Coding Standard for Ada

● Code Layout

Note that in order to force a certain layout, the programmer can insert an end-of-line, or line break that will not be removed by the formatter by entering <space> <space> <carriage-return>.

Lists of Ada elements should be broken to contain only one element per line, when the list exceeds 3 items, and when they do not fit on one line.

Ex)

argument association

```
pragma Suppress (Range_Check,
                 On => This_Type,
                 On => That_Type,                 On => That_Other_Type);
```

identifier list, component list

```
Next_Position,
Previous_Position,
Current_Position : Position;
type Some_Record is
    record
        A_Component,
        B_Component,
        C_Component : Component_Type;
    end record;
```

enumeration type definition

```
type Navaid is
       (Vor,
        Vor_Dme,
        Dme,
        Tacan,
        Vor_Tac,
        NDB);
```

discriminant constraint

```
subtype Constrained is Element
        (Name_Length    => Name'Length,
         Valid          => True,
         Operation      => Skip);
```

sequence of statements (done by formatter)

formal part, generic formal part, actual parameter part, generic actual parameter part

```
procedure Just_Do_It (This     : in Some_Type;
                      For_That : in Some Other_Type;
                      Status   : out Status_Type);
Just_Do_It (This     => This_Value;
            For_That => That_Value;
            Status   => The_Status);
```

Naming Conventions

General

Choose clear, legible, meaningful names : Unlike many other programming languages, Ada does not limit the length of identifiers to 6, 8, or 15 characters

Separate various words of a name by an underscore:

Ex) Is_Name_Valid rather than IsNameValid

Use full names rather than abbreviations.

Do not use as identifiers the words: **abstract**, **aliased**,**protected**, **requeue**, **tagged** and **until**, which will become keywords in Ada 95.

Packages

When a package introduces some object class, give it the name of the object class, usually a common noun in singular form, with the suffix _Generic if necessary

```
package Text is
package Line is
package Mailbox is
package Message is
package Attributes is
package Subscriber is
package List_Generic is
```

When a package specifies an interface or some grouping of functionality, and does not relate to an object, express this in the name

```
package Low_Layer_Interface is
package Math_Definitions is
```

Exceptions

Since exceptions must be used only to handle error situations, use a noun or a noun phrase that clearly conveys a negative idea:

Ex) Overflow, Threshold_Exceeded, Bad_Initial_Value

Subprograms

Use verbs for procedures (and task entries). Use nouns with the attributes or characteristics of the object class for functions. Use adjectives (or past participles) for functions returning a Boolean (predicates).

```
Subscriber.Create
Subscriber.Destroy
Subscriber.List.Append
Subscriber.First_Name          -- Returns a string.
Subscriber.Creation_Date       -- Returns a date.
Subscriber.List.Next
Subscriber.Deleted             -- Returns a Boolean.
Subscriber.Unavailable         -- Returns a Boolean.
Subscriber.Remote
```

For predicates, it may be useful in some cases to add the prefix Is_ or Has_ before a noun; be accurate and consistent with respect to tense:

```
 function Has_First_Name ...
 function Is_Administrator ...
 function Is_First...
 function Was_Deleted ...
```

✓ Objects and Subprogram parameters

To indicate uniqueness, or to show that this entity is the main focus of the action, prefix the object or parameter name with The_ or This_. To indicate a side, temporary, auxiliary object, prefix it with A_ or Current_:

```
procedure Change_Name (The_Subscriber : in Subscriber.Handle;
                       The_Name       : in Subscriber.Name );
declare
    A_Subscriber : Subscriber.Handle := Subscriber.First;
begin
    ...
    A_Subscriber := Subscriber.Next (The_Subscriber);
end;
```

For Boolean objects, use a predicate clause, with the positive form:

Ex) Found_It , Is_Available

● Expressions and Statements

✓ Expressions

Record aggregates should use named associations and should be qualified:

```
Subscriber.Descriptor'(Name    => Subscriber.Null_Name,
                       Mailbox => Mailbox.Nil,
                       Status  => Subscriber.Unknown,
                   ...);
```

Use simple Boolean expressions in place of "if…then…else" statements for simple predicates:

```
function Is_In_Range(The_Value: Value; The_Range: Range)
     return Boolean is
begin
    return The_Value >= The_Range.Min
        and The_Value <= The_Range.Max;
end Is_In_Range;
```

✓ **Statements**

Loop statements should have names:

① When they extend over more than 25 lines

② When they are nested

③ When there is a meaningful name to designate what they perform

④ When the loop has no end :

Ex) Forever: loop

        …

  end loop Forever;

Subprograms should have a single point of return :

Try to exit from subprograms at the end of the statement part. Functions should have a single return statement. Return statements sprinkled freely over a function body are akin to *goto* statements, making the code difficult to read and to maintain.