AWS Coding Style

Pascal Obry.

1 General

This document described the style rules for the development of the AWS project. The goal is to have a consistent style used for all AWS codes.

2 Ada 2005

As the Ada 2005 support on GNAT is maturing, it is possible to use some Ada 2005 constructs for AWS development. We list here the features that can be used:

- Ada.Containers
- raise .. with "";
- object.method notation
- limited with
- anonymous access fields/parameters
- use of overriding keyword

Constructs that are not ready for use:

- interfaces
- extended return statement

In addition, all constructs should be compatible with GNAT 6.2 and GPL 2009.

3 Lexical Elements

———————————————————————————-

3.1 Character Set and Separators

- The character set used should be plain 7-bit ASCII. The only separators allowed are space and the end-of-line sequence. No other control character or format effector (such as HT, VT, FF) should be used.

  The end-of-line sequence used must be the standard UNIX end-of-line character, a single LF (16#0A#).
- A line should never be longer than 79 characters, not counting the line separator.
- Lines must not have trailing blanks.
- Indentation is 3 characters per level for if-statements, loops, case statements.

3.2 Identifiers

- Identifiers will start with an upper case letter, and each letter following an underscore will be upper case. Short acronyms may be all upper case. All other letters are lower case. An exception is for identifiers matching a foreign language. In particular, we use all lower case where appropriate for C.
- Use underscores to separate words in an identifier.

- Try to limit your use of abbreviations in identifiers. It is ok to make a few abbreviations, explain what they mean, and then use them frequently, but don't use lots of obscure abbreviations.

- Don't use the variable I, use J instead, I is too easily mixed up with 1 in some fonts. Similarly don't use the variable O, which is too easily mixed up with zero.

3.3 Numeric Literals

- Numeric literals should include underscores where helpful for readability.

```
1_000_000
16#8000_000#
3.14159_26535_89793_23846
```

3.4 Reserved Words

- Reserved words use all lower case.

```
return else procedure
```

- The words **"Access"**, **"Delta"** and **"Digits"** are capitalized when used as attribute_designator.

3.5 Comments

- Comment start with `--` (ie `--` followed by two spaces). The only exception to this rule (i.e. one space is tolerated) is when the comment ends with `--`. It also accepted to have only one space between `--` and the start of the comment when the comment is at the end of a line, after an Ada statement.

- Every sentence in a comment should start with an upper-case letter (including the first letter of the comment).

- When declarations are commented with "hanging" comments, i.e. comments after the declaration, there is no blank line before the comment, and if it is absolutely necessary to have blank lines within the comments these blank lines *do* have a – (unlike the normal rule, which is to use entirely blank lines for separating comment paragraphs). The comment start at same level of indentation as code they are commenting.

```
Z : Integer;
--  Integer value for storing value of Z
--
--  The previous line was a blank line
```

- Comments that are dubious or incomplete or comment on possibly wrong or incomplete code should be preceded or followed by ???

- Comments in a subprogram body must generally be surrounded by blank lines, except after a **"begin"**:

```
begin
   --  Comment for the next statement

   A := 5;

   --  Comment for the B statement

   B := 6;
```

- In sequences of statements, comments at the end of the lines should be aligned.

```
My_Identifier := 5;       -- First comment
Other_Id := 6;            -- Second comment
```

- Short comments that fit on a single line are NOT ended with a period. Comments taking more than a line are punctuated in the normal manner.

- Comments should focus on why instead of what. Descriptions of what subprograms do go with the specification.

- Comments describing a subprogram spec should specifically mention the formal argument names. General rule: write a comment that does not depend on the names of things. The names are supplementary, not sufficient, as comments.

- Do NOT put two spaces after periods in comments.

4 Declarations and Types

- In entity declarations, colons must be surrounded by spaces. Colons should be aligned.

```
Entity1   : Integer;
My_Entity : Integer;
```

- Declarations should be grouped in a logical order. Related groups of declarations may be preceded by a header comment.

- All local subprograms in a subprogram or package body should be declared before the first local subprogram body.

- Avoid declaring discriminated record types where the discriminant is used for constraining an unconstrained array type. (Discriminated records for a variant part are allowed.)

- Avoid declaring local entities that hide global entities.

- Don't declare multiple variables in one declaration that spans lines. Start a new declaration on each line, instead

- The defining_identifiers of global declarations serve as comments of a sort. So don't choose terse names, but look for names that give useful information instead.

- Local names can be shorter, because they are used only within one context, where comments explain their purpose.

5 Expressions and Names

- Every operator must be surrounded by spaces, except for the exponentiation operator.

```
E := A * B**2 + 3 * (C - D);
```

- When folding a long line, fold before an operator, not after.

- Use parentheses where they clarify the intended association of operands with operators:

```
(A / B) * C
```

6 Statements

6.1 Simple and Compound Statements

- Use only one statement or label per line.
- A longer sequence_of_statements may be divided in logical groups or separated from surrounding code using a blank line.
- Prefer using "/=" to "not =" except in complex expression if it makes the expression easier to read or in "well-known" expressions for whose the reverse must be checked.

6.2 If Statements

- When the **"if"**, **"elsif"** or **"else"** keywords fit on the same line with the condition and the **"then"** keyword, then the statement is formatted as follows:

```
if <condition> then
   ...
elsif <condition> then
   ...
else
   ...
end if;
```

When the above layout is not possible, **"then"** should be aligned with **"if"**, and conditions should preferably be split before an **"and"** or **"or"** keyword a follows:

```
if <long_condition_that_has_to_be_split>
  and then <continued_on_the_next_line>
then
   ...
end if;
```

The **"elsif"**, **"else"** and **"end if"** always line up with the **"if"** keyword. The preferred location for splitting the line is before **"and"** or **"or"**. The continuation of a condition is indented with two spaces or as many as needed to make nesting clear.

```
if x = lakdsjfhlkashfdlkflkdsalkhfsalkdhflkjdsahf
     or else
   x = asldkjhalkdsjfhhfd
     or else
   x = asdfadsfadsf
then
```

- Conditions should use short-circuit forms (**"and then"**, **"or else"**).
- Complex conditions in if-statements are indented two characters:

```
if this_complex_condition
  and then that_other_one
  and then one_last_one
then
    ...
```

- Every **"if"** block is preceded and followed by a blank line, except where it begins or ends a sequence_of_statements.

```
A := 5;

if A = 5 then
    null;
```

```
        end if;

        A := 6;
```

6.3 Case statements

- Layout is as below.

```
case <expression> is
   when <condition> =>
      ...
   when <condition> =>
      ...
end case;
```

If the condition and the code for the case section is small, it is possible to put the code for each when section right after the condition without a new-line.

```
case <expression> is
   when <condition> => ...
   when <condition> => ...
end case;
```

6.4 Loop statements

When possible, have **"for"** or **"while"** on one line with the condition and the **"loop"** keyword.

```
for J in S'Range loop
   ...
end loop;
```

If the condition is too long, split the condition (see if_statement) and align **"loop"** with the **"for"** or **"while"** keyword.

```
while <long_condition_that_has_to_be_split>
  and then <continued_on_the_next_line>
loop
   ...
end loop;
```

If the loop_statement has an identifier, it is layout as follows:

```
Outer : while not <condition> loop
   ...
end Outer;
```

6.5 Block Statements

- The (optional) **"declare"**, **"begin"** and **"end"** statements are aligned, except when the block_statement is named:

```
Some_Block : declare
   ...
begin
   ...
end Some_Block;
```

7 Subprograms

7.1 Subprogram Declarations

- Never write the **"in"** for parameters.

  function `Length (S : String)` **return** `Integer;`

- The mode should be indented as follow

  procedure `My_Proc`
  `(First  : Integer;`
  `Second :` **out** `Character;`
  `Third  :` **access** `String;`
  `Fourth :` **in out** `Float);`

- When the declaration line for a procedure or a function is too long, fold it

  function `Head`
  `(Source : String;`
  `Count  : Natural;`
  `Pad    : Character := Space)`
  return `String;`

- For function an alternate style is to put the **return** at the end of the last declaration line

  function `Head`
  `(Source : String;`
  `Count  : Natural;`
  `Pad    : Character := Space)` **return** `String;`

- The parameter list for a subprogram is preceded by a space

  procedure `Func (A :` **in out** `Integer);`

7.2 Subprogram Bodies

- The functions and procedures should always be sorted alphabetically in a compilation unit.

- All subprograms have a header giving the function name, with the following format:

  `-----------------`
  `-- My_Function --`
  `-----------------`

  procedure `My_Function` **is**
  begin

  Note that the name in the header is preceded by a single space, not two spaces as for other comments.

- If the subprogram parameters are on multiple lines and there is some declaration the **"is"** must be on a separate line.

  procedure `My_Function (X : Integer)` **is**
  `X : Float;`
  begin

  procedure `My_Function`
  `(X : Integer;`
  `Y : Float)`
  is

```
        A : Character;
   begin
```

- Every subprogram body must have a preceding subprogram_declaration.

- If declarations of a subprogram contain at least one nested subprogram body, then just before the begin is a line:

```
   --  Start of processing for bla bla

   begin
```

- Unchecked_Deallocation instances must be named Unchecked_Free.

8 Packages and Visibility Rules

- All program units and subprograms have their name at the end:

```
   package P is
      ...
   end P;
```

- Avoid "use-ing" the with-ed packages except when it has been designed for. A common example is Ada.Strings.Unbounded where the type is named Unbounded_String. This unit is clearly designed to be use-ed. To ease readability a use clause may be used in a small scope. Another solution is to use renaming. Do not with two times the same unit, always use the deepest child unit to with. For example do not write:

```
   with Ada.Strings;
   with Ada.Strings.Unbounded;
```

but the equivalent form:

```
   with Ada.Strings.Unbounded;
```

- Names declared in the visible part of packages should be unique, to prevent name clashes when the packages are "use"d.

```
   package Entity is
      type Entity_Kind is ...;
      ...
   end Entity;
```

- After the file header comment, the context clause and unit specification should be the first thing in a program_unit.

- try grouping the context clauses

   It is good to group the context clauses in 3 parts. The Ada standard clauses, the components from other projects and then the project's clauses. In each group it is required to sort the clauses by alphabetical order.

```
   with Ada.Exceptions;
   with Ada.Strings;

   with Lists;
   with Ordered_Set;

   with AWS.Server;
   with AWS.URL;
```

9 Program Structure and Compilation Issues

- Every AWS source file must be compiled with the **"-gnatwcfijmpruv -gnatwe -gnaty3abcefhiklmnoprst"** switches to check the coding style.
- Each source file should contain only one compilation unit.
- Body filename should end with ".adb" and spec with ".ads".