION  : constant Return_Status_Type
   := 12;
Return_Status_Type INVALID_PARAMETER         : constant Return_Status_Type
   := 13;

type Seconds_Type is mod 2 ** 32;
for Seconds_Type'Size use 32;
for Seconds_Type'Alignment use 4;

type Nanoseconds_Type is range 0 .. 999999999;
for Nanoseconds_Type'Size use 32;
for Nanoseconds_Type'Alignment use 4;

type HR_Time_Type is record
   Seconds     : Seconds_Type := 0;
   Nanoseconds : Nanoseconds_Type := 0;
end record;
for HR_Time_Type'size use 64;
for HR_Time_Type'Alignment use 4;

type Global_Time_Type is record
   Seconds     : Seconds_Type := 0;
   Nanoseconds : Nanoseconds_Type := 0;
end record;
for Global_Time_Type'size use 64;
for Global_Time_Type'Alignment use 4;

type Timestamp_Type is record
   Seconds     : Seconds_Type := 0;
   Nanoseconds : Nanoseconds_Type := 0;
end record;
for Timestamp_Type'size use 64;
for Timestamp_Type'Alignment use 4;

type Duration_Type is record
   Seconds     : Seconds_Type := 0;
   Nanoseconds : Nanoseconds_Type := 0;
end record;
for Duration_Type'size use 64;
```

```
for Duration_Type'Alignment use 4;

type Log_Elements_Size_Type is range 0..256;
for Log_Elements_Size'size use 32;
for Log_Elements_Size'Alignment use 4;
subtype Log_Elements_Index_Type is Log_Elements_Size_Type range 0..255;

type Log_Elements_Type is array (Log_Elements_Index_Type) of
  ECOA.Character_8_Type;
for Log_Elements_Type'size use 2048;
for Log_Elements_Type'Alignment use 4;

type Log_Type is
   record
      Current_Size : Log_Elements_Size_Type;
      Data         : Log_Elements_Type;
   end record;
for Log_Type'size use 2080;
for Log_Type'Alignment use 4;

type Module_States_Type is new Unsigned_32_Type;
Module_States_Type_IDLE    : constant Module_States_Type := 0;
Module_States_Type_READY   : constant Module_States_Type := 1;
Module_States_Type_RUNNING : constant Module_States_Type := 2;

type Module_Error_Type is new Unsigned_32_Type;
Module_Error_Type_ERROR       : constant Module_Error_Type := 0;
Module_Error_Type_FATAL_ERROR : constant Module_Error_Type := 1;

type Error_Id_Type is new Unsigned_32_Type;

type Asset_Id_Type is new Unsigned_32_Type;

type Asset_Type is new Unsigned_32_Type;
Asset_Type_COMPONENT         : constant Asset_Type := 0;
Asset_Type_PROTECTION_DOMAIN : constant Asset_Type := 1;
Asset_Type_NODE              : constant Asset_Type := 2;
Asset_Type_PLATFORM          : constant Asset_Type := 3;
Asset_Type_SERVICE           : constant Asset_Type := 4;
Asset_Type_DEPLOYMENT        : constant Asset_Type := 5;

type Error_Type is new Unsigned_32_Type;
Error_Type_RESOURCE_NOT_AVAILABLE : constant Error_Type := 0;
Error_Type_UNAVAILABLE            : constant Error_Type := 1;
```

```
   Error_Type_MEMORY_VIOLATION       : constant Error_Type := 2;
   Error_Type_NUMERICAL_ERROR        : constant Error_Type := 3;
   Error_Type_ILLEGAL_INSTRUCTION    : constant Error_Type := 4;
   Error_Type_STACK_OVERFLOW         : constant Error_Type := 5;
   Error_Type_DEADLINE_VIOLATION     : constant Error_Type := 6;
   Error_Type_OVERFLOW               : constant Error_Type := 7;
   Error_Type_UNDERFLOW              : constant Error_Type := 8;
   Error_Type_ILLEGAL_INPUT_ARGS     : constant Error_Type := 9;
   Error_Type_ILLEGAL_OUTPUT_ARGS    : constant Error_Type := 10;
   Error_Type_ERROR                  : constant Error_Type := 11;
   Error_Type_FATAL_ERROR            : constant Error_Type := 12;
   Error_Type_HARDWARE_FAULT         : constant Error_Type := 13;
   Error_Type_POWER_FAIL             : constant Error_Type := 14;
   Error_Type_COMMUNICATION_ERROR    : constant Error_Type := 15;
   Error_Type_INVALID_CONFIG         : constant Error_Type := 16;
   Error_Type_INITIALISATION_PROBLEM : constant Error_Type := 17;
   Error_Type_CLOCK_UNSYNCHRONIZED   : constant Error_Type := 18;
   Error_Type_UNKNOWN_OPERATION      : constant Error_Type := 19;
   Error_Type_OPERATION_OVERRATED    : constant Error_Type := 20;
   Error_Type_OPERATION_UNDERRATED   : constant Error_Type := 21;

   type Recovery_Action_Type is new Unsigned_32_Type;
   Recovery_Action_Type_SHUTDOWN          : constant Recovery_Action_Type
     := 0;
   Recovery_Action_Type_COLD_RESTART      : constant Recovery_Action_Type
     := 1;
   Recovery_Action_Type_WARM_RESTART      : constant Recovery_Action_Type
     := 2;
   Recovery_Action_Type_CHANGE_DEPLOYMENT : constant Recovery_Action_Type
     := 3;

   type Pinfo_Filename_Elements_Size_Type is range 0..256;
   for Pinfo_Filename_Elements_Size_Type'size use 32;
   for Pinfo_Filename_Elements_Size_Type'Alignment use 4;
   subtype Pinfo_Filename_Elements_Index_Type is
Pinfo_Filename_Elements_Size_Type range 0..255;

   type Pinfo_Filename_Elements_Type is array
     (Pinfo_Filename_Elements_Index_Type) of ECOA.Character_8_Type;
   for Pinfo_Filename_Elements_Type'size use 2048;
   for Pinfo_Filename_Elements_Type'Alignment use 4;

   type Pinfo_Filename_Type is
      record
```

```
         Current_Size : Pinfo_Filename_Elements_Size_Type;
         Data         : Pinfo_Filename_Elements_Type;
      end record;
   for Pinfo_Filename_Type'size use 2080;
   for Pinfo_Filename_Type'Alignment use 4;

   type Seek_Whence_Type is new Unsigned_32_Type;
   Seek_Whence_Type_SEEK_SET : constant Seek_Whence_Type := 0;
   Seek_Whence_Type_SEEK_CUR : constant Seek_Whence_Type := 1;
   Seek_Whence_Type_SEEK_END : constant Seek_Whence_Type := 2;

end ECOA;
```