**Additional Guidelines**

* 1. **Function Names**

Usually every function performs an action, so the name should make clear what it does: Count\_Defrost\_Cycles() instead of DefrostCyclceCount(), Software\_Version\_Check() instead of VersionChk(). This will also make functions and data objects more distinguishable.

* 1. **Global Variables**
* Global variables should be prepended with a 'g\_'.
* Global variables should be avoided whenever possible.
  1. **Global Constants**
* Global constants should be all caps with '\_' separators.
  1. **Alarm Names**
* Alarm names should be prepended with 'AL\_'.
  1. **Defined words**
* Defined words should be all caps with '\_' separators.
  1. **Structured Text Comments**
* Adding comments to each condition can help when you are reading the code.
* Also adding comment to closing braces can help when you are reading code because you don't have to find the begin brace to know what is going on.

For example:

WHILE (1) DO

IF (valid)

(\* if valid \*)

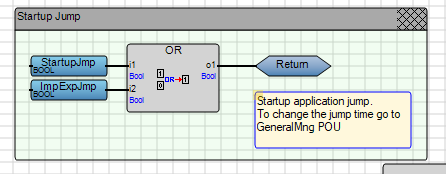
ELSE

(\* not valid \*)

END\_WHILE (\* end forever \*)

* 1. **Function Block Diagram comments**
* Adding region and comments makes you when you are going through the code.

For example:



* 1. **IF THEN ELSE Formatting**

**Layout**

It's up to the programmer. Different bracing styles will yield slightly different looks. One common approach is:

IF (condition) THEN

ELSIF (condition)

ELSE

(condition)

END\_IF;

If you have *ELSIF* statements, then it is usually a good idea to always have an else block for finding unhandled cases. Maybe put a log message in the else even if there is no corrective action taken.

**Condition Format**

Always put the constant on the left hand side of an equality/inequality comparison. For example:

if ( 6 == errorNum ) ...

One reason is that if you leave out one of the = signs, the compiler will find the error for you. A second reason is that it puts the value you are looking for right up front where you can find it instead of buried at the end of your expression. It takes a little time to get used to this format, but then it really gets useful.

* 1. **CASE Formatting**
* Falling through a case statement into the next case statement shall be permitted as long as a comment is included. At the end use END\_CASE to end the case statement.

**Example**

CASE (...) OF

1:

...

(\* comments \*)

2:

...

(\* comments \*)

END\_CASE;

* 1. **Use of FOR Statement**
* Use of FOR statement should be like below,

**Example**

FOR (\*index\*) := (\*mini\*) TO (\*maxi\*) BY (\*step\*) DO

(\*place your code here\*)

END\_FOR;

* 1. **Use of REPEAT Statement**
* Use of REPEAT statement should be like below,

**Example**

REPEAT

(\*place your code here\*)

UNTIL (\*boolean expression\*)

END\_REPEAT;

* 1. **Use of WHILE Statement**
* Use of WHILE statement should be like below,

**Example**

WHILE (\*boolean expression\*) DO

(\*place your code here\*)

END\_WHILE;

**Documentation**

**Comments Should Tell a Story**

Consider your comments a story describing the system. Expect your comments to be extracted by a robot and formed into a man page. Class comments are one part of the story, method signature comments are another part of the story, method arguments another part, and method implementation yet another part. All these parts should weave together and inform someone else at another point of time just exactly what you did and why.

**Document Decisions**

Comments should document decisions. At every point where you had a choice of what to do place a comment describing which choice you made and why.

**Comment Layout**

Each part of the project has a specific comment layout.