What has the ARG been up to?

Recent and future changes to Ada 2012

Ada 2012 Corrigendum

Will be able to use

```
X'Image
```

rather than the cumbersome

```
X_Type'Image (X)
```

Same as GNAT's 'Img, but as a vendor extension it wasn't allowed to re-use the word Image.

Subtype Predicates (Static)

- Posh form of constraint, the sub-range doesn’t have to be contiguous.
- If the Assertion_Policy is Check and the run-time check fails, an Assertion_Error exception is raised.

```
pragma Assertion_Policy (Static_Predicate => Check);

type Day_Of_Week is (Monday, Tuesday, Wednesday, Thursday, Friday,
                     Saturday, Sunday);
subtype Weekend is Day_Of_Week range Saturday .. Sunday;
subtype Milk_Delivery_Day is Day_Of_Week
  with Static_Predicate =>
    Milk_Delivery_Day in Monday | Wednesday .. Thursday | Weekend;

procedure Order_Milk (Milk_Day : Milk_Delivery_Day);
```

Subtype Predicates (Dynamic - 1)

```
with Ada.Text_IO;

package My_Text_IO is

   pragma Assertion_Policy (Dynamic_Predicate => Check);

   use type Ada.Text_IO.File_Mode;
   use type Ada.Text_IO.File_Type;

   type File_Type is private;

   subtype Open_File_Type is File_Type
      with Dynamic_Predicate => (Is_Open (Open_File_Type));
```

BAE SYSTEMS

Subtype Predicates (Dynamic - 2)

```
subtype Input_File_Type is Open_File_Type
   with Dynamic_Predicate => Mode (Input_File_Type) =
                              Ada.Text_IO.In_File;

subtype Output_File_Type is Open_File_Type
   with Dynamic_Predicate => Mode (Output_File_Type) /=
                              Ada.Text_IO.In_File;
```

Subtype Predicates (Dynamic - 3)

```
   function Is_Open (Open_File : File_Type) return Boolean;

    function Mode (File : in Open_File_Type) return
     Ada.Text_IO.File_Mode;

    … Close, Open, Get, Put …

private

  type File_Type is new Integer;

end My_Text_IO;
```

Ada 2012 Corrigendum - Subtype Predicates (1)

Can choose which exception to raise when fails, doesn't have to be Assertion_Error. Uses the new aspect "Predicate_Failure".

So in the Milk example, if regarding as a posh constraint rather than a contract (assertion), can raise Constraint_Error instead of Assertion_Error.

```
subtype Milk_Delivery_Day is Day_Of_Week
  with Static_Predicate  =>
    Milk_Delivery_Day in Monday | Wednesday .. Thursday | Weekend,
       Predicate_Failure => raise Constraint_Error;
```

Ada 2012 Corrigendum - Subtype Predicates (2)

And in the My_Text_IO example, can re-use exceptions of pre-defined Text_IO instead of Assertion_Error.

```
subtype Open_File_Type is File_Type
   with Dynamic_Predicate => (Is_Open (Open_File_Type)),
        Predicate_Failure => raise Status_Error;
```

BAE SYSTEMS

Ada 2012 Corrigendum - Subtype Predicates (3)

```
subtype Input_File_Type is Open_File_Type
   with Dynamic_Predicate => Mode (Input_File_Type) =
                                  Ada.Text_IO.In_File,
        Predicate_Failure => raise Mode_Error with
                    "Can't read file: " & Name (Input_File_Type);

subtype Output_File_Type is Open_File_Type
   with Dynamic_Predicate => Mode (Output_File_Type) /=
                                  Ada.Text_IO.In_File,
        Predicate_Failure => raise Mode_Error with
                  "Can't write file: " & Name (Output_File_Type);
```

BAE SYSTEMS

Ada 2012 Corrigendum - Subtype Predicates (4)

- The order of checking is better defined – constraints and null exclusions, then predicates in order of derivation.
- So if the user has specified different exceptions for different predicates, the appropriate one should be raised. In the My_Text_IO example, the appropriate one of Status_Error and Mode_Error.

Ada 2012 Corrigendum - Subtype Predicates (5)

Predicate failure in a membership test now just means that the test returns False, no exception.

```
subtype Winter is Month
   with Static_Predicate => Winter in Dec | Jan | Feb;
M : Month := Mar;

if M in Winter then
   ...  -- do this if M is a winter month
else
   ...  -- do this if M is not a winter month
end if;
```

No exception is raised since this is a membership test and not a predicate check.

Ada 2012 Corrigendum - Containers

- The design of containers assumes that they are not accessed concurrently, on the ground that access can be protected with protected objects if necessary.
- However, containers provide functions that modify the state of the container; it is therefore not possible to access these through protected functions, as these can be called concurrently.
- So aspect "Exclusive_Functions" can now be applied to protected types and single protected objects to forbid concurrent execution of protected functions. E.g.

```
protected type PT
   with Exclusive_Functions;
   …
```

Ada 2012 Corrigendum – Postconditions

- If we requeue from one entry to another, how do we ensure that the postcondition of the first entry are still met? By insisting that the entry requeued to has a matching postcondition.
- But X'Old at the entry to the second entry might differ from X'Old at the entry to the first entry, so the postconditions cannot use 'Old.
- The details of the constants created for 'Old clarified. In particular, X'Old has the same type as X even if it's anonymous.

BAE SYSTEMS

Ada 2012 Corrigendum – Multiprocessors

- Dispatching domains can now have sets of CPUs besides ranges. Hardware designs don't necessarily group CPUs with contiguous CPU numbers.

```
type CPU_Set is array (CPU range <>) of Boolean;
function Create (Set : CPU_Set) return Dispatching_Domain;
function Get_CPU_Set (Domain : Dispatching_Domain) return CPU_Set;
```

- Extra Restrictions for Ravenscar:
  - No_Tasks_Unassigned_To_CPU
  - No_Dynamic_CPU_Assignment

- Also No_Dependence => Ada.Synchronous_Barriers

Ada 2012 Corrigendum – Type Invariants (1)

- Remember that type invariants do not guarantee that they’re never violated, but add checks at certain points, such as return from subprogram.
- For in parameters, they are only checked for procedures not functions, to avoid infinite recursion if the invariant itself calls a function with a parameter of the type, and to make class-wide invariants possible.

BAE SYSTEMS

Ada 2012 Corrigendum – Type Invariants (2)

- Invariants now also have to be checked on the initialisation of deferred constants, e.g.:

```
package R is
   type T is private
     with Type_Invariant => Non_Zero (T);
   function Non_Zero (X : T) return Boolean;
   Zero : constant T;
private
   type T is new Integer;
   function Non_Zero (X : T) return Boolean is
      (X /= 0);
   Zero : constant T := 0;
end R;
```

Ada 2012 Corrigendum – Type Invariants (3)

- Interface types can now have class-wide type invariants, e.g.:

```
type Window is interface
  with Type_Invariant'Class => Window.Width * Window.Height = 100;

function Width  (W : Window) return Integer is abstract;
function Height (W : Window) return Integer is abstract;
```

BAE SYSTEMS

Ada 2012 Corrigendum – Storage Pools

- Ada 2012's

  ```
  pragma Default_Storage_Pool (My_Pool);
  ```

  says which storage pool to use by default if not specified explicitly when an access type is declared, and

  ```
  pragma Default_Storage_Pool (null);
  ```

  says that there is no default, the storage pool must be specified explicitly when an access type is declared.

  The Corrigendum adds

  ```
  pragma Default_Storage_Pool (Standard);
  ```

  to revert the default to the standard storage pool.

Ada 2012 Corrigendum – Stream-oriented Attributes

- Stream-oriented attributes can now be specified using aspect specs, e.g. instead of

```
for Date’Write use Date_Write;
```

we can write

```
type Date is record … end record
   with Write => Date_Write;
```

and similarly for a class-wide stream-oriented aspect

```
type My_Type is abstract tagged null record
   with Write’Class => Date_Write;
```

Ada 2012 Corrigendum – Generalized iterators (1)

- In Ada 2012, for an array of elements of type T

```
type ATT is array (1..N) of T;
The_Array : ATT;
```

we can simplify

```
for I in The_Array’Range loop
   The_Array (I) := 99;
end loop;
```

to

Ada 2012 Corrigendum – Generalized iterators (2)

```
for E of The_Array loop
   E := 99;
end loop;
```

or

```
for E : T of The_Array loop
   E := 99;
end loop;
```

where the optional “: T” acts as a comment to the reader that the subtype of E is T.

The corrigendum adds the obvious requirement that the subtype given has to be correct.

Ada 2012 Corrigendum - Expression functions

- Expression functions and null procedures can now also be used in protected bodies.
- Expression functions and null procedures that act as completions to traditional subprogram specs cannot have pre- or postconditions.

BAE SYSTEMS

Ada 2012 Corrigendum - Quantified expressions

What if the array is empty?

```
A := (for some I in X'Range => X (I) = 0);
```

This assigns True if any element is 0. If the array doesn't have any elements, it assigns False, as one might expect.

```
A := (for all I in X'Range => X (I) = 0);
```

This assigns True if all elements are 0. If the array doesn't have any elements, it still assigns True.

Beware that in formal logic, expressions such as "all my children are green" are true if you have no children.

BAE SYSTEMS

Ada 2012 Corrigendum – Class-wide pre and post (1)

```
package Parent_Pkg is
   type T is …;
   function Primitive (X : T) return Integer
    with Pre’Class => Another_Primitive (X) > 0;
…

package Parent_Pkg.Child_Pkg is
   type NT is new T with …;
   overriding
   function Primitive (Y : NT) return Integer
    with Pre’Class => Another_Primitive (Y) < 10;
…

C : Parent_Pkg.Child_Pkg.NT := …;
```

BAE SYSTEMS

Ada 2012 Corrigendum – Class-wide pre and post (2)

```
function F (O : in Parent_Pkg.T’Class) return Integer is
   (Parent_Pkg.Primitive (O));

Result := F (C);
```

This dispatches to Parent_Pkg.Child_Pkg.Primitive.

Is the class-wide precondition checked that of Parent_Pkg.Primitive or that of Parent_Pkg.Child_Pkg.Primitive? (The latter may be weaker as it is an ‘or’ of its Pre’Class and its parent’s Pre’Class).

The former.

Ada 2012 Corrigendum – Class-wide pre and post (3)

```
function F (O : in Parent_Pkg.T'Class) return Integer is
  (O.Primitive);

Result := F (C);
```

This dispatches to Parent_Pkg.Child_Pkg.Primitive

In Primitive's class-wide precondition, does the call of Another_Primitive invoke Parent_Pkg.Another_Primitive, or Parent_Pkg.Child_Pkg.Another_Primitive?

The latter.

Similarly for postconditions.

Ada 2012 Corrigendum – Class-wide pre and post (4)

```
Result := Parent_Pkg.Primitive (Parent_Pkg.T (C));
```

This is a statically bound call to Parent_Pkg.Primitive, but with a parameter with a tag for a type declared in Parent_Pkg.Child_Pkg.

In Primitive's class-wide precondition, is the call of Another_Primitive statically bound to Parent_Pkg.Another_Primitive, or does it dispatch to Parent_Pkg.Child_Pkg.Another_Primitive?

The former. The tag doesn't matter.

Similarly for postconditions.

ACATS

- ACATS is the Ada Conformity Assessment Test Suite.
- Whereas Ada 95 had about 80% coverage, by 2013 the new features of Ada 2005 only had 30%.
- ACATS 3.1, released in April 2014, endeavoured to catch up for Ada 2005, e.g. tests for Containers. 12 updates since then.
- ACATS 4.0, released in August 2014, covers the most important features for Ada 2012. 11 updates since then.

BAE SYSTEMS

Ideas for Ada 202X (1)

- Global aspects. Already in GNAT for SPARK 2014 support:

```
procedure Include_Piece_In_Board
  with Global => (Input => Cur_Piece, In_Out => Cur_Board);
```

This specifies which global objects a subprogram may access, and in which mode.

- Pragma Loop_Invariant. Already in GNAT for SPARK 2014 support:

```
for ... loop
   Some_Complex_Calculation;
   pragma Loop_Invariant (Calculation_Is_Converging);
end loop;
```

Checked each time round loop.

Ideas for Ada 202X (2)

- Index parameters in array aggregates:

```
subtype Index is Positive range 1 .. 10;
type Array_Type is array (Index) of Positive;
Squares_Array : Array_Type := (for I in Index => I * I);
```

- Attribute ‘Object_Size:

S'Object_Size denotes the size of an object of subtype S. Reading ‘Size is not terribly useful as it just gives the theoretical minimum number of bits required for a value of a given range, not the number of bits that the compiler is actually going to allocate to an object of the type. Specifying ‘Size just gives a minimum, the compiler may allocate more.

Ideas for Ada 202X (3)

- Placeholder for the left hand side.

```
My_Package.My_Array(I).Field := My_Package.My_Array(I).Field + 1;
```

could be shortened to

```
My_Package.My_Array(I).Field := @ + 1;
```

or even

```
My_Package.My_Array(I).Field := My_Package.My_Array(I).Field ** 3
+ My_Package.My_Array(I).Field ** 2 +
My_Package.My_Array(I).Field;
```

to

```
My_Package.My_Array(I).Field := @ ** 3 + @ ** 2 + @;
```

- In the further future : Parallelism, study led by Canada. See OpenMP for what has been done for C, C++ and Fortran.

BAE SYSTEMS

Ideas for Ada 202X (4)

- Other suggestions from AdaCore (not yet officially raised):
  - Lambda functions
  - Generators/co-routines
  - Function decorators
  - Declare variable in expression constructs

Ideas for Ada 202X (5)

- But aim to constrain the growth:
    - Try to put any new features in Optional annexes.
    - More Restrictions, e.g. no anonymous access types.