Mixed Language Programming in Ada

Martyn Pike

university.adacore.com

What is Mixed Language Programming ?

- Large systems are rarely written in a single language
- Need for reuse of low-level routines in C or assembly
- GUI Toolkits might be written in C++ or Java
- Numeric Libraries may be written in FORTRAN
- Legacy libraries may be written in COBOL
- Newer code may be written in Python
- Code from different languages is foreign to each other
- Careful consideration when combining foreign modules

Importing C Subprograms

```
with Interfaces.C; with Ada.Text_IO;
procedure Main is

   function Get_Length return Interfaces.C.size_t with
      Convention    => C,
      Import        => True,
      External_Name => "getLength";

begin
   Ada.Text_IO.Put_Line(Get_Length'Img);
end Main;
```

```
#include <stdlib.h>

extern size_t getLength(void)
{
   return (size_t)10;
}
```

Details of Mixed Language Programming

- **Compilers output Object Code.**
- **Compiler can use its own policies to do this.**
- **An example policy is subprogram Calling Convention.**
- **Another is object memory layout.**
- **Calling convention can be language or compiler specific.**
- **Linkers aren't interested in the calling convention used in object code.**
- **Programmer is left to correctly reference symbols**
- **Ada has features to ease mixed language programming**

Features of Ada for Mixed Language Programming

- **Interfacing to foreign languages is part of the Ada Language Standard**
  - Ada 83, Ada95 and Ada 2005 uses pragmas

```
with Interfaces;
package Test is

   procedure Proc1(In_Param : in Interfaces.Unsigned_8);
   pragma Import(C, Proc1);

end Test;
```

  - Ada 2012 can also use aspects

```
with Interfaces;
package Test is

   procedure Proc2(In_Param : in Interfaces.Unsigned_8);
   pragma Import(C, Proc2);

   procedure Proc1(In_Param : in Interfaces.Unsigned_8) with
     Import     => True,
     Convention => C;

end Test;
```

Aspects - Import

- **Applicable to subprograms and entities**

```
with Interfaces;
package Test is

   procedure Proc1(In_Param : in Interfaces.Unsigned_8) with
     Import     => True,
     Convention => C;

   A_Byte : Interfaces.Unsigned_8 with
     Import      => True,
     Convention  => C;

   function Func1 return Interfaces.Unsigned_8 with
     Import     => True,
     Convention => C;

end Test;
```

- **An Ada program is not responsible for elaboration of Imported entities. Explicit initialisation is illegal.**

```
A_Byte : Interfaces.Unsigned_8 := 0 with
  Import      => True,
  Convention  => C;
```

Aspects - Export

- Similar rules to the Import aspect
- Bodies must be provided for exported subprograms
- Explicit initialisation of exported variables is legal

```
with Interfaces; use Interfaces;
package Test is

   procedure Proc1(In_Param : in Unsigned_8) with
     Export     => True,
     Convention => C;

   A_Byte : Unsigned_8 := 0 with
     Export     => True,
     Convention => C;

   function Func1 return Unsigned_8 with
     Export     => True,
     Convention => C;

end Test;
```

```
package body Test is

   procedure Proc1(In_Param : in Unsigned_8) is
   begin
      null;
   end Proc1;

   function Func1 return Unsigned_8 is
   begin
      return Unsigned_8'First;
   end Func1;

end Test;
```

Packages - Interfaces

- **Size constrained types**
  - Integer_8, Integer_16, Integer_32 and Integer_64
  - Unsigned_8, Unsigned_16, Unsigned_32 and Unsigned_64
  - IEEE_Float_32, IEEE_Float_64 and IEEE_Extended_Float
- **Bit Wise Operations as Subprograms**
  - Shift_Left, Shift_Right, Shift_Right_Arithmetic
  - Rotate_Left, Rotate_Right
  - Defined for all Unsigned_xx types
- **Essential for interfacing to external hardware**
- **Child packages of Interface are available for C, C++, Fortran and COBOL**

Aspects - Convention

- **Fortran**

```
with Interfaces.Fortran; use Interfaces.Fortran;

package Test is

   type Fortran_Matrix is array (Integer range <>,
                                 Integer range <>) of Double_Precision
      with Convention => Fortran;                    -- stored in Fortran's
                                                     -- column-major order
end Test;
```

- **COBOL**

```
with Interfaces.COBOL; use Interfaces.COBOL;

package Test is

   type COBOL_Record is
      record
         Name   : Numeric(1..20);
         SSN    : Numeric(1..9);
         Salary : Binary;  -- Assume Binary = 32 bits
      end record
      with Convention => COBOL;

   procedure Prog(Item : in out COBOL_Record) with
      Import     => True,
      Convention => COBOL;

end Test;
```

Aspects - External_Name

- **Aspect value is of string type**
- **Name of the entity as seen by the foreign language**
- **Can be applied to Imported and Exported entities**
- **Useful for renaming entities**
  - Workaround for different casing conventions
- **Provides a thin veneer if required**

```
with Interfaces;
package Test is

   A_Byte : Interfaces.Unsigned_8 with
     Convention    => C,
     Import        => True,
     External_Name => "byte_for_ada";

end Test;
```

Aspects - Link_Name

- Aspect value is of string type
- Symbol Name of the entity as it appears in the foreign languages object code symbol table
- Introduces a compiler specific identifier
- Incorrect link name may not show as an error until link time

```
with Interfaces;
package Test is

   A_Byte : Interfaces.Unsigned_8 with
     Convention => C,
     Import     => True,
     Link_Name  => "__byte_for_ada";

end Test;
```

Quiz

Is this correct? (1/10)

YES
(click on the check icon)

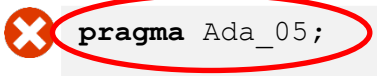
NO
(click on the error location(s))

```
pragma Ada_05;

with Interfaces;

package Test is

   A_Byte : Interfaces.Unsigned_8 with
     Convention    => C,
     Import        => True,
     External_Name => "byte_for_ada";

end Test;
```

Is this correct? (1/10)

NO

Ada 2005 does not support Aspect notation



```
pragma Ada_05;

with Interfaces;

package Test is

   A_Byte : Interfaces.Unsigned_8 with
     Convention    => C,
     Import        => True,
     External_Name => "byte_for_ada";

end Test;
```

Correct Ada 2005 code

```
pragma Ada_05;

with Interfaces;

package Test is

   A_Byte : Interfaces.Unsigned_8;
   pragma Import(
     Convention    => C,
     Entity        => A_Byte,
     External_Name => "byte_for_ada"
   );

end Test;
```

Is this correct? (2/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
package Test is

   procedure Proc1 with
     Export     => True,
     Convention => C;

   procedure Proc2 with
     Import     => True,
     Convention => C;

end Test;
```

```
package body Test is

   procedure Proc1 is
   begin
     null;
   end Proc1;

end Test;
```

Is this correct? (2/10)

YES

```
package Test is

   procedure Proc1 with
     Export     => True,
     Convention => C;

   procedure Proc2 with
     Import     => True,
     Convention => C;

end Test;
```

```
package body Test is

   procedure Proc1 is
   begin
     null;
   end Proc1;

end Test;
```

Is this correct? (3/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
with Interfaces;

procedure Main is

  A_Byte : Interfaces.Unsigned_8 with
     Convention    => C,
     Import        => True,
     External_Name => "byte_for_ada";

begin

  byte_for_ada := Interfaces.Unsigned_8'Last;

end Main;
```

Is this correct? (3/10)

NO

Incorrect use of foreign language entity name and not the Ada entity name

```
with Interfaces;

procedure Main is

  A_Byte : Interfaces.Unsigned_8 with
     Convention    => C,
     Import        => True,
     External_Name => "byte_for_ada";

begin

  byte_for_ada := Interfaces.Unsigned_8'Last;

end Main;
```

Is this correct? (4/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
with Interfaces; use Interfaces;

procedure Main is

  A_Byte : constant Unsigned_8 := 0 with
     Convention    => C,
     Import        => True;

  B_Byte : Unsigned_8;

begin

  B_Byte := A_Byte;

end Main;
```

Is this correct? (4/10)

NO

Imported entities cannot be initialised

```
with Interfaces; use Interfaces;

procedure Main is

  A_Byte : constant Unsigned_8 := 0 with
     Convention    => C,
     Import        => True;

  B_Byte : Unsigned_8;

begin

  B_Byte := A_Byte;

end Main;
```

Is this correct? (5/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
with Interfaces; use Interfaces;

procedure Main is

  A_Byte : Unsigned_8 with
     Convention => C,
     Import     => True,
     Link_Name  => a_byte_for_ada;

begin

  null;

end Main;
```

Is this correct? (5/10)

NO

Link_Name aspect is of string type

```
with Interfaces; use Interfaces;

procedure Main is

  A_Byte : Unsigned_8 with
     Convention => C,
     Import     => True,
     Link_Name  => a_byte_for_ada;

begin

  null;

end Main;
```

(6/10)

If both External_Name and Link_Name aspects are specified for a given entity, then the Link_Name is ignored.

- A) True
- B) False

Submit

You must answer the question before continuing.

<< >>

(6/10)

The answer is FALSE

If both External_Name and Link_Name aspects are specified for a given entity, then the External_Name is ignored.

Text Caption (650x209)

(7/10)

It's OK to use standard Ada types when passing parameters into imported or exported subprograms or returning values from imported or exported functions

- A) True
- B) False

Image (960x720): Q7.PNG

Submit

You must answer the question before continuing.

<< >>

(7/10)

The answer is FALSE

To guarantee correct operation you must use the types in the Interfaces package or one of its child packages.

Image (960x720): A7.PNG

(8/10)

It's legal to use the interfacing pragmas in Ada 2012 source code

- A) True
- B) False

Submit

Image (960x720): Q8.PNG

nswer the question before continuing.

<< >>

(8/10)

The answer was TRUE

It is possible to still use the interfacing pragmas in Ada 2012 source code, however their use is deprecated and it's recommended to move to using aspects

Is this correct? (9/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
with Interfaces; use Interfaces;

procedure Main is

  A_Byte : constant Unsigned_8 := 0 with
     Convention    => C,
     Export        => True;

begin

  null;

end Main;
```

Is this correct? (9/10)


```
with Interfaces; use Interfaces;

procedure Main is

  A_Byte : constant Unsigned_8 := 0 with
     Convention    => C,
     Export        => True;

begin

  null;

end Main;
```

(10/10)

```
with Interfaces.C; with Ada.Text_IO;
procedure Main is

   function Get_Length return Interfaces.C.size_t with
      Convention    => C,
      Import        => True,
      External_Name => "getLength";

begin
   Ada.Text_IO.Put_Line(Get_Length'Img);
end Main;
```

```
#include <stdlib.h>

extern size_t getLength(void)
{
   return (size_t)10;
}
```

(10/10)

```
with Interfaces.C; with Ada.Text_IO;
procedure Main is

   function Get_Length return Interfaces.C.size_t with
      Convention    => C,
      Import        => True,
      External_Name => "getLength";

begin
   Ada.Text_IO.Put_Line(Get_Length'Img);
end Main;
```

```
#include <stdlib.h>

extern size_t getLength(void)
{
   return (size_t)10;
}
```

university.adacore.com