

Guide de codage des programmes automates

Itris Automation- 10 avril 2019 - Version 1.9.3

www.itris-automation.com

©Itris Automation

2008-2019

Itris Automation

Contents

1 Objectif de ce document **4**

2 Structure de ce document **5**
2.1 Classification des règles 5
2.2 Présentation de chaque règle 5

3 Règles de codage **6**
3.1 Nommage 6
3.1.1 Règles communes à tous les automates 6
3.2 Commentaires 8
3.2.1 Règles communes à tous les automates 8
3.3 Écriture du code 9
3.3.1 Règles communes à tous les automates 9
3.3.2 Règles spécifiques à Schneider Electric Unity 10
3.4 Structuration 10
3.4.1 Règles communes à tous les automates 10
3.5 Informations utiles 14
3.5.1 Informations utiles communes à tous les automates 14
3.6 Options 18
3.6.1 Options Unity des projets automates 18
3.7 Langages 20
3.7.1 Vérification de langage de programmation 20

4 Annexes **21**
4.1 Résumé des règles 21
4.2 Historique du document 23

Iris Automation

Itris Automation

Chapter 1

Objectif de ce document

Ce document propose un style de codage pour la programmation des automates industriels, afin d'améliorer la lisibilité, la maintenabilité et la cohérence des programmes. Il décrit des règles simples permettant d'obtenir, dès leur première application, des résultats probants sur la qualité du code automate.

Toutes les règles présentes dans ce document sont vérifiables automatiquement à l'aide de PLC Checker, un outil d'analyse statique automatique proposé par Itris Automation, et ce pour les automates programmables de plusieurs marques comme Schneider Electric, Siemens, Rockwell Automation, et Codesys, parmi d'autres.

Ce document a été rédigé à partir d'une analyse critique de nombreux référentiels déjà utilisés par les clients d'Itris Automation dans des domaines d'activité variés et possédant des gammes d'automates diversifiées. Il s'inspire également de l'expérience des équipes d'automaticiens d'Itris Automation.

Présentation d'Itris Automation
Itris Automation, une société de Schneider Electric, est une société éditrice de logiciels qui fournit des outils de conversion, vérification, documentation, et aide au diagnostic pour les automates programmables, peu importe la marque et le modèle de l'automate. Grâce à ses technologies avancées, Itris Automation aide ses clients à améliorer leurs process de développement, et ainsi à écrire des programmes de meilleure qualité plus rapidement. Spécialistes de la qualité logicielle et des standards de codage, les équipes d'Itris Automation mettent en œuvre des technologies innovantes au service de l'industrie.

Présentation de Schneider Electric
Schneider Electric mène la Transformation Numérique de la Gestion de l'énergie et des Automatismes dans le Résidentiel, les Bâtiments, les Centres de données, les Infrastructures et les Industries.
Présent dans plus de 100 pays, Schneider est le leader incontesté dans la Gestion électrique – Moyenne tension, Basse tension et Energie sécurisée, et dans les Systèmes d'automatismes. Nous fournissions des solutions intégrées pour l'efficacité, en associant énergie, automatismes et logiciels.
Dans notre Ecosystème global, nous collaborons avec la plus large Communauté de Partenaires, d'Intégrateurs, et de Développeurs sur notre Plateforme ouverte pour offrir contrôle en temps réel et efficacité opérationnelle.
Nous pensons que des collaborateurs et des partenaires formidables font de Schneider une entreprise exceptionnelle et que notre engagement à l'Innovation, la Diversité et le Développement durable contribuent à Enrichir la vie (Life is On) de chacun, partout et à tout moment.

Itris Automation

Chapter 2

Structure de ce document

2.1 Classification des règles

Les règles contenues dans ce document sont classées dans plusieurs catégories : Nommage, Commentaire, Écriture, Structuration. Vient ensuite une partie appelée "Informations utiles" qui propose également des pistes de réflexion. Elles ne constituent pas à proprement parler des règles de codage, mais peuvent permettre l'amélioration de la qualité des programmes. Pour finir, des règles de vérifications des options du projet et des langages sont proposées pour certains ateliers.

Chaque catégorie est éventuellement subdivisée en deux parties : les règles qui peuvent s'appliquer indépendamment de l'automate pour lequel le programme est développé et celles qui ne sont pertinentes que pour un automate particulier ou qui nécessitent d'être déclinées afin d'être effectivement applicables.

2.2 Présentation de chaque règle

Les règles sont numérotées en accord avec leur implémentation dans PLC Checker, l'outil d'analyse statique automatique pour la gestion de la qualité des codes automates proposé par Itris Automation. Pour chaque règle, on retrouve une structure identique composée d'un titre de section qui résume la règle, de la règle proprement dite et de commentaires justifiant notamment la présence de la règle dans le document. Lorsque cela semblait nécessaire pour faciliter la compréhension de la règle, un exemple de code a été rédigé.

Il est à noter que les explications complémentaires et les exemples n'ont pas pour but de constituer une formation exhaustive à l'aspect de la programmation auquel il est fait référence dans la règle.

Itris Automation

Chapter 3

Règles de codage

3.1 Nommage

Les éléments manipulés dans les ateliers automates portent des mnémoniques. Ce sont par exemple, des variables, des routines (sections, sous-routines, FC…), des blocs fonction (FB). L'objectif de cette section est de s'assurer que les mnémoniques suivent des règles assurant la lisibilité, la maintenabilité et permettant une plus grande pérennité du code. D'une manière générale, les éléments propres aux librairies des ateliers constructeurs sont exclus des contrôles de nommage. Bien entendu, lorsque les ateliers le permettent, nous déconseillons de modifier les noms des éléments propres aux librairies.

3.1.1 Règles communes à tous les automates

N.1 - Tous les éléments constituant le programme doivent être nommés

Tous les éléments constituant le programme doivent être nommés (Programme, Modules, Variables, DFB, FB, OB, SR, routine, Add-On, UDT).

Explication complémentaire : Si tous les éléments du programme sont nommés de façon significative, la maintenance est facilitée et il est plus facile de comprendre le programme. Par ailleurs, ne pas utiliser de mnémonique revient à lier le code à une implémentation matérielle donnée et induit donc une perte d'évolutivité.

Niveau conseillé : Obligatoire

Tous les éléments constituant le programme doivent être nommés (Programme, Modules, Variables, DFB, FB, OB, SR).

N.2 - Les noms des éléments du programme doivent avoir une taille minimum et maximum

Les noms des éléments du programme doivent comporter au moins un certain nombre de caractères et ne doivent pas être trop long.

Explication complémentaire : Un nombre minimum de caractères assure que le nommage est plus significatif.

Un maximum de caractères permet :

- sous Unity, PL7pro et Rockwell, d'éviter les mnémoniques tronqués à l'affichage, à l'export, . . .
- sous Step7, d'améliorer la lisibilité

Niveau conseillé : Recommandé

itris Automation

N.3 - Les mnémoniques des éléments ne font pas référence à leur adresse physique

Les mnémoniques des éléments ne doivent pas contenir une référence à leur adresse physique (par exemple, "MW2", "FB4"...).

Explication complémentaire : La référence à l'adresse physique est susceptible de ne plus être pertinente suite à une évolution du code et donc de nuire à la lisibilité du code voire d'induire une confusion dangereuse. Il est notable que la notion de couplage avec l'adresse physique n'est pas nécessairement pertinente pour certains automates du marché pour lesquels les variables ne sont pas localisées. Une telle référence à l'adresse physique peut-être involontaire, elle n'en demeure pas moins interdite.

Exemple :

Adresse Physique	Mnémoniques	Commentaire
%MW23	Cde_23	Ne jamais lier le nommage de la variable à son adresse physique

Niveau conseillé : Obligatoire

N.4 - Les mnémoniques de mise au point ne doivent pas faire partie du programme final

La mise au point ne doit pas faire partie du programme. Notamment, les noms d'éléments suivants (*test, toto, titi, tata, tutu, a effacer, todo, ,...*) sont interdits et les commentaires ne doivent pas contenir d'expressions laissant penser que le programme est encore en cours d'écriture (*a faire plus tard, a effacer, todo*). De même les variables inconnues du développeur sont à éviter (dont le nom ou le commentaire contient « unknown » , « inconnu », ou encore « ? »). Il peut s'agir par exemple de variables venant de versions précédentes créées par d'autre développeurs et sans suffisamment de commentaire.

Explication complémentaire : Laisser des éléments liés à la mise au point dans le programme en production nuit à sa lisibilité et peut avoir pour conséquences un comportement différent de celui attendu.

Note: La liste des éléments interdits est évolutive
Niveau conseillé : Obligatoire pour le programme final (et sans objet en cours de développement)

N.5 - Les caractères spéciaux sont interdits dans les noms des mnémoniques

Les caractères spéciaux sont interdits dans les noms des mnémoniques. Seuls les caractères alphanumériques et le "souligné" sont autorisés. En particulier, l'espace, le double "souligné" et les accents sont interdits.

Explication complémentaire : Utiliser des caractères spéciaux dans les noms des mnémoniques nuit à la portabilité du code vers des versions postérieures potentielles des ateliers de développement et vers d'autres plateformes. Ils nuisent aussi à la lisibilité dans un contexte international. Enfin, restreindre le jeu de caractères autorisés permet de s'assurer d'une meilleure distinction entre les noms des variables.

Exemple :

```
départ := true;

# .... beaucoup plus loin dans le code

depart := false;       # Le fait d'autoriser le caractère spécial 'é' conduit à une confusion
possible entre les mnémoniques.
```

Itris Automation

Niveau conseillé : Obligatoire

N.6 - Les mots clefs des langages de programmation ne doivent pas servir de mnémoniques

Les mots clefs des langages de programmation (par exemple "if", "set", "start"...) ne peuvent pas être utilisés comme noms des éléments du programme. En revanche les mnémoniques peuvent contenir un mot clé du langage.

Explication complémentaire : Utiliser des mots clefs des langages comme noms des mnémoniques nuit à la lisibilité, à la portabilité du programme et peut poser des problèmes lors de la compilation du code. Il est à noter qu'un certains nombre d'ateliers interdisent déjà un tel nommage.

Exemple :

```
start := true;        # Mauvais mnemonique, start est un mot-cle du langage
start_moteur := true; # Correct mnemonique
```

Niveau conseillé : Obligatoire

3.2 Commentaires

Au-delà du nommage, il est important de suivre des règles quant à la bonne utilisation des commentaires. En effet, plus un commentaire est explicite et détaillé, plus il facilitera la compréhension du code. D'une manière générale, les éléments propres aux librairies des ateliers constructeurs sont exclus des contrôles de commentaires.

3.2.1 Règles communes à tous les automates

C.1 - Tous les éléments constituant le programme doivent être commentés

Tous les éléments constituant le programme doivent être commentés (Programme, Modules, Variables, DFB, FB, OB).

Explication complémentaire : Si tous les éléments du programme sont commentés de façon significative, le programme est plus facile à lire donc plus facile à comprendre et donc plus facile à maintenir.

Niveau conseillé : Obligatoire

C.2 - Les commentaires doivent comporter un minimum de caractères

Les commentaires doivent comporter au moins 7 caractères significatifs (pour les variables) et 15 caractères significatifs (pour les codes).

Explication complémentaire : Un nombre minimum de caractères assure que les commentaires sont plus significatifs.

Niveau conseillé : Obligatoire

C.3 - Chaque réseau doit être commenté

Lorsqu'ils sont utilisés, les réseaux (aussi appelés rungs) doivent être commentés. Cette règle est spécifique à certains langages de la norme IEC61131. Par ailleurs, la notion de rung n'est pas toujours présente dans les ateliers de développement des constructeurs d'automates.

itris Automation

Explication complémentaire : Au delà du taux de commentaire global, il est important d'assurer une bonne répartition des commentaires. Une telle règle est aussi un guide qui assure que le développeur respectera une certains cohérence dans le découpage de son code.

Niveau conseillé : Obligatoire

C.4 - Les commentaires ne contiennent pas le marqueur de début de commentaire

Les commentaires ne doivent pas contenir le marqueur de début de commentaire (par exemple, "/ *", "(*").

Explication complémentaire : La présence d'un marqueur de début de commentaire dans un commentaire est souvent le symptôme d'un oubli de marqueur de fin de commentaire, pouvant conduire à la non-exécution d'une partie de code.

Exemple:

```
/* Un commentaire quelconque dont le marqueur de fin a été oublié


Section_critique_qu_il_faut_a_tout_prix_executer_mais_qui_se_retrouve_commentee();

/* Ce commentaire ne suit pas la norme car le précédent marqueur de fin de commentaire a été
oublié*/
```

Niveau conseillé : Recommandé

3.3 Écriture du code

3.3.1 Règles communes à tous les automates

E.1 - Toutes les variables doivent être écrites avant d'être lues

A l'exception des entrées et des variables système, toutes les variables doivent être écrites avant d'être lues durant un cycle automate.

Explication complémentaire : Une variable lue avant d'être écrite a pour conséquence de rendre le code non-réactif : un code efficace doit respecter un cycle lecture des entrées → traitement → mise à jour des sorties. Par ailleurs, on doit s'assurer que les variables qui ne ne sont jamais écrites par le code ont bien été initialisées ou proviennent de l'extérieur du programme. Dans un certain nombres de cas, il est justifié de ne pas respecter cette règle : pour les variables qui ont été initialisées dans la base de données, pour les variables qui proviennent des communications, pour les variables d'état qui mémorisent la valeur d'un cycle antérieur.

Exemple:

```
if alarmeUrgente then
  traitementAlarme;
end if;

# ...

alarmeUrgente := presenceDefaut AND ...;

# Dans ce cas là, l'appel de la routine de traitement d'alarme se fait lors du cycle suivant
# ce qui augmente le temps de réponse moyen.
# Ce retard est de l'ordre de grandeur d'un tour de cycle.
```

Itris Automation

Niveau conseillé : Recommandé

E.2 - Les paramètres formels des blocs fonctionnels utilisateur doivent être utilisés correctement

Cette règle se subdivise en sous-règles :

- a, b - Les entrées des blocs fonctionnels utilisateur doivent être lues et ne pas être écrites.
- c, d - Les entrées/sorties des blocs fonctionnels utilisateur doivent être lues et écrites.
- e - Les sorties des blocs fonctionnels utilisateur doivent être écrites.

Explication complémentaire : Des paramètres formels non-utilisés de façon correcte sont des symptômes d'un code non-finalisé ou d'un code qui contient encore des éléments inutiles à son bon fonctionnement. Il est notable que sur certains automates, la mauvaise utilisation des paramètres peut générer des effets de bords potentiellement dangereux tels qu'une corruption de données.

> **Note:** La vérification de l'utilisation des variables privées des blocs fonctionnels utilisateur se fait par la règle S7.
> **Niveau conseillé** : Obligatoire

3.3.2 Règles spécifiques à Schneider Electric Unity

E.U.1 - L'utilisation de fonctions des librairies obsolètes d'Unity est à éviter

L'utilisation des bibliothèques obsolètes d'Unity est à éviter.

Explication complémentaire : L'utilisation des fonctions des bibliothèques obsolètes ne facilite pas le portage d'une application d'un automate à un autre. Sous l'atelier Unity, certaines librairies sont obsolètes et ont pour unique vocation d'assurer la Compatibilité ascendante avec des versions antérieures des automates Schneider. L'utilisation des fonctions de ces librairies est à éviter pour des problèmes de portabilité futur. De plus, certaines d'entre elles peuvent rendre le code non-déterministe (ex: les décalages sur les entiers signés : shl_int, shlz_int, shr_int, shrz_int, rol_int, ror_int, . . .).

Niveau conseillé : Recommandé

3.4 Structuration

3.4.1 Règles communes à tous les automates

S.1 - Les sauts en arrière sont interdits

Les sauts en arrière sont interdits.

Explication complémentaire : L'utilisation d'un saut en arrière fait courir le risque de déclencher une boucle infinie, ce qui conduit à un arrêt de l'automate. Par ailleurs, de manière générale, l'utilisation de sauts nuit à une bonne compréhension du code et donc à sa maintenabilité.

Exemple:

```
L3:
  /* code execute en permanence */

Condition
```

Itris Automation

```
toujours
vraie                          L3
-| |--------------------------->>
```

Niveau conseillé : Obligatoire

S.2 - Une variable doit être écrite au sein d'une seule routine

Une variable doit être écrite au sein d'une seule routine.

Explication complémentaire : Le respect d'une structuration simple et claire facilite la compréhension du code et donc sa maintenance. Par ailleurs, des élaborations multiples d'une variable peuvent avoir pour conséquence un comportement du code erratique non conforme aux spécifications.

Niveau conseillé : Recommandé

S.3 - Une variable doit être écrite au sein d'une seule tâche

Une variable doit être écrite au sein d'une seule tâche.

Explication complémentaire : Elaborer une variable depuis plusieurs tâches conduit à un comportement non-déterministe en conséquence du conflit d'accès potentiel. Il est nécessaire d'utiliser une autre approche pour synchroniser des données partagées entre plusieurs tâches.

Niveau conseillé : Obligatoire

S.4 - Une sortie physique ne doit être écrite qu'une seule fois par cycle automate

Une sortie physique ne doit être écrite qu'une seule fois par tour de cycle automate.

Explication complémentaire : L'écriture multiple d'une sortie peut conduire à des problèmes de sécurité de fonctionnement. Par ailleurs, un code dans lequel les sorties sont écrites plusieurs fois est moins facile à comprendre et donc à maintenir. A noter que l'écriture dans une boucle est considérée comme une écriture multiple et constitue donc une violation de cette règle.

Exemple:

```
if condition_tres_complique then
   ma_sortie_critique := TRUE;
end if;


#.... beaucoup plus loin dans le code

if une_autre_condition tres complique then
   ma_sortie_critique := FALSE;
end if;

# Dans le cas où cette sortie n'est pas commandée alors qu'elle le devrait, il est très
# difficile d'identifier quelle affectation est exécutée.
```

Niveau conseillé : Obligatoire

Itris Automation

S.5 - Les variables ne doivent pas en général être localisées

La localisation des variables doit être strictement réservée aux variables dont l'élaboration ou la consommation sont extérieures au programme : variables de communication, variables d'entrées-sorties, variables de configuration de cartes spéciales, variables de tests. En revanche toutes les variables internes du programme doivent être non localisées.

Explication complémentaire : Localiser une variable réduit les possibilités d'évolution à cause d'une organisation mémoire plus rigide. Cela réduit également la portabilité, puisque les possibilités d'organisation mémoire varient d'un automate à l'autre. Cela empêche la réutilisabilité du code car dans le cadre d'une autre application, la zone mémoire choisie peut être réservée pour d'autres usages et enfin, cela peut conduire à des erreurs d'organisation mémoire dans lesquelles plusieurs variables se trouvent localisées à la même case mémoire.

Niveau conseillé : Recommendé

S.6 - Les instances de FB/DFB sont appelées une et une seule fois

Chaque instance de bloc fonctionnel doit être appelé une fois et une seule par tour de cycle automate.

Explication complémentaire : Une instance de bloc fonctionnel possède ces propres variables qui permettent la sauvegarde de l'état de l'instance entre deux tours de cycle. Chaque exécution de l'instance doit faire évoluer ces variables pour rendre compte du changement d'état à ce tour de cycle. Une exécution multiple pour la même instance de ce code semble faire avancer les cycles plus vite que la réalité pour cette instance. En fonction du programme, cela peut conduire à des erreurs difficiles à diagnostiquer.

Enfin, une même instance peut être appelée deux fois suite à un copier/coller pas fini dans lequel les paramètres ont été modifiés mais pas le nom de l'instance.

Niveau conseillé : Obligatoire pour le programme final (Laisser sans objet pendant le développement)

S.7 - Les variables définies (hors réserves) sont utilisées

Les variables déclarées doivent être lues ou écrites à l'exception des variables qui ont été définies comme variables de réserve.

Explication complémentaire : Les variables déclarées et non utilisées sont souvent le signe d'une base de données obsolète ou pas à jour qui rend le travail de la maintenance plus compliqué.

Niveau conseillé : Obligatoire pour le programme final (Laisser sans objet pendant le développement)

S.8 - Les variables ne se chevauchent pas

Dans le cas où des variables doivent être localisées, deux variables ne doivent pas utiliser la même localisation.

Explication complémentaire : Les variables localisées qui se chevauchent sont équivalentes à un transfert implicite de valeur entre elles. Cela peut nuire à la lisibilité du code. C'est parfois un signe d'une mauvaise organisation mémoire qui peut rester invisible lors des tests et conduire à un comportement non déterministe du programme lors de l'exploitation.

Niveau conseillé : obligatoire

S.9 - Mesure de la complexité

Le code complexe cache des erreurs.

Itris Automation

Explication complémentaire : Certaines parties de programmes sont plus complexes que d'autres. C'est particulièrement vrai lorsque le flot d'exécution séquentiel du programme est interrompu. Cette règle recherche certaines causes d'interruption du flot que sont les boucles, les sauts arrière et les niveaux d'imbrications d'instructions complexe trop élevés.

Niveau conseillé : recommandé

S.12 - Instructions conditionnelle sans cas par défaut

Lorsque le programmeur utilise des instructions conditionnelles comme le IF ou le CASE en ST, il doit toujours prendre en compte le cas où toutes les conditions sont fausses et donc prévoir ce cas là explicitement. Attention cette règle ne s'applique que sur le langage littéral structuré.

Explication complémentaire : Lorsque toutes les instructions conditionnelles ont leur cas par défaut qui est prévu, le mainteneur sait que tous les chemins possibles ont été pris en compte. Dans le cas contraire, il est possible que le programmeur ait oublié un cas important. Cette règle vérifie donc que chaque instruction IF contient bien un ELSE, et de même que chaque instruction CASE contienne bien un cas OTHERS, même si il s'agit de "ELSE NULL;".

Note : Cette règle n'est utilisable qu'avec du code écrit en langage texte structuré (STL).
Niveau conseillé : Recommandé

S.13 - Grafcet sans étapes initiales

Un grafcet doit contenir une et une seule étape initiale, hormis pour les macro-étapes qui ne doivent pas en contenir.

Explication complémentaire : Utiliser plus d'une étape initiale dans le même grafcet nuit à la lisibilité du Grafcet. De même, les grafcets sans étapes initiales doivent être identifiés. Ils correspondent souvent à des cas particuliers dont le lancement est programmé à partir d'autres sections du code. La compréhension du comportement du grafcet est donc moins évidente. La maintenabilité de ces Grafcets s'en trouve altérée.

Niveau conseillé : obligatoire

S.15 - L'utilisation des valeurs immédiates est interdite

L'utilisation des valeurs immédiates est interdite.

Explication complémentaire : Il ne faut pas utiliser les valeurs immédiates dans le programme. Cela nuit à la compréhension et limite l'évolutivité du programme.

Exemple: Il ne faut pas faire:

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

Il faut faire:

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

Niveau conseillé : Recommandé

Itris Automation

S.16 - Les flottants ne doivent pas être comparés entre eux en utilisant les opérateurs d'égalité et de différence

Les flottants doivent être comparés entre eux en utilisant les opérateurs inférieur, inférieur ou égal, supérieur ou supérieur ou égal.

Explication complémentaire : La grande précision des nombres en virgule flottante fait que même si leur valeur est très proche, elle est différente. Une condition qui utiliserait les opérateurs d'égalité et de différence serait quasiment tout le temps fausse ou vraie.

Exemple: Il ne faut pas faire:

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

Il faut faire:

```
EPSILON : REAL := 0.1; (* valeur d'imprécision de la mesure *)

IF ABS(Temperature - 20.0) <= EPSILON THEN
  (* Si la différence entre la mesure et la consigne est inférieure à un seuil alors la condition
est vraie *)
...
```

Niveau conseillé : Obligatoire

S.17 - Indentation du code littéral

Lorsque le développeur écrit du code en langage Texte Structuré, celui-ci doit être correctement aligné.

Explication complémentaire : Lorsque l'indentation n'est pas respectée, le programme est plus difficile à lire et à comprendre. Une bonne indentation implique que toutes les instructions d'un même niveau de profondeur commencent à la même colonne.

Exemple: Il ne faut pas faire:

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

Il faut faire:

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

Niveau conseillé : Recommandé

3.5 Informations utiles

3.5.1 Informations utiles communes à tous les automates

I.1 - Taux de commentaires

Il est recommandé que le code contienne un pourcentage minimum de commentaires.

Explication complémentaire : A titre indicatif, Itris Automation Square recommande un taux de 30% (nombre d'instructions commentées/nombre total d'instructions). Un tel taux minimum de commentaires assure que le

Itris Automation

développement a été réalisé en ayant à l'esprit un souci de maintenabilité et de bonne compréhension par des intervenants postérieurs.

Niveau conseillé : Obligatoire

I.2 - Présence de code dans les commentaires

Il est important de vérifier la présence éventuelle de code sous forme de commentaires.

Explication complémentaire : Différentes raisons expliquent la présence de code dans des commentaires. D'une part le code est optionnel dans cette application particulière. Ou c'est un bout de programme qui ne devait pas être exécuté durant la phase de test. Ou enfin, c'est un code additionnel qui aide à mettre au point l'application. Or le code de production ne doit pas contenir de traces des phases de mise au point. Les codes restant en commentaires induisent le doute dans l'esprit d'une personne qui n'est pas l'auteur d'un programme.

Niveau conseillé : Obligatoire pour le programme final (Laisser sans objet pendant le développement)

I.3 - Présence de code mort

Le programme ne doit pas contenir de code mort.

Explication complémentaire : Le code mort nuit à la lisibilité et la maintenabilité du programme et peut être le symptôme d'un problème fonctionnel. Attention, si PLC Checker détecte les fonctions non-référencées et les parties de code isolées derrière un saut inconditionnel, aucune exhaustivité de détection n'est cependant garantie (comme, par exemple, pour le code mort lié à l'évaluation d'une expression).

Niveau conseillé : Fatal

I.4 - Liste des éléments verrouillés

Les éléments verrouillés ne sont pas analysés par le PLC Checker, il convient donc de les identifier.

Explication complémentaire : Sur des applications où la majeur partie du code est verrouillée, il est important de le savoir.

Niveau conseillé : Obligatoire

I.5 - Nombre cyclomatique v(G)

Le nombre cyclomatique donne une information relative à la complexité d'une routine. Il est important que le nombre cyclomatique reste raisonnablement bas.

Explication complémentaire : Le nombre cyclomatique représente le nombre de chemins d'exécution différents qui peuvent être exécutés dans une routine. Plus ce nombre est élevé, plus il est difficile de valider le comportement correct de cette routine. Il est alors nécessaire de diviser celles-ci en d'autres sous-routines de nombre cyclomatique inférieur au seuil.

Niveau conseillé : info

Itris Automation

I.6 - Complexité essentielle ev(G)

La complexité essentielle donne une information relative à l'utilisation d'instructions non structurées. Il est important que la complexité essentielle reste raisonnablement bas.

Explication complémentaire : La complexité essentielle représente le nombre d'instructions non structurée appartenant à une routine. Plus ce nombre est élevé, plus il est difficile de valider le comportement correct de cette routine. La plupart du temps il est possible de remplacer des instructions non structurées par des instructions structurées.

Niveau conseillé : Info

I.7 - Nombre de lignes de code

Le nombre de ligne de code d'une procédure ou d'un bloc fonction est une métrique très simple qu'il convient de garder sous contrôle.

Explication complémentaire : De très grosses procédures ou blocs fonctions ont de mauvaises propriétés de test, validation, lisibilité et maintenance.

Niveau conseillé : info

I.8 - Nombre d'étapes du Grafcet

Il s'agit du nombre d'étapes Grafcet présents.

Explication complémentaire : Cette règle retourne le nombre d'étapes trouvé dans un grafcet de l'application. Ce nombre peut servir à détecter des grafcets plus complexes parmi tous les grafcets d'une application.

Niveau conseillé : Info

I.9 - Nombre de branches du Grafcet

Le nombre est la somme de nombre de branches de chacune des divergences ET/OU trouvées dans un grafcet donné.

Explication complémentaire : Cette règle calcule le nombre de branches d'un Grafcet. Ce nombre donne des informations sur la complexité d'un grafcet donné. Ce nombre est équivalent à la complexité cyclomatique (v(G)) calculé pour les langages de programmation impératifs.

Niveau conseillé : Info

I.10 - Ratio de code dupliqué sur l'application

Ce nombre détecte le pourcentage de code qui est dupliqué.

Explication complémentaire : Réutiliser le code en le copiant/collant n'est pas une bonne pratique. La maintenance et l'évolution du programme deviennent plus difficiles car les modifications doivent être appliquées à plusieurs endroits. De plus l'opération manuelle consistant à Copier/Coller est souvent responsable d'erreurs difficile à détecter : la phase de modification qui suit le copier/coller est parfois incomplète.

Niveau conseillé : Info

itris Automation

I.11 - Métriques de Halstead

Toutes les métriques de Halstead sont données par cette règle. Il s'agit des différentes métriques proposées par Maurice Halstead (Nombre d'opérateurs, Nombre d'opérandes, Volume du programme, Niveau de difficulté, . . .).

Explication complémentaire : Ces métriques permettent entre autres d'obtenir une estimation du nombre d'erreurs fournies dans le programme au moment de sa livraison.

Niveau conseillé : Info

I.12 - Nb Références externes au bloc

Toutes les métriques de Halstead sont données par cette règle. Il s'agit des différentes métriques proposées par Maurice Halstead (Nombre d'opérateurs, Nombre d'opérandes, Volume du programme, Niveau de difficulté, . . .).

Explication complémentaire : Ces métriques permettent entre autres d'obtenir une estimation du nombre d'erreurs fournies dans le programme au moment de sa livraison.

Niveau conseillé : Info

I.13 - Nb d'appels du bloc

Le nombre d'appels de chaque routine/POU dans un programme.

Explication complémentaire : Hormis le fait qu'un bloc appelé zéro fois correspond à un code mort, il n'y a pas de limite pour le nombre d'appels d'un bloc de code. Ce nombre permet d'identifier les blocs réutilisés.

Niveau conseillé : Info

I.14 - Nb d'appels à d'autres blocs

Cette règle donne le nombre de routines/POU appelées depuis une routine/POU donnée.

Explication complémentaire : Un bloc de code ne faisant pas référence à d'autres blocs correspond à un bloc de code autonome qui se suffit à lui-même. Lorsque ce nombre est élevé, cela permet d'identifier un bloc qu'il conviendrait de découper en plusieurs autres blocs afin d'améliorer la structuration du programme.

Niveau conseillé : Info

I.15 - Profondeurs d'appels du bloc

Il s'agit des profondeurs min et max d'appel d'un bloc de code (POU) donné.

Explication complémentaire : Ces mesures permettent d'identifier à quel niveau de profondeur se situent les appels du bloc dans la structure du programme. Un bloc de code peut être appelé au niveau d'une tâche, dans ce cas la profondeur est faible, ou bien être appelé plus profondément dans la structure du programme, auquel cas la profondeur est élevée. Un bloc de code peut être appelé plusieurs fois dans le programme et peut donc avoir plusieurs niveaux de profondeurs. Un niveau faible correspond à un bloc de gestion ou d'organisation du programme. Un niveau élevé, correspond à un bloc de code spécialisé.

Niveau conseillé : Info

Iris Automation

3.6 Options

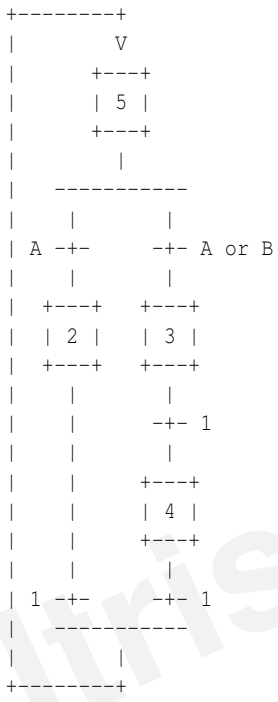
3.6.1 Options Unity des projets automates

O.U.1 - SFC : Le multi-jetons sur le Grafcet est interdit

Sous l'atelier Unity, dans les options du projet, il faut décocher la possibilité de faire tourner le grafcet en mode multi-jetons.

Explication complémentaire : Le mode multi-jetons sous Unity n'a que pour vocation d'assurer la compatibilité ascendante avec des versions antérieures des automates Schneider. Il ne doit pas être utilisé dans le cadre de nouveaux projets car il peut rendre le code non-déterministe.

Exemple : Si le code Grafcet suivant est exécuté avec une condition A toujours vraie, après plusieurs tours de cycles, toutes les étapes sont actives simultanément.



Itris Automation

Niveau conseillé : fatale

O.U.2 - SFC : Non maintien des étapes précédentes actives pour SetSteps

Sous l'atelier Unity, dans les options du projet, il faut décocher l'option "SetSteps : maintient des étapes précédentes actives".

Explication complémentaire : Cette option est liée au mode multi-jetons voir O.U.1.

Niveau conseillé : Recommandé

O.U.3 - SFC : Pas de saut In/Out dans les divergences en ET

Sous l'atelier Unity, dans les options du projet, il faut décocher l'option "Divergence en ET : autoriser saut In/Out".

Explication complémentaire : Cette option est liée au mode multi-jetons voir O.U.1.

Niveau conseillé : Recommandé

O.U.4 - SFC : Une seule évolution par divergence

Sous l'atelier Unity, dans les options du projet, il faut décocher l'option "Autoriser plusieurs évolutions par divergence".

Explication complémentaire : Cette option est liée au mode multi-jetons voir O.U.1.

Niveau conseillé : Recommandé

O.U.5 - ST : Sauts et labels interdits

Sous l'atelier Unity, dans les options du projet, il faut décocher l'option "Autoriser les sauts et les labels".

Explication complémentaire : De manière générale, l'utilisation de saut nuit à une bonne compréhension du code et donc à sa facilité de maintenance.

Niveau conseillé : fatal

O.U.6 - Pas de chiffre en début d'identificateur

Sous l'atelier Unity, dans les options du projet, il faut décocher l'option "Chiffres en début autorisés".

Explication complémentaire : De manière générale, l'utilisation de chiffres en début de mnémoniques nuit à la lisibilité et à la portabilité du programme.

Niveau conseillé : fatal

O.U.7 - Jeu de caractères standard pour les identificateurs

Sous l'atelier Unity, dans les options du projet, il faut sélectionner le jeu de caractères Standard (et pas Etendu ou Unicode) pour les identificateurs.

Explication complémentaire :De manière générale, l'utilisation d'un jeu de caractères non standard nuit à la lisibilité et à la portabilité du programme.

Exemple : Utiliser par exemple les accents des alphabets latins nuit à la lisibilité lorsque deux variables du même nom l'une avec les accents et l'autre sans sont utilisées. Un exemple tiré de la vraie vie comprend deux variables, l'une nommée 'départ' et l'autre nommée 'depart'.

Niveau conseillé : fatal

O.U.8 - Les expressions ST sont interdites dans le FBD et le LD

Sous l'atelier Unity, dans les options du projet, il faut décocher l'option "Utilisation d'expressions ST".

Explication complémentaire : De manière générale, l'utilisation d'expression ST dans les langages FBD et LD nuit à la lisibilité du programme.

Niveau conseillé : Recommandé

Itris Automation

O.U.9 - Les informations d'upload avec commentaires et tables d'animation

Sous l'atelier Unity, dans les options du projet, il faut :

1. cocher l'option "Avec" les informations d'upload,
2. cocher l'option "Commentaires (variables et types)" pour les informations d'upload,
3. cocher l'option "Tables d'animation" pour les informations d'upload

Explication complémentaire : Dans la mesure où la mémoire automate le permet, le stockage des informations d'upload le plus complet possible permet de retrouver ces informations en cas de perte de la sauvegarde du programme, ce qui améliore la maintenabilité à long terme.

Niveau conseillé : fatal

O.U.10 - Les bobines en LD sont alignées à droite

Sous l'atelier Unity, dans les options du projet, il faut cocher l'option "Bobines alignées à droite" pour l'éditeur Ladder.

Explication complémentaire : L'alignement des bobines sur la droite de l'éditeur permet une meilleur lisibilité du programme.

Niveau conseillé : fatal

O.U.12 - L'UC de l'automate doit être configurée pour démarrer automatiquement en RUN

Sous l'atelier Unity, le module d'UC de l'automate peut être configuré pour démarrer automatiquement en RUN ou non après une mise sous tension. Cette règle vérifie que l'UC est configurée pour démarrer automatiquement en RUN. Cela permet d'éviter que l'automate soit bloqué en STOP lors d'une remise sous tension après perte d'alimentation.

Explication complémentaire :

Niveau conseillé : fatal

3.7 Langages

3.7.1 Vérification de langage de programmation

L.1 - Le langage LIST est interdit

Il est recommandé d'éviter l'utilisation du langage list car étant un langage de bas niveau, il est peu lisible.

Explication complémentaire : Selon le type d'automate, ce langage de liste d'instructions est plus ou moins facile à éviter. En effet, pour un programme Step5 ou Step7, les langages CONT/LIST/LOG sont en fait enregistrés en langage LIST. Par contre pour les automates Unity Pro et RSLogix5000, ce langage, nommé IL, peut être évité.

Niveau conseillé : Fatal

itris Automation

Chapter 4

Annexes

4.1 Résumé des règles

Cette annexe récapitule toutes les règles et informations utiles mentionnées dans ce document.

Règles de Nommage

- N.1 - Tout type d'automates - Tous les éléments constituant le programme doivent être nommés
- N.2 - Tout type d'automates - Les noms des éléments du programme doivent avoir une taille minimum et maximum
- N.3 - Tout type d'automates - Les mnémoniques des éléments ne font pas référence à leur adresse physique
- N.4 - Tout type d'automates - Les mnémoniques de mise au point ou inconnus ne doivent pas faire partie du programme final
- N.5 - Tout type d'automates - Les caractères spéciaux sont interdits dans les noms des mnémoniques
- N.6 - Tout type d'automates - Les mots clefs des langages de programmation ne doivent pas servir de mnémoniques

Règles sur les Commentaires

- C.1 - Tout type d'automates - Tous les éléments constituant le programme doivent être commentés
- C.2 - Tout type d'automates - Les commentaires doivent comporter un minimum de caractères
- C.3 - Tout type d'automates - Chaque réseau doit être commenté
- C.4 - Tout type d'automates - Les commentaires ne contiennent pas le marqueur de début de commentaire

Règles d'Écriture

- E.1 - Tout type d'automates - Toutes les variables doivent être écrites avant d'être lues
- E.2.a, E.2.b - Tout type d'automates - Les entrées des blocs fonctionnels utilisateur doivent être lues et ne pas être écrites
- E.2.c, E.2.d - Tout type d'automates - Les entrées/sorties des blocs fonctionnels utilisateur doivent être lues et écrites
- E.2.e - Tout type d'automates - Les sorties des blocs fonctionnels utilisateur doivent être écrites avant leur éventuelle lecture
- E.U.1 - SE Unity - L'utilisation de fonctions des librairies obsolètes d'Unity est à éviter

Iris Automation

Règles de Structuration

- S.1 - Tout type d'automates - Les sauts en arrière sont interdits
- S.2 - Tout type d'automates - Une variable doit être écrite au sein d'une seule routine
- S.3 - Tout type d'automates - Une variable doit être écrite au sein d'une seule tâche
- S.4 - Tout type d'automates - Une sortie physique ne doit être écrite qu'une seule fois par cycle automate
- S.5 - SE Unity - Les variables ne doivent pas en général être localisées
- S.6 - Tout type d'automates - Les instances de FB/DFB sont appelées une et une seule fois
- S.7 - Tout type d'automates - Les variables définies sont utilisées
- S.8 - Tout type d'automates - Les variables ne se chevauchent pas
- S.9 - Tout type d'automates - Mesure de la complexité
- S.12 - Tout type d'automates - Les instructions conditionnels gèrent le cas par défaut
- S.13 - Tout type d'automates - Les Grafcets doivent avoir une et une seule étape initiale
- S.15 - Tout type d'automates - L'utilisation des valeurs immédiate est interdite
- S.16 - Tout type d'automates - Les flottants ne doivent pas être comparés entre eux en utilisant les opérateurs d'égalité et de différence
- S.17 - Tout type d'automates - Indentation du code littéral

Informations Utiles

- I.1 - Tout type d'automates - Taux de commentaires
- I.2 - Tout type d'automates - Présence de code dans les commentaires
- I.3 - Tout type d'automates - Présence de code mort
- I.4 - Tout type d'automates - Présence d'éléments verrouillés
- I.5 - Tout type d'automates - Nombre cyclomatique v(G)
- I.6 - Tout type d'automates - Complexité essentielle ev(G)
- I.7 - Tout type d'automates - Nombre de lignes de code
- I.8 - Tout type d'automates - Nombre d'étapes du Grafcet
- I.9 - Tout type d'automates - Nombre de branches du Grafcet
- I.10 - Tout type d'automates - Ratio de code dupliqué sur l'application
- I.11 - Tout type d'automates - Métriques de Halstead
- I.12 - Tout type d'automates - Nb de Références externes au bloc
- I.13 - Tout type d'automates - Nb d'appels du bloc
- I.14 - Tout type d'automates - Nb d'appels à d'autres blocs
- I.15 - Tout type d'automates - Profondeurs d'appels du bloc

itris Automation

Options

- O.U.1 - SE Unity - SFC : Le multi-jetons sur le Grafcet est interdit
- O.U.2 - SE Unity - SFC : Non maintien des étapes précédentes actives pour SetSteps
- O.U.3 - SE Unity - SFC : Pas de saut In/Out dans les divergences en ET
- O.U.4 - SE Unity - SFC : Une seule évolution par divergence
- O.U.5 - SE Unity - ST : Sauts et labels interdits
- O.U.6 - SE Unity - Pas de chiffre en début d'identificateur
- O.U.7 - SE Unity - Jeu de caractères standard pour les identificateurs
- O.U.8 - SE Unity - Les expressions ST sont interdites dans le FBD et le LD
- O.U.9 - SE Unity - Les informations d'upload avec commentaires et tables d'animation
- O.U.10 - SE Unity - Les bobines en LD sont alignées à droite
- O.U.12 - SE Unity - L'UC de l'automate doit être configurée pour démarrer automatiquement en RUN

Langages

- L.1 - Tout type d'automates - Le langage LIST est à éviter

4.2 Historique du document

Date	Changements
9 juin 2009	Version initiale
26 octobre 2010	Ajout de la section sur les options
9 novembre 2011	Ajout les descriptions pour les règles S5, S6, S7, S8, S9, S10, I5, I6, I7, I8, I9, I10
21 novembre 2011	Corrections texte dans Annexes et C4
19 janvier 2012	Relecture et corrections (ortho, gramm) par Pauline
27 novembre 2012	Ajout des règles S12 et L1 par DSA
27 Janvier 2014	Ajout de la règle S13 et complétion de N4 par DSA
13 Novembre 2015	Relecture et corrections (ortho) par DSA&LHU
17 Novembre 2016	Ajout des règles S15, S16 et OU12. Relecture et corrections en se basant sur la version 3.1.1 du fichier GQR. (par WAI)
08 Octobre 2018	Ajout des règles S17 et I11 à I15 (par DSA)
22 Janvier 2019	Relecture et corrections(par RBU)
31 Janvier 2019	Automatisation du document via liens vers wiki du catalogue de règles (par SLE)
14 Mars 2019	Ajout de la présentation d'IAS et de SE dans L'objectif de ce document
10 Avril 2019	Retrait de la règle S10 dans le résumé en annexe en fin de document (par SLE)

Itris Automation