RSL Conventions

1. Object names should be nouns or noun phrases (where phrases, of course, have no spaces between words).
2. Operation names should be verbs or verb phrases.
3. Object and operation names should start with uppercase letters, have an uppercase letter at the beginning of each word in the name, and otherwise be lowercase. For example, `Artifact`, `TextArtifact` are conventional object names `artifact`, `ARTIFACT`, `textArtifact`, and `text_artifact` are ***not*** conventional.
4. Object and operation names should trace as directly as possible to names of corresponding elements of the user interface.
   a. Names should be spelled as close as possible to how they appear on the screen, modulo conventions 1, 2, and 3 above.
   b. For example:
      i. A UI button named "Press me" could trace to an operation named "PressMe"; here the two words have merged into 1, and the "M" has been capitalized.
      ii. A dialog labeled "Enter values for a new information record" could trace to an object named "InfoRecord"; here "information" has been abbreviated, and the two words of the name have been capitalized and merged.
5. Variable names, i.e., the *name* half of a name/type pair, should be all lowercase, with no underscores. The letters of a variable name should be the capitalized letters of the type of which the variable is declared. For example, `a:Artifact`, `ta:TextArtifact` are conventional variable names; `art:Artifact`, `t_a:TextArtifact`, `texta:TextArtifact` are ***not*** conventional.
   a. Where variable name duplication would occur due to different types with the same leading letter, one or more additional disambiguating letters should be used to define unique names. For example

      ```
      op F(id:Identifier, ind:Index, it:Iteration);
      ```

      Note in this example that the name "`in`" cannot be used as a variable name since **in** is a keyword of RSL.
   b. Where variable name duplication would occur due to different types with the same leading letter, one or more additional disambiguating letters should be used to define unique names. For example

      ```
      op F(i1:Identifier, i2:Identifier)
      ```

   c. Where a variable name might have significantly more mnemonic quality if spelled with more than one character, the specifier may use discretion to extend the name. For example,

      ```
      op F(id1:Identifier, id2:Identifier)
      ```

      or

      ```
      obj Date is time:Time and day:Day;
      ```

   d. In general, variable names should be kept short.

6. The use of mixed composition operators and built-in atomic type names should be avoided within composition expressions. For example, the following sequence of definitions is good form

```
obj BlahBlahList is BlahBlah*;
obj BlahBlah is Blah and Blah;
obj Blah is integer;
```

whereas the following semantically equivalent definition is bad form:

```
obj BlahBlahList is (integer and integer)*;     -- Bad form
```

7. An object name of the form "XXXList" connotes a collection with a user-perceivable ordering. An object name of the form "XXXs" (or appropriate plural form of "XXX") connotes a collection for which the order is of no concern to the user. For example,

```
obj Things is Thing*;     (* A collection of Things where order is unimportant
                             to the user. *)
obj ThingList is Thing*;  (* A collection of Things where order is important to
                              * the user. *)
```

8. An operation prefixed with "Is" should output a single boolean value.

9. An operation should be named "GetXXX" iff the (first) input object is a tuple with a component named "XXX", and the (first) output is "XXX". For example,

```
obj Datum is X and Y;
op GetX(Datum)->X;
op GetY(Datum)->Y;
```

10. An operation should be named "Find" iff an input object is a list of "XXX", and the (first) output is "XXX". Typically, "XXX" can be a tuple with a distinguished key field used to (uniquely) identify elements of type "XXX", and this key will be used as the second input to the Find operation. If he key is non-unique, the Find operation should output an "XXXList" or "XXXs".

```
obj DataList is Datum*;
obj Datum is k:Key and v:Value;
op Find(DataList, Key)->Datum;
```

11. Short-form versus long-form definitions, plus other forms of keyword abbreviation should be used consistently throughout a given specification. For example, do not use

```
obj X < Parent is A and B and C;
```

and

```
object Y instance of Parent is
    components: D and E and F;
end Y;
```

in the same specification.

12. Indentation and commenting conventions for object and operation definitions within a module are shown in Figure 1.

13. The "--" form of comments should be limited to comments of one line or less in length.

```
(*
 * Put descriptive comment for the module in a comment of this form.
 *)

module M;

  object Name is ...
--^ Indent object and operation keywords 2 spaces

    description: (*
----^ indent attribute names 4 spaces

        This is the description ... .  It should be longer for more complicated
        and conceptually major entities, shorter for smaller, more conceptually
        minor entities.
--------^ Indent description text, and other multi-line attribute values 8
          spaces, i.e., 1 tab, with tab width set = 8
    *);
----^ indent end of description comment 4 spaces

  end Name;
--^ Indent object and operation ender 2 spaces.

end M;
```

Figure 1: Indentation and commenting conventions.