

PLC program development guidelines

Itris Automation- February 07th 2019 - Version 1.9.2

www.itris-automation.com

©Itris Automation

2008-2019

Itris Automation

Contents

1 Document purpose **4**

2 Document structure **5**
2.1 Rules classification . . . 5
2.2 Presentation of each rule . . . 5

3 Coding Rules **6**
3.1 Naming . . . 6
3.1.1 PLCs common Rules . . . 6
3.2 Comments . . . 8
3.2.1 PLCs common Rules . . . 8
3.3 Writing code . . . 9
3.3.1 PLCs common Rules . . . 9
3.3.2 Unity Pro specific rules (Schneider Electric). . . . 10
3.4 Structure . . . 10
3.4.1 PLCs common Rules . . . 10
3.5 Useful Information . . . 14
3.5.1 Useful information for all PLCs . . . 14
3.6 Options . . . 17
3.6.1 Unity Project Options . . . 17
3.7 Languages . . . 19
3.7.1 Verification of the programming languages . . . 19

4 Annexes **20**
4.1 Summary of Rules . . . 20
4.2 Document history . . . 23

Iris Automation

Itris Automation

Chapter 1

Document purpose

This document contains coding style guidelines for PLC programming, to improve readability, maintainability and the consistency of PLC programs. It describes simple rules which allow the user to get meaningful quality indicators from the first day of development.

All rules in this document are checked automatically with PLC Checker, an automatic static analysis tool from Itris Automation, which supports a number of different PLC models from brands such as Schneider Electric, Siemens, Rockwell Automation, Codesys, and more.

This document was written by Itris Automation's seasoned consultants, based on their experiences and the analysis of many codes used by our customers in several industries and using various ranges of PLCs. It also reflects some best practices from other computer engineering domains such as embedded systems, which have been applying standard sets of programming guidelines for years.

About Itris Automation

Itris Automation is a software engineering company that provides development tools for the verification, conversion, and documentation of PLC codes to complement the programming tools of international PLC vendors. Thanks to their advanced technologies, Itris Automation helps companies improve development processes and thus deliver higher quality programs in shorter timeframes. The Itris Automation team are experts in software quality and coding standards, and they have combined their knowledge and skills to provide innovative solutions for the industry.

About Schneider Electric

Schneider Electric is leading the Digital Transformation of Energy Management and Automation in Homes, Buildings, Data Centers, Infrastructure and Industries.
With global presence in over 100 countries, Schneider is the undisputable leader in Power Management - Medium Voltage, Low Voltage and Secure Power, and in Automation Systems. We provide integrated efficiency solutions, combining energy, automation and software.
In our global Ecosystem, we collaborate with the largest Partner, Integrator and Developer Community on our Open Platform to deliver real-time control and operational efficiency.
We believe that great people and partners make Schneider a great company and that our commitment to Innovation, Diversity and Sustainability ensures that Life Is On everywhere, for everyone and at every moment.

itris Automation

Chapter 2

Document structure

2.1 Rules classification

The rules described in this document are classified into several categories: Naming, Comments, Writing and Structure. The following part of this document, "Useful Information" provides some points to consider which are not necessarily coding rules, but which can also help improve a program's quality. The last part is about the optional rules based on a PLC workbench or a programming language.

Each category is divided into two parts:
1) The rules that may be applied independently of the PLC for which the program is developed.
2) The rules that are only relevant for a specific PLC or that need to be adapted in order to be applicable.

2.2 Presentation of each rule

The rules are numbered according to their implementation in "PLC Checker", the automatic static analysis tool for code quality management proposed by Itris Automation. For each rule, we find an identical structure made up of a title, an explanation of the rule, a justification of its importance and, when relevant, an example to facilitate understanding of the rule.

Itris Automation

Chapter 3

Coding Rules

3.1 Naming

The elements manipulated in PLCs have mnemonics. Elements are, for example, variables, routines (sections, subroutines, FCs ...) or function blocks (FB). The purpose of this section is to make sure that the symbols follow rules ensuring readability, maintainability and allowing for greater code durability. Generally, the libraries' elements are excluded from naming verification. Of course, when the workbench allows this, we don't recommend changing the names of elements defined in the libraries.

3.1.1 PLCs common Rules

N.1 - All program elements should have a mnemonic

All program's elements should have a mnemonic (program, modules, variables, DFB, FB, OB, SR, routine, Add-On, UDT).

Detailed explanation : When all programs elements have a mnemonic, maintenance is easier and it is easier to understand program content. Moreover, not using mnemonic makes an implicit link between a code or a hardware implementation.

Level : Mandatory

N.2 - Elements names length should be in a given range

The names of program elements must contain at least `Min` characters and not more of `Max` characters.

Detailed explanation : A minimum number of characters ensures more meaningful naming. A maximum of characters allows depending on PLC manufacturer to avoid the mnemonics truncated in display or export, or to improve readability

A maximum number of characters allows:

- In Schneider Unity, Schneider PL7pro and Rockwell RSLogix : to avoid the mnemonics being truncated in display or export, ...
- In Siemens Step7: to improve readability

Level : Recommended

N.3 - Topological physical address should not be used as part of a mnemonic

The elements mnemonics must not contain a reference to their physical address (for instance, "MW2", "FB4 "...).

itris Automation

Detailed explanation : The reference to the physical address of the entity in the code may have bad properties : readability is worse, and it may induce dangerous confusion. It should be noted that with the evolution of software workbenches, referencing the elements names by their physical address is not necessary anymore. Such a reference to the physical address may be unintentional, it is still prohibited.

Example :

Physical address	Mnemonics	Comment
%MW23	Cde_23	Never link the naming of a variable to its physical address

Level : Mandatory

N.4 - Debug mnemonics should not be part of the final program

The following mnemonics (test, toto, titi, tata, tutu, to_delete , todo,...) are forbidden, and the comments should not contain debug adjustment expressions like (to do later, to delete, todo). This is the same issue for unknown variables that the developer does not understand (whose name contains "unknown" or "?" ...). This kind of variable can come from a previous version of the code made by another developer with not enough comment for example.

Detailed explanation : Leaving debug items in the final program reduces its readability and may have a different behaviour than expected. The list of prohibited elements can be configured.

Note: The list of prohibited elements is configurable.
Level : Mandatory for the final program (and of no relevance during the development)

N.5 - Special or graphical characters are prohibited in mnemonics

The special characters set (e.g : à, é, ç...), space, double underscore and accents are prohibited in the mnemonics names. Only alphanumeric characters and the underscore are allowed.

Detailed explanation : Using special characters in naming, may generate some problems of portability to future workbench versions and / or to other platforms. It also reduces the readability in an international context. Finally, the restriction to only alphanumeric characters, ensures a better distinction between variable names.

Example :

```
start__motor := true;

# .... much later in the code

start_motor := false;       # Allowing double underscore '__' generates confusion between these two
mnemonics.
```

Level : Mandatory

N.6 - The programming language keywords should not be used as mnemonics

Programming language keywords (for example "if", "set", "start"...) should not be used as a mnemonic. But it is possible that mnemonics contain keywords. Most cases are handled by PLC workbenches, but there are exceptions.

Detailed explanation : Using keywords as mnemonics affects program readability and portability.

Example :

```
start := true;        # Bad mnemonic, start is a language keyword
start_motor := true;  # Correct mnemonic
```

Level : Mandatory

3.2 Comments

3.2.1 PLCs common Rules

C.1 - All program elements should have a comment

All program elements must be commented (Program, Section, Modules, Variables, DFB, FB, OB, SR).

Detailed explanation : If all the program elements are commented significantly, it is easier to read and understand the program, therefore its maintenance is facilitated.

Level : Mandatory

C.2 - Comments should be longer than a given number of characters

The comments of program elements must contain a minimum number of characters (7 for variables, 15 for codes) to avoid insignificant comments.

Detailed explanation : having minimum length ensures that the comments are more meaningful. Having a maximum length ensures that the comments are still readable.

Level : Mandatory

C.3 - Each network should have a comment

The networks must be commented. This rule is specific to some languages of IEC61131 standard. Furthermore, the notion of network is not always present in manufacturer workbenches.

Detailed explanation : Moreover the comments rate for networks is global, it's important to ensure a good distribution of comments.

Level : Mandatory

C.4 - Comments should not contain comment start marker

Comment should not contain start comment marker (e.g, "/ *", "(*"). In addition to that, nested comments are forbidden (IEC11631 §2.1.5, system requirement 1.5.1 d).

Detailed explanation : The presence of a start comment marker is often a symptom of a forgotten end comment marker, which can lead to inhibition of part of the program. In some PLC workbenches, a global projects option enables or disables support for nested comment.

Example

```
/* A comment with the end marker has been forgotten

Critical_section_that_must_be_executed_which_is_found_in_the comment();
```

itris Automation

```
/* <- The comment of the first comment start marker is now commented because the end marker of the
previous comment has been forgotten */
```

Level : Recommended

3.3 Writing code

3.3.1 PLCs common Rules

E.1 - All variables should be written before being read

Except for some well-defined categories of variables, all other variables should be written before being read during one PLC cycle.

Detailed explanation : a variable that is read before being written, can increase program latency. Efficient code should respect during one PLC cycle the following diagram:

In addition, make sure that the variables that are never written in the code, have been initialized otherwise or have been updated by other ways.

In some cases it is justified to ignore this rule:

- variables that were initialized in the database,
- variables updated by a communications link,
- state variables that store the value of a previous PLC cycle.

Example:

```
if urgenteAlarme then
  alarmeTraitement();
end if;

# ...

urgenteAlarme := presenceDefault AND ...;
```

In this example, the call of alarm function `alarmeTraitement` happens during the next PLC cycle, which increases the average response time. This delay is around one PLC cycle.

Level : Recommended

E.2 - The parameters of the user functional blocks should be used correctly

This rule is divided into three sub-rules:

- a and b - Inputs of the user functional blocks should be read and not written.
- c and d - Inputs/Outputs of the user functional blocks should be read and written.
- e and f - Outputs of the user functional blocks should be written.

Detailed explanation : Formal parameters incorrectly used are proof of unfinished code or a code that still contains unnecessary elements of correct behavior. In some PLCs, the parameter misuses may generate undesirable effects potentially dangerous, such as data corruption. The use check of the private parameters of user blocks is done by the rule S7.

Note: Checking whether private parameters of user blocks are used is done by the rule S7.

Level : Mandatory

Itris Automation

3.3.2 Unity Pro specific rules (Schneider Electric).

E.U.1 - The functions of the Unity obsolete libraries should not be used

Unity's obsolete libraries shall not be used.

Detailed explanation : It is recommended not to use obsolete functions when portability of code is required. In Unity, some libraries are obsolete and exist to ensure backward compatibility with previous versions of Schneider PLCs. In new applications, the usage of the functions of those libraries must be avoided for future portability problems. Furthermore, some of those functions can make the code non-deterministic (ex: shifts on signed integers shl_int, shlz_int, shr_int, shrz_int, rol_int, ror_int).

Level : Recommended

3.4 Structure

3.4.1 PLCs common Rules

S.1 - Backward jumps are forbidden

Backward jumps are forbidden.

Detailed explanation : Backward jumps may trigger an infinite loop which will halt the PLC loop. Moreover, avoiding jumps leads to an easier understanding of the code and therefore a better maintainability.

Example:

```
L3:
  /* code always executed */

Always
True
Condition                       L3
-| |----------------------------->>
```

Level : Mandatory

S.2 - A variable should be elaborated only in one routine

A variable should be elaborated only in one routine.

Detailed explanation : A simple and clear code architecture facilitates understanding of code and therefore its maintenance. Moreover, multiples variable elaborations can lead to erratic behavior of the code.

Level : Recommended

S.3 - A variable should be written from one task only

A variable should be written from one task only.

Detailed explanation : Writing a variable from several tasks leads to a non-deterministic behavior coming from a potential access conflict such as one task misses a value that was accessed by the other task.

Level : Mandatory

itris Automation

S.4 - A physical output should be written only once per PLC cycle

A physical output should be written only once per PLC cycle.

Detailed explanation : Multiple output writing may lead to problems of dependability. Moreover, a code where the outputs are written more than one time is less easy to understand and thus maintain. It should be noted that writing in a loop is considered like a multiple writing and therefore constitutes a violation of this rule.

Example :

```
if very_complicated_condition then
   Critical_Output := TRUE;
end if;


#.... much later in the code

if another_very_complicated_condition then
   Critical_Output := FALSE;
end if;
```

In the case where this output is not controlled how it should, it is very difficult to identify which assignment is executed.

Level : Mandatory

S.5 - Variables should not be localized

Variable memory mapping should be reserved to variables whose elaboration or consumption is done outside the program : communication tags, I/O variables, special function variables. All internal variables should not be localized.

Detailed explanation : Using localized variables leads to constraints in the ability of program evolution due to a stronger memory organization. It reduces portability as memory organization is varying from one PLC to another. So the code can not be reused in another application as in the other application the given memory zone may be reserved for other purposes. Moreover, it may lead to memory organisation mistakes in which several variables will be saved in the same memory cells. This bug may be silent during a long time and only be encountered late in the development or in the production cycle.

Level : Recommended

S.6 - DFB/FBs instances should be called only once

FB/DFB/ADD_ON instances should be called once.

Detailed explanation : A functional block instance uses its variables to save the FB state between two consecutive cycles. Each FB execution should modify those variables to show the state evolution. When executed more than once, it looks like the cycle is progressing twice as quick as other instances.

Often using the same instance more than once is the result of a cut and paste that was not updated after.

It may lead to bugs hard to discover.

Level : Mandatory for the final program (Leave empty while under developpment)

S.7 - Declared variables should be used

Variables and defined types should be used, except reserves variables which should not be used.

Itris Automation

Detailed explanation : Variables (except reserves) and unused types are sometimes a sign of a forgotten treatment or incomplete database cleaning. The reserves variables used are sometimes the sign of renaming not performed.

Level : Mandatory for the final program (Leave empty while under developpment)

S.8 - Variables location shall not overlap

The overlapping variables have to be checked.

Detailed explanation : Localized variables that overlap are equivalent to an implicit transfer of value between each other. This is sometimes a difficulty to code readability. It is also sometimes a localization error.

Level : Mandatory

S.9 - Complexity measure

Complex code hides bugs.

Detailed explanation : Parts of program are more complicated than others. This is especially the case when the execution of sequential flow is often cut. This rule searches for loops, backward gotos and high complex instructions nested level.

Level : recommended

S.12 - Conditional instructions without default case

When using an instruction IF or an instruction CASE in structured text, it is important to always add a default case handler: an ELSE part for IF instructions or DEFAULT for CASE instructions.

Detailed explanation : When always writing default cases for conditional instructions, readers know that all decision paths have been taken into account. Therefore this rule make sure that for each IF instruction there is a ELSE statement and that for each CASE instruction there is a OTHERS statement. Thus , the developer is making sure that all cases have been taken into account in the program, even if it's a 'ELSE NULL;' case.

Note: This rule is only usable for code written in Structured Text language.

Level : Recommended

S.13 - SFC diagrams without an initial step

A SFC code should contain at least one and no more than one initial step, except for macro-steps that should not contain any initial step.

Detailed explanation : Using more than one initial step in the same SFC code is error prone. Also SFC without intial steps have to be identified. Such diagrams are generally launched for other sections in the program. It is more complicated to understand the behavior of the diagram, thus reducing the maintainability of the application. .

Level : Mandatory

S.15 - Immediate values should not be used in the code

The immediate values or constants should not be used in the code.

Detailed explanation : Using immediate values or constants in the code gives less information than using an explicit well named constant. In case of modification of values to tweak the program, you need to use the defined one.

itris Automation

Example: Don't do:

```
FOR I IN 1..16 DO
  ...
END_FOR;
```

Instead do:

```
FOR I IN 1..ELEMENT_MAX DO
  ...
END_FOR;
```

Level : Recommended

S.16 - The float or reals should not be compared using equality or difference operators

The floats or reals variables should use the less than, less or equal, greater than or greater or equal operators only.

Detailed explanation : Due to the high precision of floating point, a condition using equality or difference operators may be always TRUE or always FALSE.

Example: Don't do:

```
IF Temperature = 20.0 THEN
...
```

Instead do:

```
EPSILON : REAL := 0.1; (* measure uncertainty *)

IF ABS(Temperature - 20.0) <= EPSILON THEN
...
```

Level : Mandatory

S.17 - Indentation of Structured Text

When writing Structured Text code, the developer should ensure that the code is correctly indented to make it readable.

Detailed explanation : A code badly indented is difficult to read and understand. Good indentation requires, that all instructions with the same level of deepness begin at the same point of their line.

Example: Don't do:

```
a := B + C; E := D + H;
     G := F + K;
```

Instead do:

```
a := B + C;
E := D + H;
G := F + K;
```

Level : Recommended

Itris Automation

3.5 Useful Information

3.5.1 Useful information for all PLCs

I.1 - Comments rate

It is recommended to have a minimum percentage of commented code.

Detailed explanation : It is recommended to have at least 30% of commented instructions (ratio between commented instructions over overall instructions count). Such a ratio ensures that maintenance and readability properties are well taken into account during the development cycle.

Level : Mandatory

I.2 - Code in the comments

It is important not to have code in comments.

Detailed explanation : Code in comments can be optional code, that is not part of the main program. It could also be bypassed code during commissioning time avoiding that it is executed or it could also be additional instructions to help test the program. Finally it can also be a peace of code that is not working and was stored in comments, and then not removed. Production code should not contain development phases traces. Code in Comment can leave in doubt the engineer who was not the author of the code, but has to maintain it.

Level : Mandatory for the final program (Leave empty while under developpment)

I.3 - Dead code

the program should not contain dead code.

Detailed explanation : Programs with dead code are more difficult to read and to maintain. It can be a symptom of a functional problem. Warning if PLC Checker can detect unreferenced functions and code parts behind unconditional jumps. This rule is not exhaustive at all (for example dead code due to an expression evaluation is not detected).

Level : Fatal

I.4 - List of locked elements

The locked elements are not scanned by the PLC Checker tool, they should be identified.

Detailed explanation : In applications where the major part of the code is locked, it is important to know it.

Level : Mandatory

I.5 - Cyclomatic number v(G)

Cyclomatic number gives information about complexity. It is important to keep it reasonably low.

Detailed explanation : Cyclomatic number counts the number of different paths that can be executed in a routine. The higher this number is, the more difficult it is to validate the correct behavior of the routine. In such a case, the routine should be divided in subroutines. Each of them having a cyclomatic number below the given threshold.

itris Automation

Level : info

I.6 - Essential Complexity ev(G)

Essential complexity gives information about unstructured instructions. It is important to keep it very low.

Detailed explanation : Essential complexity counts the number of different unstructured instructions in a routine. The higher this number is, the more difficult it is to validate the correct behavior of the routine. Most of the time, unstructured instructions can normally be replaced by structured instructions.

Level : Info

I7 - Source Line of Code count

Source line of code counts number of instructions in a POU. It is important to keep it under control.

Detailed explanation : Very big procedure or function blocks may be difficult to test, to read and to maintain. It may be better to split a very big procedure into two or more procedure of reasonable size.

Level : Info

I8 - Sequential Function Chart (SFC) steps count

This is the count of steps in a given SFC.

Detailed explanation : This rules returns the number of steps found by SFC. This number may be further used to detect complex SFC or to give an overview of relative size of SFCs.

Level : Info

I9 - SFC branches count

This number sums the number of branches in each AND/OR divergence found in a given SFC.

Detailed explanation : This rule calculates the number of branches available in a given SFC. This count gives information about SFC complexity. This number is equivalent to the v(G) applied to imperative coding languages.

Level : Info

I10 - Copy/Paste detection ratio

This number detects the percentage of code which has been duplicated.

Detailed explanation : Reusing the code by copying code is not something good. Maintenance and evolution are more expensive to perform because modifications and correction should be applied at each copy/paste instance. Moreover, the manual copy/paste is error prone : after doing the copy/paste, code should be modified. When code is not modified correctly, it introduces subtle bugs hard to discover.

Level : Info

Itris Automation

I.11 - Halstead Metrics

All Halstead metric attribute are given in this rule. Those are the different metrics proposed by Maurice Halstead (number of operators, number of operands, program volume, difficulty level...)

Detailed explanation : These metrics help to provide an estimation of the number of errors in a program upon delivery.

Level : Info

I.12 - Number of External References

This is the number of external references in a POU.

Detailed explanation : To ensure the re-usability of POU code, the use of external references should be avoided. The external exchanges should be made through the defined interfaces that are the formal parameters of inputs, outputs, and input/output of the POU. The aim is to have zero external references in the POU code that is to be reused.

Note : It is acceptable to reference system variables because their signification and address do not change from one program to another.

Level : Info

I.13 - Number of calls of a POU

The number of calls of each routine/POU in a program.

Detailed explanation : Except for the fact that a POU called zero times corresponds to dead code, there is no limit for the number of times a POU can be called. This number allows the POUs used multiple times to be identified.

Level : Info

I.14 - Number of times a POU calls another POU

This rule counts the number of routines called from a given routine/POU.

Detailed explanation : A POU which does not reference any other blocks of code is an autonomous POU. When this number is high, the concerned POU should be split into smaller blocks in order to improve the structure of the program.

Level : Info

I.15 - Depth of calls of a POU

This rule measures the min and max depths of call of a given block of code (POU).

Detailed explanation : These measures allow the identification of the depth of calls in the structure of the program. A block of code can be called at the task level, in which case the depth is low, or at a greater depth in the structure of the program, in which case the depth is high. A block of code can be called multiple times in a program and can therefore have multiple depths. A low depth corresponds to an organisation or management block of a program. A high depth corresponds to a specialised block of code.

Level : Info

itris Automation

3.6 Options

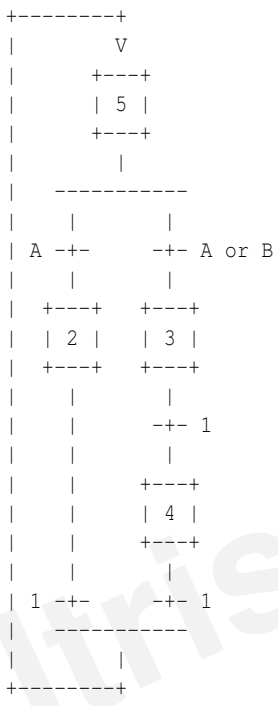
3.6.1 Unity Project Options

O.U.1 - SFC : "Multiple Token" should not be Allowed

Under Unity Pro workbench, in project options, multi-token checked box should not be selected.

Detailed explanation : The multi-token mode under Unity Pro is used to enforce ascending compatibility with old Schneider-Electric PLC generations. It should not be used for new development as the resulted behavior of the PLC may be nondeterministic.

Example : Running the following SFC with an always true A condition, all steps are simultaneously activated after several cycles.



Level : fatal

O.U.2 - SFC : "Keep previous steps in active state" should not be Allowed

In the language extensions tab of project settings, you have to desactivate the checkbox "SetSteps: keep previous step(s) in active state".

Detailed explanation : This option is only available if the Allow multiple token checkbox activated see O.U.1.

Level : Recommended

O.U.3 - SFC : "Parallel branches: allow jump in/out" should not be Allowed

In the language extensions tab of project settings, you have to deactivate the checkbox "Parallel branches: allow jump in/out".

Itris Automation

Detailed explanation : This option is only available if the Allow multiple token checkbox activated see O.U.1.

Level : Recommended

O.U.4 - SFC : Only one evolutions by divergence of sequence selection

In the language extensions tab of project settings, you have to deactivate the check-box "Allow multiple evolutions by divergence of sequence selection".

Detailed explanation : This option is only available if the Allow multiple token check-box activated see O.U.1.

Level : Recommended

O.U.5 - ST : "Jump and label" should not be Allowed

In the language extensions tab of project settings, you have to deactivate the check-box "Allow Jump and label".

Detailed explanation : In general, usage of jumps reduces an understanding of the code and therefore its ease of maintenance.

Level : fatal

O.U.6 - Identifiers: leading digits in names should not be Allowed

In the language extensions tab of project settings, you have to deactivate the check-box "Allow leading digits".

Detailed explanation : In general, usage of leading digits in names, affects the readability and portability of the program.

Level : fatal

O.U.7 - Identifiers: the standard (IEC) character set shall be used in names

In the language extensions tab of project settings, you have to deactivate the check-box "standard".

Detailed explanation : In general, usage of the no standard (IEC) character set in names , affects the readability and portability of the program.

Example : In Latin languages, it is difficult to see the difference between two different variables with same name except accents. A real example seen in a production program was a variable named ’départ’ and another one named ’depart’.

Level : fatal

O.U.8 - Graphical languages (FBD/LD): Usage of ST expressions should not be Allowed

In the language extensions tab of project settings, you have to desactivate the checkbox "Usage of ST expressions".

Detailed explanation : In general, usage of ST expressions in FBD, affects the readability of the program.

Level : Recommended

Itris Automation

O.U.9 - Upload information: Upload information with comments and animation tables

In the Build tab of project settings (project generation settings) you have to :

1. Activate the check-box "Include" in the Upload information,
2. Activate the check-box "Comments (Variables and types)" in the Upload information,
3. Activate the check-box "Animation tables" in the Upload information

Detailed explanation : If the PLC memory allow, the storage of all upload informations, ensure to restore the source of the project from the PLC, if lost backup, which improves the maintainability in the long term.

Level : fatal

O.U.10 - Ladder Diagram : Right-justify coils

In the Editors tab of project settings you have to activate the checkbox "Right-justify coils" for Ladder Diagram (LD).

Detailed explanation : Right-justify coils allow a best readability of the program.

Level : fatal

O.U.12 - PLC CPU must be configured to start automatically in RUN

In the UnityPro software the CPU of the PLC can be configured to start automatically in RUN or not. This rule checks that the CPU will start automatically in RUN. It will prevent the PLC to be stopped after a shutdown and restart.

Detailed explanation :

Level : fatal

3.7 Languages

3.7.1 Verification of the programming languages

L.1 - The IL language is forbidden

It is recommended to avoid usage of the instruction list language since it is a low level language which is not easily readable.

Detailed explanation : Depending on the type of PLC being programmed, it may be difficult to avoid the IL language. Indeed, for example, with a Siemens Step5 or Step7 PLC, all sections in Ladder (LAD) and Function Block Diagrams (FBD) are actually saved in the IL language. For other PLCs such as Schneider's and Rockwell's, it is much easier to avoid the usage of the IL language.

Level : Fatal

Itris Automation

Chapter 4

Annexes

4.1 Summary of Rules

This annex summarizes all the rules and useful information in this document.

Naming rules

- N.1 - All PLC types - All elements of the program must be named
- N.2 - All PLC types - Elements name length should be in a given range
- N.3 - All PLC types - Topological physical address should not be used as part of a mnemonic
- N.4 - All PLC types - Testing mnemonics should not be part of the final program
- N.5 - All PLC types - Special or graphical characters are prohibited in mnemonics
- N.6 - All PLC types - The programming language keywords should not be used as mnemonics

Commenting rules

- C.1 - All PLC types - All program elements should have a comment
- C.2 - All PLC types - Comments should be longer than a given number of characters
- C.3 - All PLC types - Each network should have a comment
- C.4 - All PLC types - Comments should not only contain the comment start marker

Writing rules

- E.1 - All PLC types - All variables should be written before being read
- E.2.a, E.2.b - All PLC types - Inputs of the user functional blocks should be read and not written.
- E.2.c, E.2.d - All PLC types - Inputs/Outputs of the user functional blocks should be read and written.
- E.2.e - All PLC types - Outputs of the user functional blocks should be written before being read.
- E.U.1 - SE Unity - The functions of the Unity obsolete libraries should not be used

Itris Automation

Structure rules

- S.1 - All PLC types - Backward jumps are forbidden
- S.2 - All PLC types - A variable should be elaborated only in one routine
- S.3 - All PLC types - A variable should be written from one task only
- S.4 - All PLC types - A physical output should be written only once per PLC cycle
- S.5 - All PLC types - Variable should not be localized
- S.6 - All PLC types - DFB/FBs instances should be called once
- S.7 - All PLC types - Declared variables should be used
- S.8 - All PLC types - Variables location doesn’t overlap
- S.9 - All PLC types - Complexity measure
- S.10 - All PLC types - SCADA Limitations
- S.12 - All PLC types - Conditional instructions without default case
- S.13 - All PLC types - SFC diagrams without an initial step
- S.15 - All PLC types - Immediate values should not be used in the code
- S.16 - All PLC types - The float or reals should not be compared using equality or difference operators
- S.17 - All PLC types - Indentation of Structured Text

Informations Utilities

- I.1 - All PLC types - Comments rate
- I.2 - All PLC types - Code in the comments
- I.3 - All PLC types - Dead code
- I.4 - All PLC types - List of locked elements
- I.5 - All PLC types - Cyclomatic number v(G)
- I.6 - All PLC types - Essential Complexity ev(G)
- I.7 - All PLC types - Source Line of Code count
- I.8 - All PLC types - SFC steps count
- I.9 - All PLC types - SFC branches count
- I.10 - All PLC types - Copy/Paste detection ratio
- I.11 - All PLC types - Halstead metrics
- I.12 - All PLC types - Number of external references
- I.13 - All PLC types - Number of calls of a POU
- I.14 - All PLC types - Number of times a POU calls another POU
- I.15 - All PLC types - Depth of calls of a POU

Itris Automation

Options

- O.U.1 - SE Unity - SFC : "Multiple Token" should not be Allowed
- O.U.2 - SE Unity - SFC : "Keep previous steps in active state" should not be Allowed
- O.U.3 - SE Unity - SFC : "Parallel branches: allow jump in/out" should not be Allowed
- O.U.4 - SE Unity - SFC : Only one evolutions by divergence of sequence selection
- O.U.5 - SE Unity - ST : "Jump and label" should not be Allowed
- O.U.6 - SE Unity - Identifiers: leading digits in names should not be Allowed
- O.U.7 - SE Unity - Identifiers:Using the standard (IEC) character set in names
- O.U.8 - SE Unity - Graphical languages (FBD/LD): Usage of ST expressions should not be Allowed
- O.U.9 - SE Unity - Upload information: Upload information with comments and animation tables
- O.U.10 - SE Unity - Ladder Diagram : Right-justify coils
- O.U.12 - SE Unity - PLC CPU must be configured to start automatically in RUN

Languages

- L.1 - All PLC types - The IL language is forbidden

itris Automation

4.2 Document history

Date	Changes
2010 Jul 12th	Initial version
2010 Dec 21th	Add Options chapter
2011 Nov 9th	Add description for rules S5, S6, S7, S8, S9, S10, I5, I6, I7, I8, I9, I10
2011 Nov 21th	Add version number. Texts Corrections. Extend EU1 rule to all functions of the obsolete libraries
2012 Jan 20th	Proofreading and corrections (grammar, spelling)
2014 Jan 27th	Add description for rules S12, S13 and L1 - Modify N4
2015 Nov 13th	Proofreading and corrections (spelling)
2016 Nov 17th	Add description of rules S15, S16 and OU12. Proofreading and corrections based on GQR file 3.1.1 version (by WAI)
2017 Nov 21th	Proofreading and corrections: spelling, avoid French double-quote, ... (by DSA)
2018 Oct 8th	Add rules S17 and I11 to I15 (by DSA)
2019 Jan 22nd	Proofreading and corrections (by RBU)
2019 Jan 31th	Automatisation of the document via links to the rules repository wiki (by SLE)
2019 Mar 14th	Added About Itris & About Schneider to Document Purpose (by RBU)

Itris Automation