

Manuel de référence de PLC Checker

Itris Automation Square- janvier 2019

www.itris-automation.com

©Itris Automation

2008-2019

IAS

Contents

1 Préambule **4**

2 Introduction **5**

3 Prise en main d'Eclipse **6**
3.1 Installation . . . 6
3.1.1 Configuration logicielle et matérielle requise . . . 6
3.1.2 Mode d'utilisation . . . 6
3.1.3 Installation de la plate-forme . . . 7
3.2 Mise à jour logicielle . . . 11
3.3 Lancement de la plate-forme . . . 15
3.3.1 Espace de travail . . . 15
3.3.2 Identification . . . 16
3.3.3 Plan de travail . . . 16
3.4 Importation/Exportation de fichiers dans le projet . . . 19
3.4.1 Drag'N Drop . . . 19
3.4.2 Importation . . . 19
3.5 Comparaison de fichiers . . . 24
3.6 Historique local . . . 26
3.6.1 Comparer avec une version précédente . . . 26
3.6.2 Remplacer par une version précédente . . . 27

4 Création et utilisation d'un projet PLC Checker **28**
4.1 Création d'un projet PLC Checker . . . 29
4.1.1 Création avec assistant d'une application PLC Checker par défaut sur le site d'Itris Automation Square (Projet Plc Che
4.1.2 Création avec assistant d'une application de type PLC Checker sur le site d'Itris Automation Square (Projet Plc Check
4.1.3 Accès à un programme existant de type PLC Checker sur le site d'Itris Automation Square . . . 37
4.1.4 Création d'un nouveau projet PLC Checker dans la plateforme Eclipse . . . 39
4.1.5 Création d'un projet PLC Checker existant à partir de la plateforme Eclipse . . . 41
4.2 Exécution de PLC Checker . . . 43
4.2.1 Exécution de PLC Checker à partir de la plateforme Eclipse . . . 43
4.2.2 Exécution de PLC Checker à partir du site web : . . . 47
4.3 Visualisation des résultats PLC Checker . . . 49
4.3.1 Visualisation des résultats sur la plateforme Eclipse . . . 49
4.3.2 Visualisation des résultats sur le site web . . . 52

5 Export des fichiers automates **54**
5.1 Création d'un fichier FEF PL7-Pro . . . 54
5.2 Création d'un fichier L5K RSLogix5000 . . . 55
5.3 Création des fichiers .ASC, .AWL, .GR7 et .SCL Step7 . . . 55
5.3.1 Export du Fichier .ASC . . . 56
5.3.2 Génération du Fichier .AWL . . . 59
5.3.3 Génération des Fichiers .GR7 . . . 62
5.3.4 Génération sources SCL pour codes CFC . . . 64

5.3.5 Cas d'une application Safety . . . 65
5.3.6 Export des fichiers .AWL, .GR7 et .SCL . . . 67
5.3.7 Import des Fichiers . . . 67
5.4 Création d'un fichier XEF Unity . . . 68
5.5 Récupération du fichier SMC pour OMRON Sysmac Studio . . . 68
5.6 Création d'un fichier d'export Codesys v3.x . . . 68
5.7 Création d'un fichier EXP Codesys V2 . . . 70
5.8 Création d'un fichier EXPORT SoMachine . . . 70
5.9 Création d'un fichier XML Beckhoff Twincat . . . 72
5.10 Création d'un fichier EXP Beckhoff twincat V2 . . . 73
5.11 Création d'un fichier PC5 RSLogix 5 . . . 74
5.12 Export d'un projet Tia Portal . . . 75
5.12.1 Présentation générale de l'application . . . 75
5.12.2 Pré-requis pour l'installation . . . 75
5.12.3 Unité de base . . . 75
5.12.4 Modules . . . 76
5.12.5 Fonctionnement . . . 76
5.12.6 Installation de Openness . . . 77
5.12.7 Format du projet TIA Portal . . . 79
5.13 Création d'un fichier XML Phoenix Contact Multiprog . . . 79

6 Itris Import/Export Tool 82
6.1 Présentation . . . 82
6.2 Installation . . . 83
6.3 Prise en main . . . 84
6.3.1 Interface graphique pour les opérations du quotidien . . . 84
6.3.2 Interface en ligne de commandes pour les opérations automatisées . . . 85
6.4 Visualiser le contenu des exports . . . 86
6.5 Visualiser les plugins installés . . . 87

7 Itris Import/Export Tool - STEP 7 88
7.1 Présentation de Itris Import/Export Tool . . . 88
7.2 Prérequis spécifiques à STEP 7 . . . 88
7.3 Avertissements spécifiques à STEP 7 . . . 88
7.3.1 Langages de programmation . . . 88
7.3.2 Interface en ligne de commandes . . . 89
7.3.3 Compatibilité avec les anciennes versions de PLC Checker, PLC Docgen et PLC Converter . . . 89

8 Itris Import/Export Tool - TIA Portal 90
8.1 Présentation de Itris Import/Export Tool . . . 90
8.2 Liste des plugins disponibles pour TIA Portal . . . 90
8.3 Prérequis spécifiques à TIA Portal . . . 90
8.4 Installation des plugins . . . 91
8.4.1 Installation de Openness . . . 91
8.4.2 Autorisation des utilisateurs . . . 91
8.4.3 Fichiers d'autorisation . . . 92
8.5 Avertissements spécifiques à TIA Portal . . . 92
8.5.1 Possibilité d'exporter les codes non compilés . . . 92
8.5.2 Possibilité d'exporter les codes protégés . . . 92
8.5.3 Versions des projets . . . 93
8.5.4 Autorisation du plugin . . . 94

9 Itris Import/Export Tool - Unity **95**
9.1 Présentation de Itris Import/Export Tool . . . 95
9.2 Prérequis spécifiques à Unity . . . 95
9.3 Avertissements spécifiques à Unity . . . 95
9.3.1 Compatibilité avec les anciennes versions de PLC Checker, PLC Docgen et PLC Converter . . . 95

10 Concepts **96**
10.1 Règles . . . 96
10.1.1 Généralités . . . 96
10.1.2 Activation des règles . . . 96
10.1.3 Nom de règles . . . 96
10.1.4 Identificateur de règles . . . 97
10.1.5 Filtre de code . . . 97
10.1.6 Filtre d'éléments . . . 97
10.1.7 Validation d'une règle ou d'un ensemble de règles à l'aide d'autres règles . . . 97
10.1.8 Validation d'une règle ou d'un ensemble de règles parmi la liste de règles . . . 98
10.1.9 Identificateur pour l'export au format .csv . . . 98
10.1.10 Visibilité du nœud . . . 98
10.1.11 Modification automatique de la visibilité du noeud . . . 98
10.1.12 Liste des propriétés éditables lorsque le fichier est protégé . . . 98
10.1.13 Sévérité des erreurs . . . 98
10.1.14 (propriété "warningThreshold") . . . 99
10.1.15 (propriété "errorThreshold") . . . 99
10.1.16 (propriété "defaultResultId") . . . 99
10.1.17 (propriété "errorMessage") . . . 99
10.1.18 (propriété "noMatchMessage") . . . 100
10.1.19 (propriété "noMatchSeverity") . . . 100
10.1.20 (propriété "isTextEditable") . . . 100
10.1.21 Générateur de valeur d'un attribut . . . 100
10.2 Filtres . . . 101
10.2.1 Généralités . . . 102
10.2.2 Limitations . . . 102
10.2.3 Filtres de symboles . . . 102
10.2.4 Filtres de symboles utilisant les expressions régulières . . . 102
10.2.5 Filtres d'attributs . . . 104
10.2.6 Filtres d'adresses absolues . . . 104
10.2.7 Réutilisation d'un filtre . . . 104
10.2.8 Intersection de filtres . . . 105
10.2.9 Union de filtres . . . 105
10.2.10 Différence de filtres . . . 105
10.3 Attributs Glips . . . 105
10.4 Fichier résultat . . . 114
10.4.1 Impression du fichier résultat . . . 114

11 Les règles **115**
11.1 Groupe de règles (rulesGroup) . . . 115
11.1.1 Utilisation . . . 115
11.1.2 Propriétés . . . 116
11.2 Vérification des lectures (readCheck) . . . 116
11.2.1 Utilisation . . . 116
11.2.2 Propriétés . . . 116
11.3 Vérification des écritures (writeCheck) . . . 117
11.3.1 Utilisation . . . 117
11.3.2 Propriétés . . . 117
11.4 Vérification des accès (accessCheck) . . . 120
11.4.1 Utilisation . . . 120

11.4.2 Propriétés . . . 120
11.5 Vérification des symboles (symbolCheck) . . . 121
11.5.1 Utilisation . . . 121
11.5.2 Propriétés . . . 121
11.6 Vérification des attributs (attributeCheck) . . . 122
11.6.1 Utilisation . . . 122
11.6.2 Propriétés . . . 122
11.7 Vérification des blocs de code (blockCheck) . . . 123
11.7.1 Utilisation . . . 123
11.7.2 Propriétés . . . 123
11.8 Vérification d'équation (equationCheck) . . . 124
11.8.1 Utilisation . . . 124
11.8.2 Propriétés . . . 124
11.9 Vérification de mémoire (memoryCheck) . . . 125
11.9.1 Utilisation . . . 126
11.9.2 Propriétés . . . 126
11.10 Vérification de copier/coller (copyPastCheck) . . . 126
11.10.1 Utilisation . . . 127
11.10.2 Propriétés . . . 127
11.11 Vérification des structures dans le fichier source (FEFStructureCheck) . . . 127
11.11.1 Utilisation . . . 127
11.11.2 Propriétés . . . 127
11.12 Vérification des attributs dans le fichier source (FEFAttributeCheck) . . . 128
11.12.1 Utilisation . . . 128
11.12.2 Propriétés . . . 128
11.13 Vérification des labels (labelCheck) . . . 129
11.13.1 Utilisation . . . 129
11.13.2 Propriétés . . . 129
11.14 Vérification de correspondance de modèle (patternMatchingCheck) . . . 130
11.14.1 Utilisation . . . 130
11.14.2 Propriétés . . . 130
11.15 Duplication automatique de règles (duplicate) . . . 131
11.15.1 Lancement de la duplication . . . 131
11.15.2 Définitions . . . 132
11.15.3 Syntaxe . . . 132
11.15.4 Fonctionnement . . . 132
11.15.5 Exemples . . . 134

12 Création et édition du fichiers de règles **138**
12.1 Création d'un fichier de règles . . . 138
12.2 Ajout d'une règle . . . 140
12.3 Annulation des modifications . . . 140
12.4 Copier et coller des règles . . . 141
12.5 Suppression de règles . . . 141
12.6 Opérations relatives aux propriétés . . . 141
12.7 Impression du fichier de règles . . . 141

13 Contrat de Licence Utilisateur Final pour logiciel PLC Checker **142**

IAS

Chapter 1

Préambule

Ce document est le guide de l'utilisateur du PLC Checker. Après une brève introduction concernant le sujet, il détaille la prise en main de la plateforme du PLC Checker (installation, mise à jours, lancement,...).
Il aborde ensuite les opérations liées à la création et l'édition du fichiers de règles.
Le chapitre suivant détaille la création des fichiers d'export des plate-formes automate respectives.
La création de projet, l'exécution du PLC Checker et la visualisation des résultats sont ensuite présentés.
Vient ensuite un chapitre théorique, qui introduit les concepts de règles, de filtres et qui décrit le fichier résultat.
Les deux derniers chapitres présentent les références des règles et le lien vers le Contrat de Licence Utilisateur Final.

IAS

Chapter 2

Introduction

PLC Checker analyse du code automate, en vérifiant que celui-ci respecte les règles de conception et de codage décrites par un fichier de règles. A l'issue de la vérification, un fichier "résultat" est généré. Les non-respects des règles sont consignés dans ce fichier résultat.
Cette version supporte les programmes automates écrits avec les ateliers PL7 PRO, Step 7, Unity, RSLogix5000, CodeSys, Step5, PLC Open, SMC.
Ce guide d'initiation décrit les étapes à suivre pour utiliser PLC Checker en mode SaaS (Software as a Service).

PLC Checker a obtenu le Grand Prix de l'Innovation 2004, organisé par AUTOMATION-OPTIMATION EUROPE dans la catégorie "Meilleur logiciel d'automatisme et de gestion industrielle".

IAS

Chapter 3

Prise en main d'Eclipse

L'objectif de ce tutoriel est d'installer et de prendre en main la plate-forme Eclipse.

Nous détaillerons l'installation d'Eclipse, les mises à jour, définirons les concepts d'espaces de travail, de plan de travail, de projets et enfin nous détaillerons les fonctionnalités d'Eclipse les plus importantes.

3.1 Installation

Cette section contient les procédures d'installation de la plateforme Eclipse.

3.1.1 Configuration logicielle et matérielle requise

L'atelier d'Itris Automation Square est intégré dans la plateforme Eclipse 3.7.2. (PLC Checker est pré-installé).
Cette configuration logicielle requiert un PC ayant les caractéristiques minimales suivantes :

- Microsoft Windows 2000/XP Professionnel/ Windows 7,
- Un processeur de fréquence minimum de 1GHz,
- 200 Mo d'espace disque disponible,
- 512 MB[1] de RAM

3.1.2 Mode d'utilisation

Attention, l'accessibilité aux produits d'Itris Automation Square est payante via l'utilisation en mode SaaS (Software as a Service) et est conditionnée par une validation commerciale préalable. Dans le cas où vous rencontreriez des problèmes, veuillez contacter notre commercial@automationsquare.com.

La création d'un compte utilisateur sur le site d'Itris Automation Square est indispensable pour l'utilisation des produits d'Itris Automation Square.

Pour utiliser les produits d'Itris Automation Square, l'utilisateur doit s'identifier, lorsque cela lui est demandé, en saisissant, dans la fenêtre prévue à cet effet, son login et son mot de passe définits lors de son inscription sur notre site .

[1] Megabyte

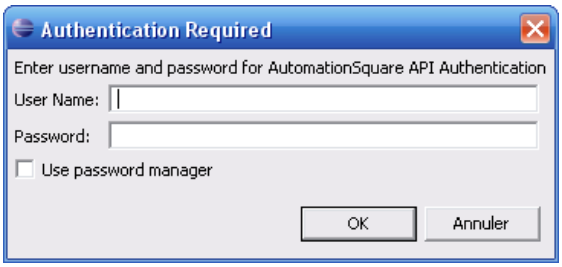


Figure 3.1: Fenêtre d'identification

3.1.3 Installation de la plate-forme

Le package PLC Checker pré-installé dans Eclipse 3.7.2 est disponible (voir ci-dessous) pour les ordinateurs de type PC (voir configuration requise ci-dessus).
Pour les autres types d'ordinateurs (Mac, Linux, ...), il est nécessaire d'installer la version Eclipse Classic 3.7.2 à partir du site http://www.eclipse.org/, puis d'installer les fonctionnalités PLC Checker à partir d'Eclipse.

Installeur du package PLC Checker dans Eclipse 3.7.2

Le programme d'installation de PLC Checker est téléchargeable ICI, et se nomme InstallPlcChecker.exe. Pour installer le logiciel, cliquez sur InstallPlcChecker.exe. Le programme demandera le répertoire où installer le logiciel et effectuera l'installation sans autre intervention.

Le programme d'installation créé une icône sur le bureau et une entrée dans le menu démarrer de Windows.
Procédure détaillée :

- Choisir la langue de l'installateur



Figure 3.2: Choix de la langue

- Au message de bienvenue, cliquer sur **Suivant**

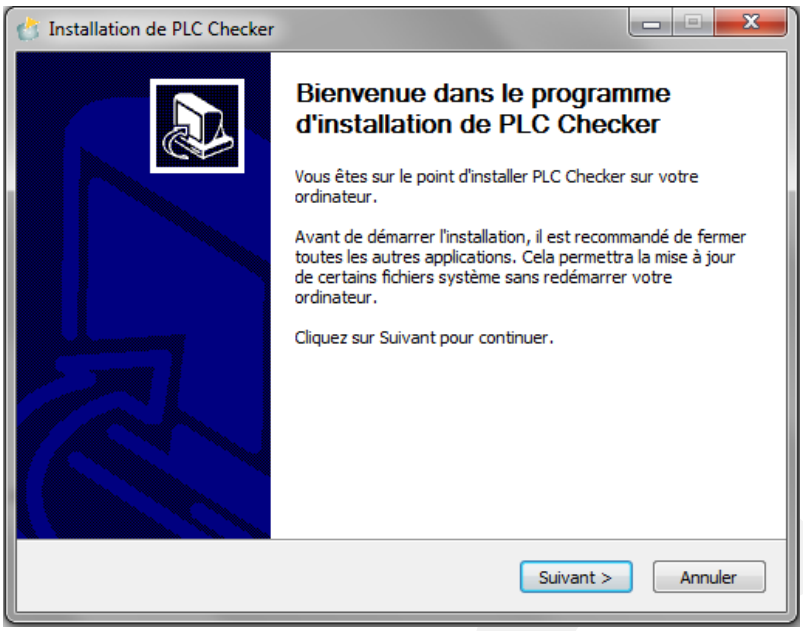


Figure 3.3: Bienvenue du programme d'installation

- Pour pouvoir installer la plateforme du PLC Checker, il faut accepter la licence, en cliquant sur **Accepter**.

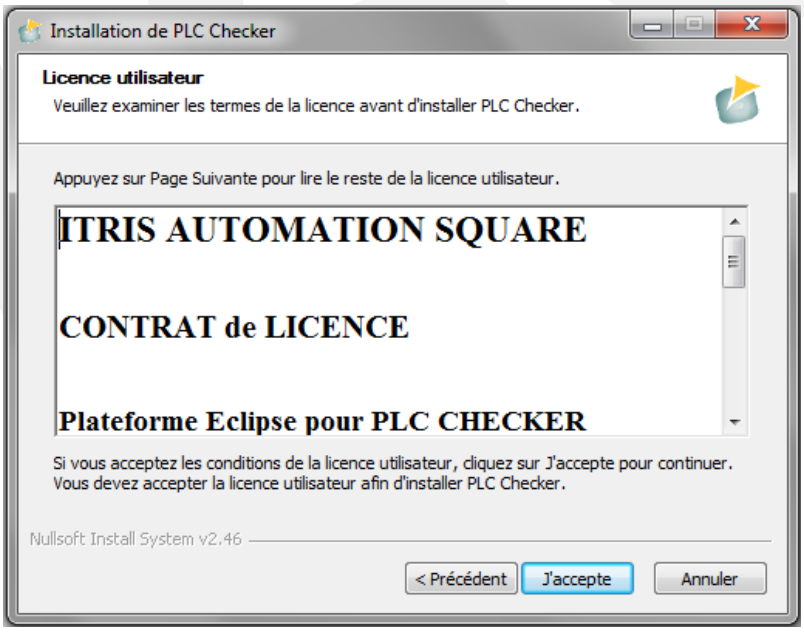


Figure 3.4: Licence

- Choisir le dossier d'installation (par défaut C:\IAS), puis cliquer sur **Installer**

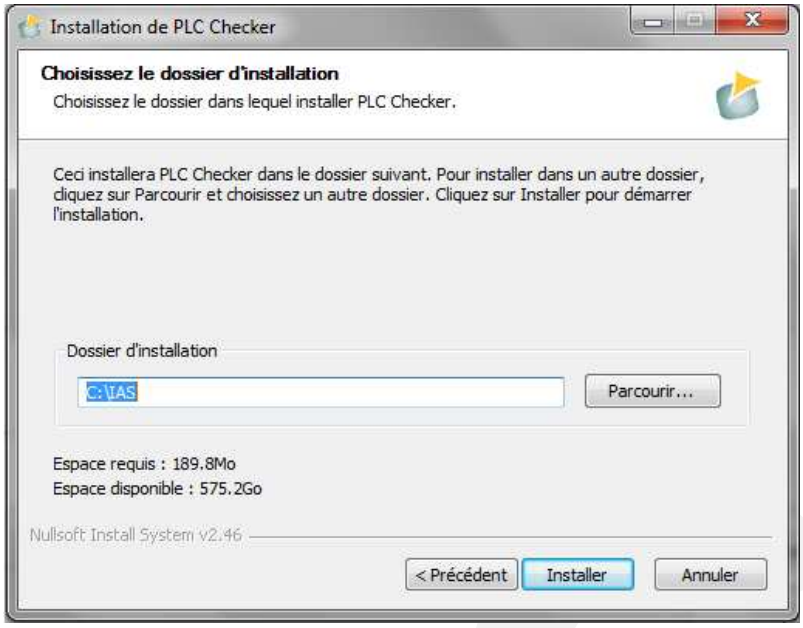


Figure 3.5: Choix de dossier d'installation

Pensez à changer de répertoire si ce dernier est déjà utilisé par exemple pour PLC Converter

- La plateforme de PLC Checker s'installe

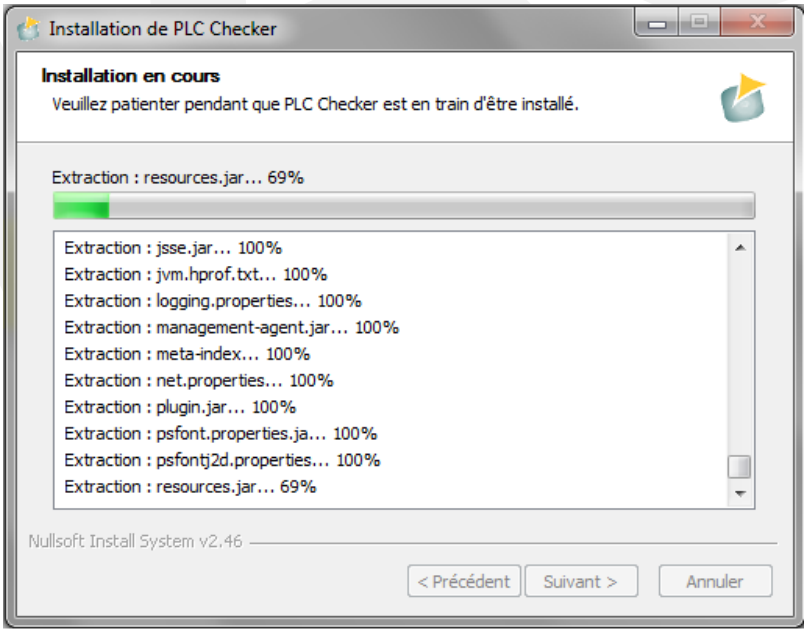


Figure 3.6: Installation

- A la fin de l'installation, cliquer sur **Fermer**.

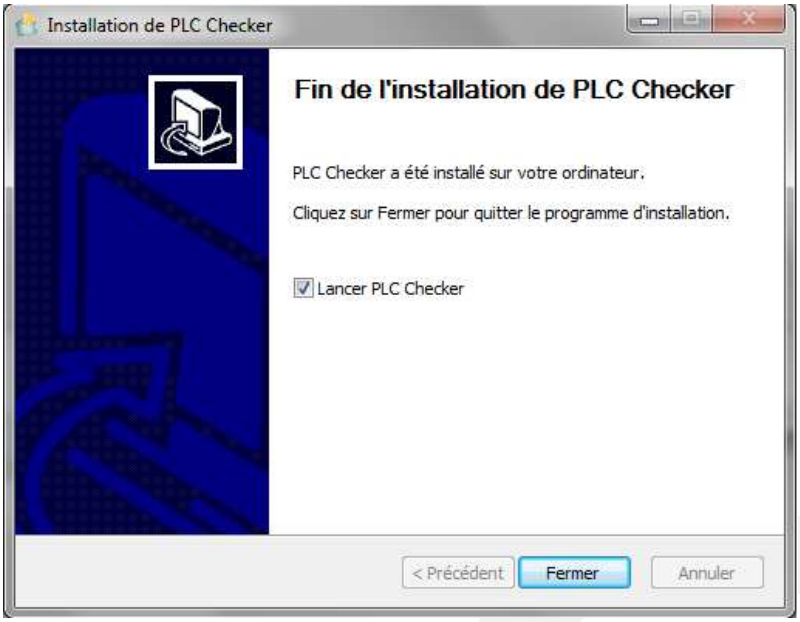


Figure 3.7: Fermeture du programme d'installation

Installer Eclipse 3.7.2 puis PLC Checker

- Installer la version Eclipse Classic 3.7.2 à partir du site http://www.eclipse.org/ pour votre type d'ordinateur (Mac, Linux, . . .)
- Installer les plugins d'Itris Automation Square pour le PLC Checker de la façon suivante :
  - Lancer la plateforme Eclipse Classic 3.7.2,
  - Ouvrir le menu Aide / Installer de nouveaux logiciels/composants. . .
  - Dans le champs "work with", entrer l'adresse du site de mise à jours d'Itris Automation Square: http://www.automationsquare
  - Dans la liste des logiciels disponibles qui s'affichent en dessous, sélectionner ces 2 éléments :
    - Atelier GLIPS
    - Qualimètre GLIPS
  - Lancer le téléchargement en cliquant sur : suivant / suivant / terminer
  - Installer les plugins téléchargés en cliquant sur "Installer tout".
  - Redémarrer la plateforme après installation

3.2 Mise à jour logicielle

La recherche et l'installation des mises à jour de la plateforme se fait automatiquement à chaque démarrage de cette dernière.

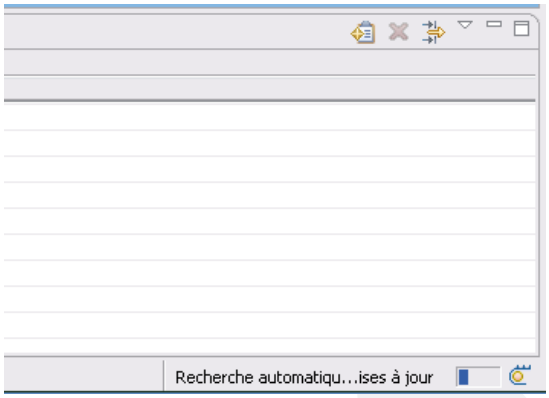


Figure 3.8: Recherche automatique des mises à jour

- Lorsque des mises à jour sont trouvées, la plateforme vous demande si vous voulez les consulter et les installer.

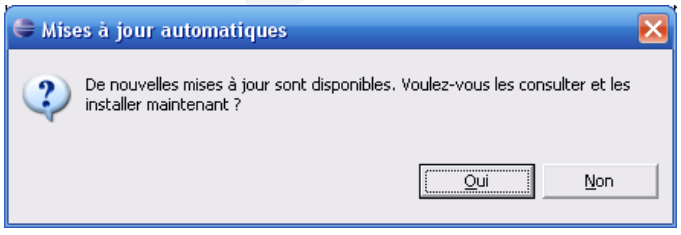


Figure 3.9: Demande d'installation

- La plateforme vous permet de consulter les mises à jour. Laissez tout sélectionné et cliquez sur "Suivant".

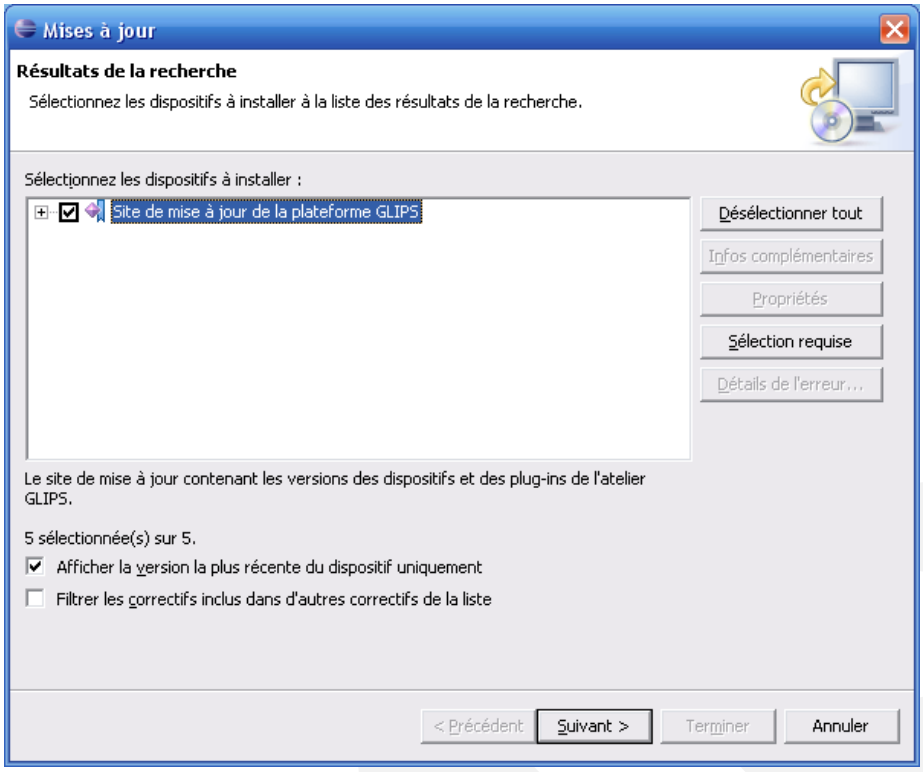


Figure 3.10: Sélection des dispositifs

- Acceptez les termes de la licence, puis cliquez sur "Suivant".

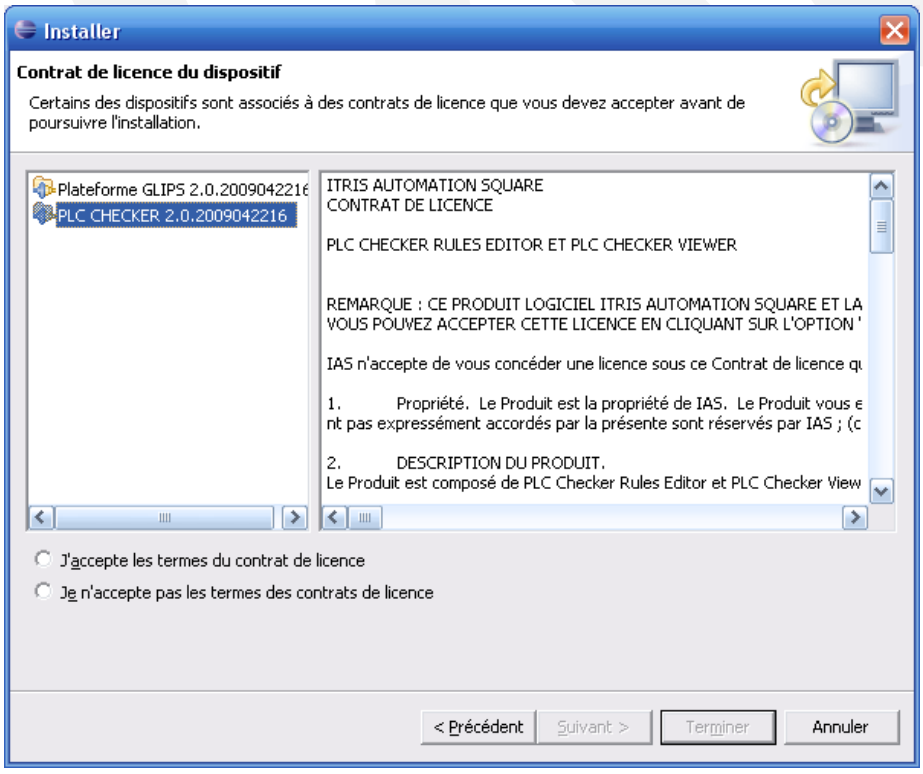


Figure 3.11: Licence

- Le récapitulatif de l'installation des mises à jour vous est présenté. Cliquez sur "Terminer".

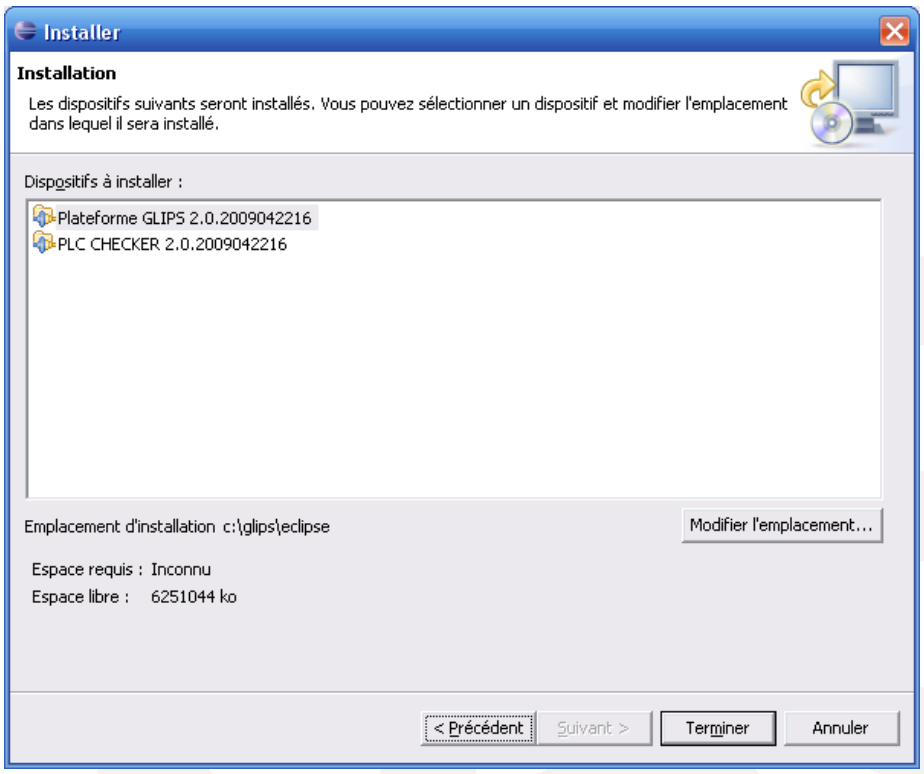


Figure 3.12: Récapitulatif de l'installation

- Après téléchargement, les dispositifs n'étant pas signés, une vérification de l'émetteur vous est demandée. Cliquez sur "Installer tout".

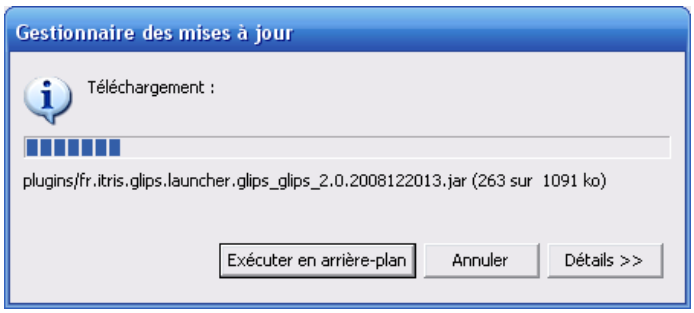


Figure 3.13: Téléchargement

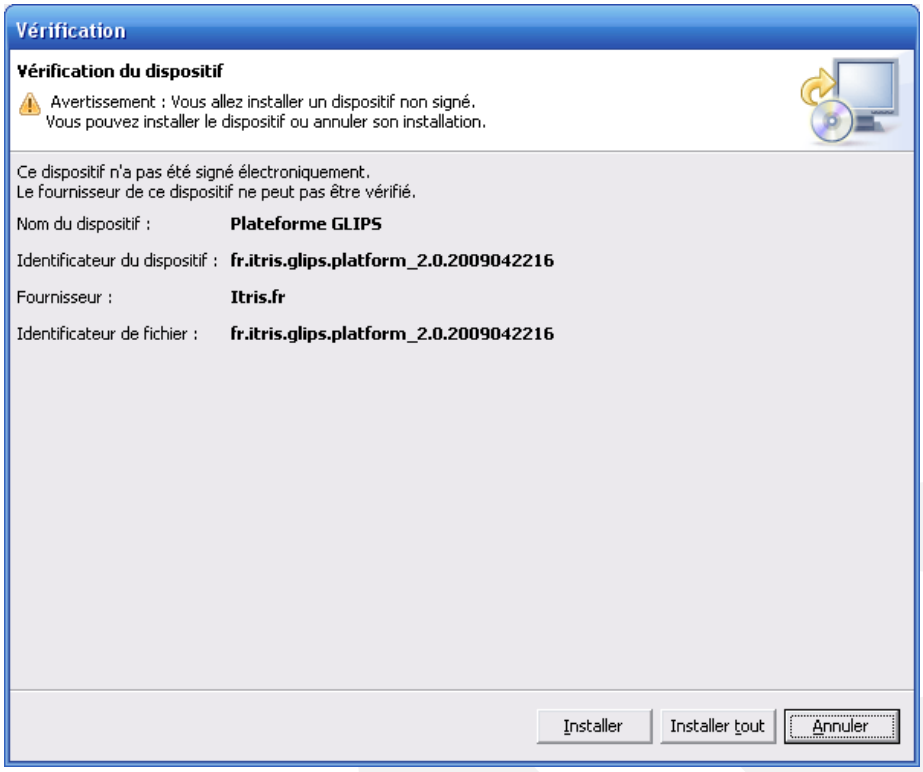


Figure 3.14: Vérification des dispositifs

- Après installation, relancez le plan de travail comme proposé.



Figure 3.15: Installation des dispositifs

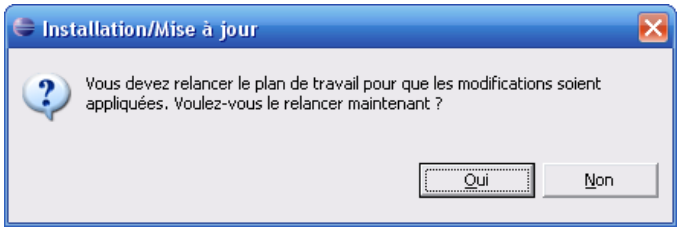


Figure 3.16: Demande de redémarrage

3.3 Lancement de la plate-forme

Pour lancer la plate-forme, cliquez sur l'icône présente sur le bureau ou choisir l'une des entrées **ITRIS Automation Square/PLC Converter** ou **ITRIS Automation Square/PLC Checker** dans le menu **Démarrer**.

A la fin d'une installation de la plate-forme, elle se lance automatiquement.

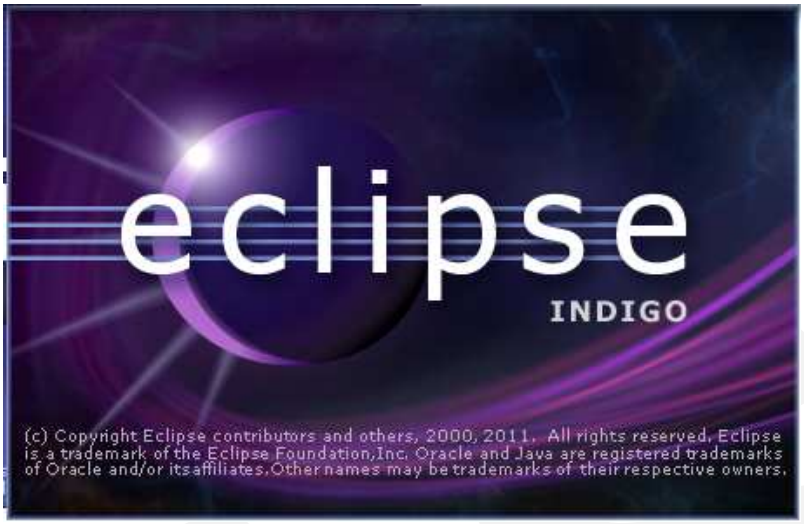


Figure 3.17: Logo d'Eclipse

3.3.1 Espace de travail

Lors du lancement de la plateforme, la boîte de dialogue suivante apparaît:

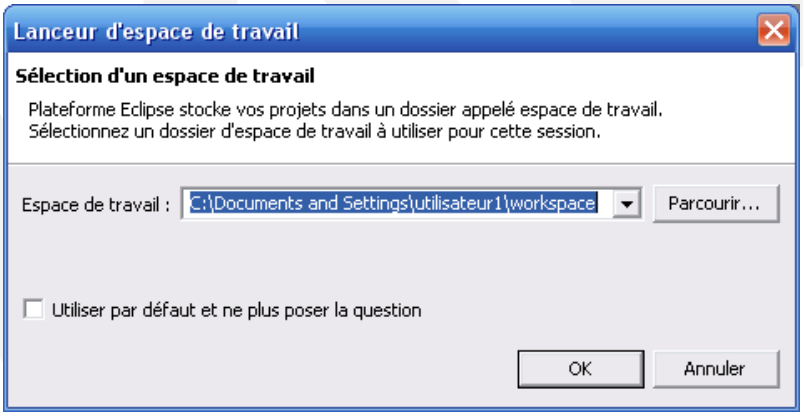


Figure 3.18: Choix de l'espace de travail

L'espace de travail est un répertoire sur votre disque dont l'usage est réservé à Eclipse. Dans le répertoire spécifié, Eclipse va créer un fichier **.metadata** qui contiendra les données dont il a besoin pour fonctionner. Il est important:

1. De séparer l'espace de travail d'Eclipse, de vos répertoires de travail usuels
2. De ne pas utiliser l'explorateur pour gérer les fichiers de l'espace de travail (utiliser pour cela les fonctions d'import/export ou le drag'n drop de fichiers).
3. D'avoir un seul espace de travail

Le fichier **.metadata** contient le descriptif des projets pour Eclipse.

Une fois le répertoire qui vous convient choisi, vous pouvez cocher la case de la boîte de dialogue vous indiquant que la question ne sera plus posée lors des prochaines ouvertures d'Eclipse.

3.3.2 Identification

Lorsque cela vous est demandé, identifiez-vous en saisissant, dans la fenêtre prévue à cet effet, votre login et votre mot de passe définis lors de votre inscription sur notre site à la page d'enregistrement .

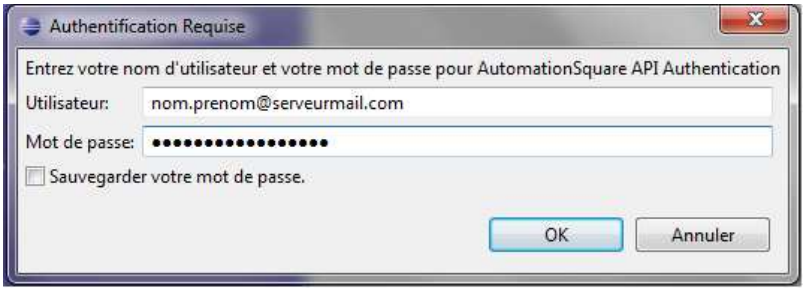


Figure 3.19: Saisie des identifiants

3.3.3 Plan de travail

Pour le cas où il s'agit du premier démarrage, la fenêtre suivante vous accueille :

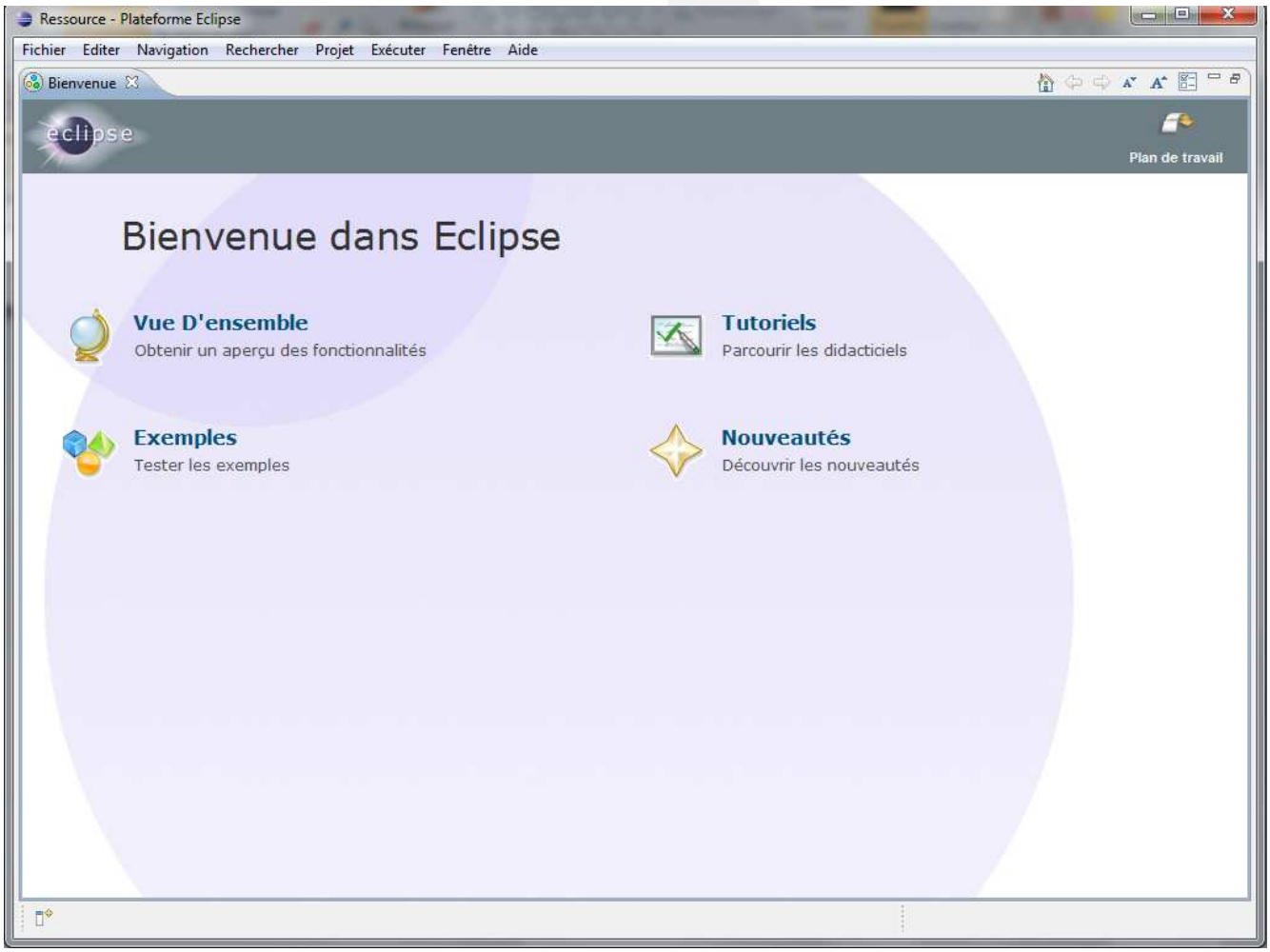


Figure 3.20: Page d'accueil Eclipse

Vous avez la possibilité de suivre les tutoriaux d'utilisation d'Eclipse.

Dans le cas contraire, vous pouvez quitter cet écran en cliquant sur **Retourner au plan de travail** (flèche en haut à gauche) ou en fermant la fenêtre intérieure.

Le plan de travail d'Eclipse comporte plusieurs fenêtres.

La disposition par défaut est la suivante:

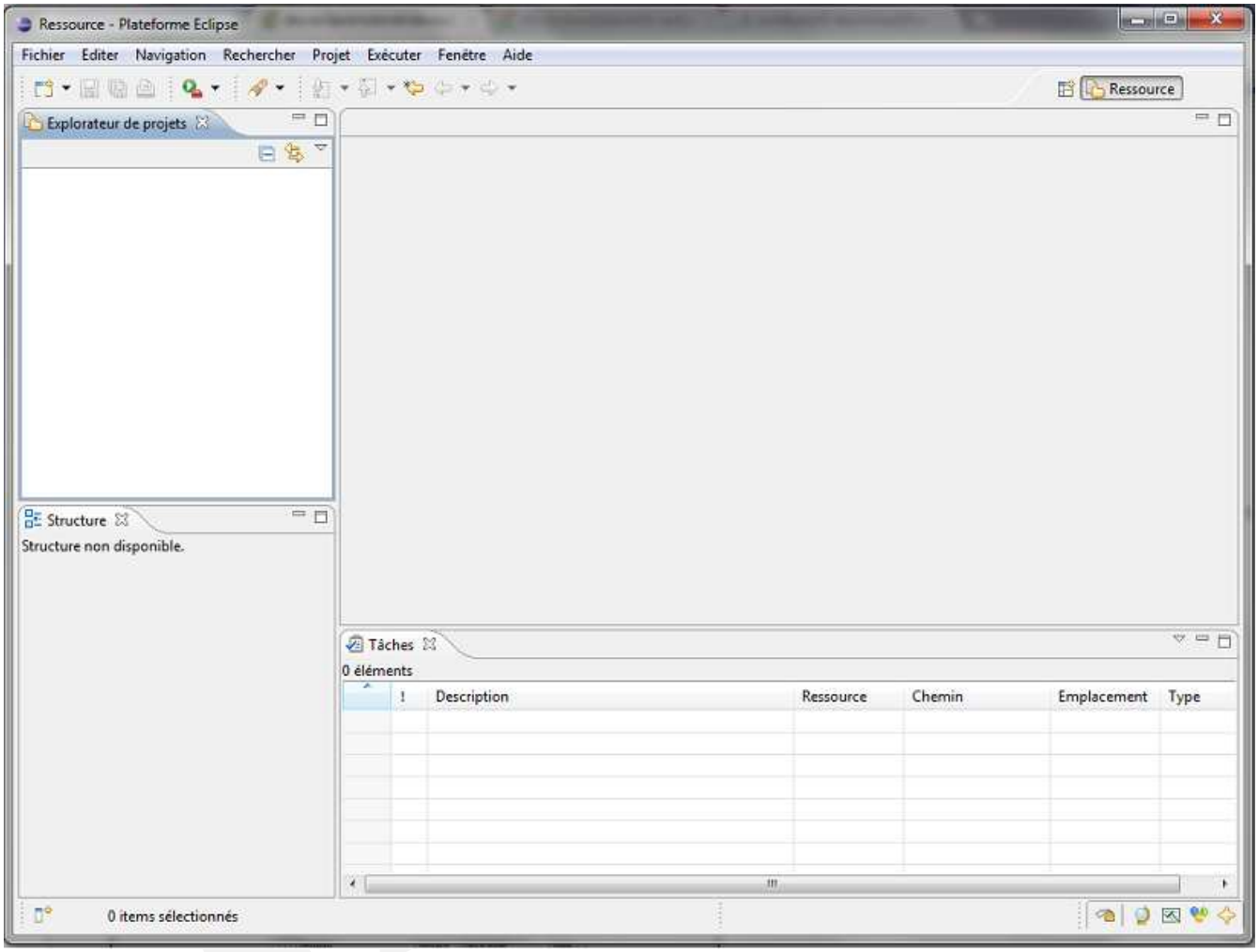


Figure 3.21: Plan de travail Eclipse

Sur la partie gauche se trouve le navigateur. Il propose une vue de navigation dans l'espace de travail.

A droite, la grande partie grise contient les éditeurs susceptibles d'être ouverts lorsque l'on sélectionne des fichiers de l'espace de travail.

Au bas à gauche de l'écran, la fenêtre "Structure" offre une vue structurée de l'éditeur courant. Dans le cas présent, elle est vide.

Enfin, au bas de l'écran se trouve une fenêtre de tâches à accomplir. Eclipse permet de développer efficacement et offre la possibilité d'ajouter des tâches à faire, afin de ne rien oublier.

Remarque: La recherche et l'installation des mises à jour de la plateforme se fait automatiquement à chaque démarrage de cette dernière. Si l'interface semble ne pas répondre, vérifier qu'elle ne fait pas une MAJ.

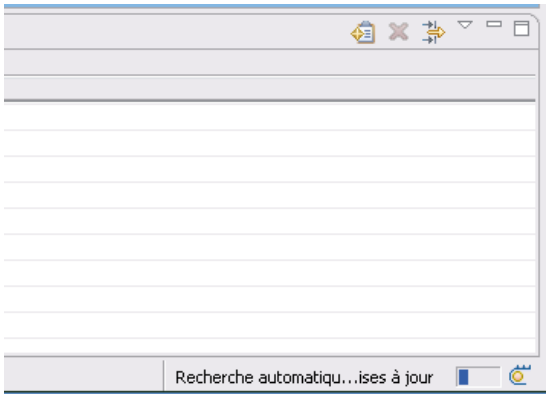


Figure 3.22: Recherche automatique des mises à jour

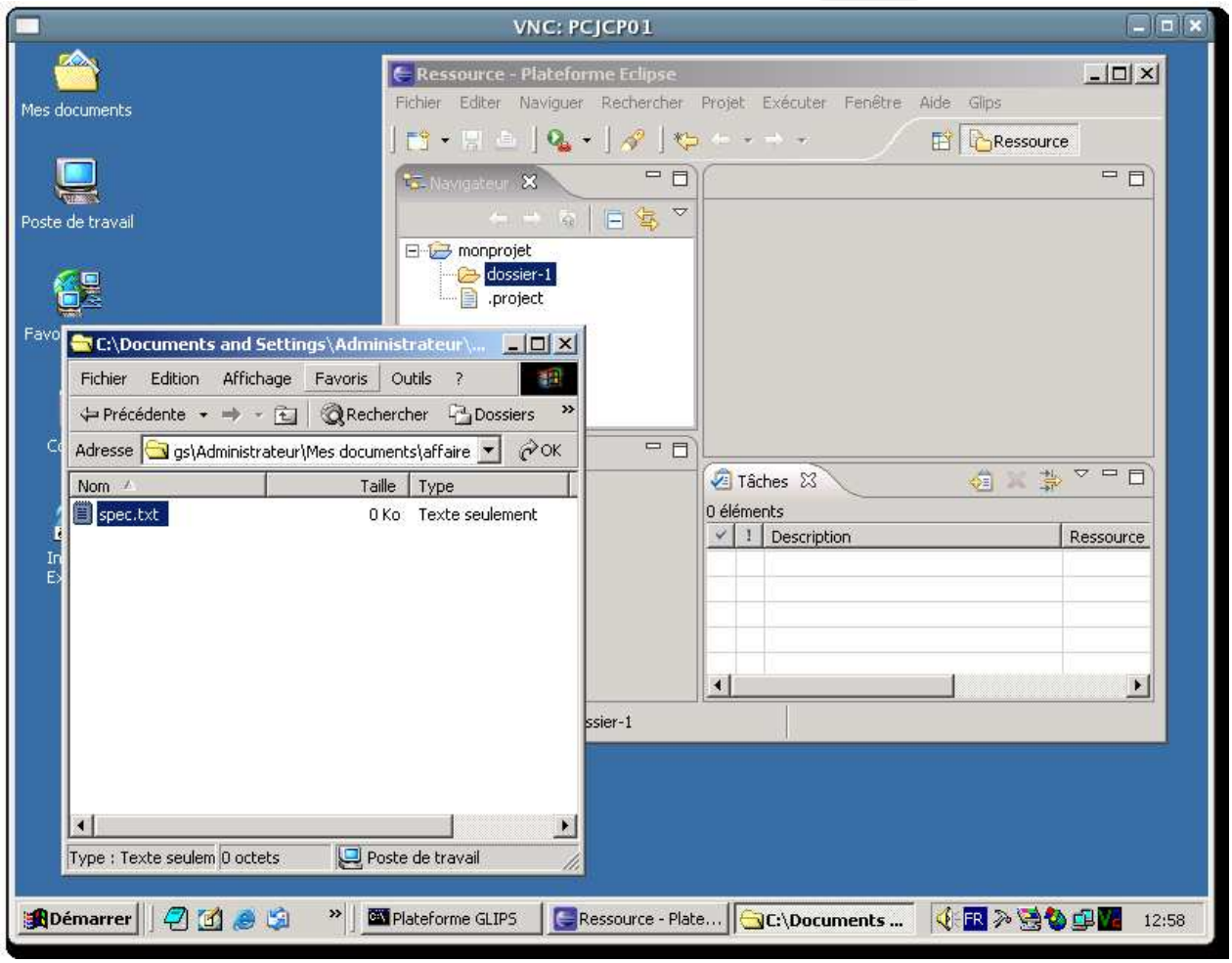
IAS

3.4 Importation/Exportation de fichiers dans le projet

L'intégration de fichiers à l'intérieur de l'espace de travail doit se faire par Eclipse et pas en utilisant le navigateur de Microsoft. Il existe plusieurs manières d'importer et d'exporter des données dans l'espace de travail. Nous allons voir ici comment faire un drag'n drop ou utiliser les fonctions d'importation.

3.4.1 Drag'N Drop

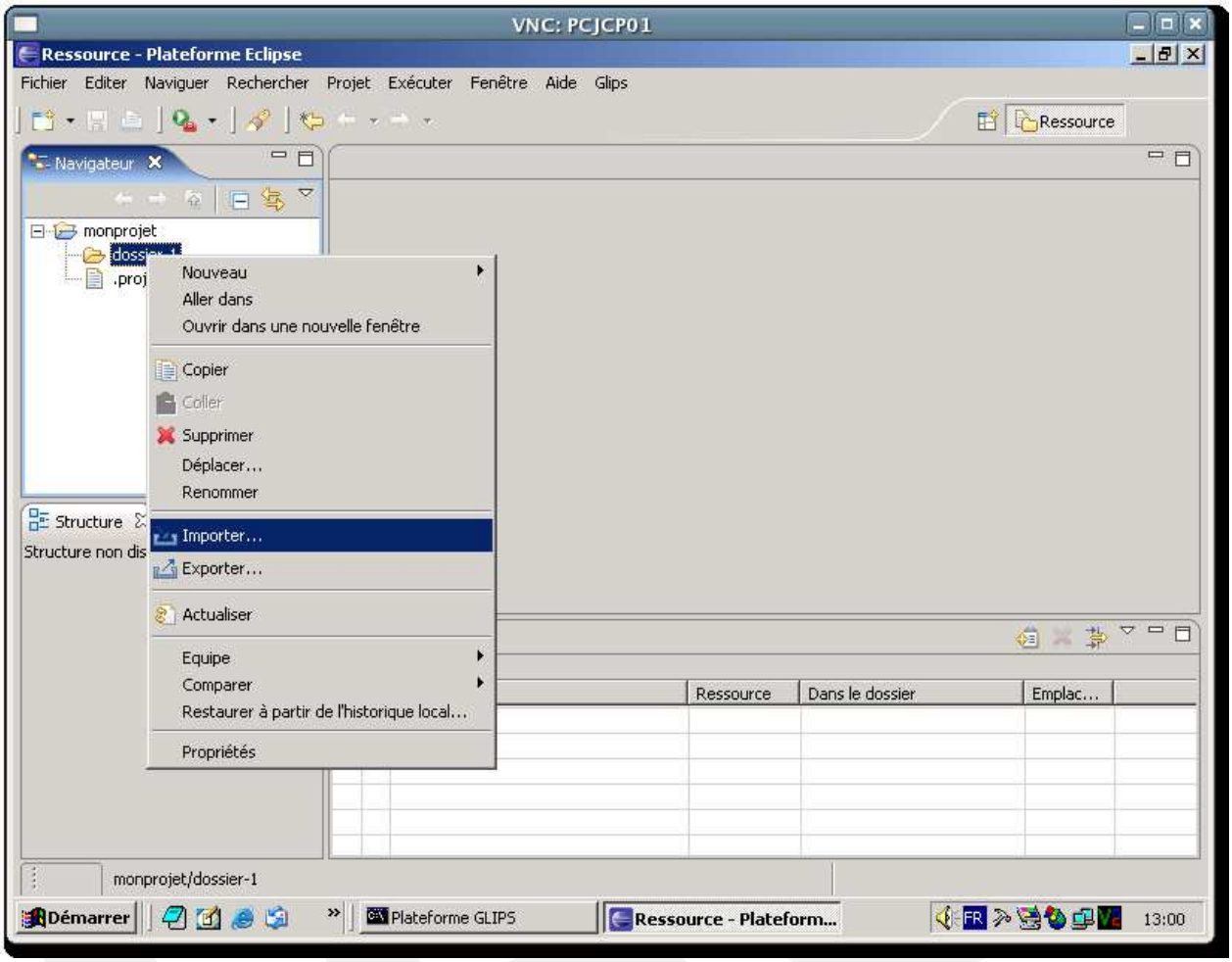
Les photos d'écrans suivantes montrent les étapes à suivre pour importer des fichiers dans l'espace de travail en utilisant le drag'n drop (glisser-déposer). Il suffit d'avoir un explorateur de fichier ouvert, de glisser, déposer les fichiers dont on a besoin dans l'espace de travail. Il est à noter que ceux-ci ne seront pas détruits de leur emplacement source par l'opération.



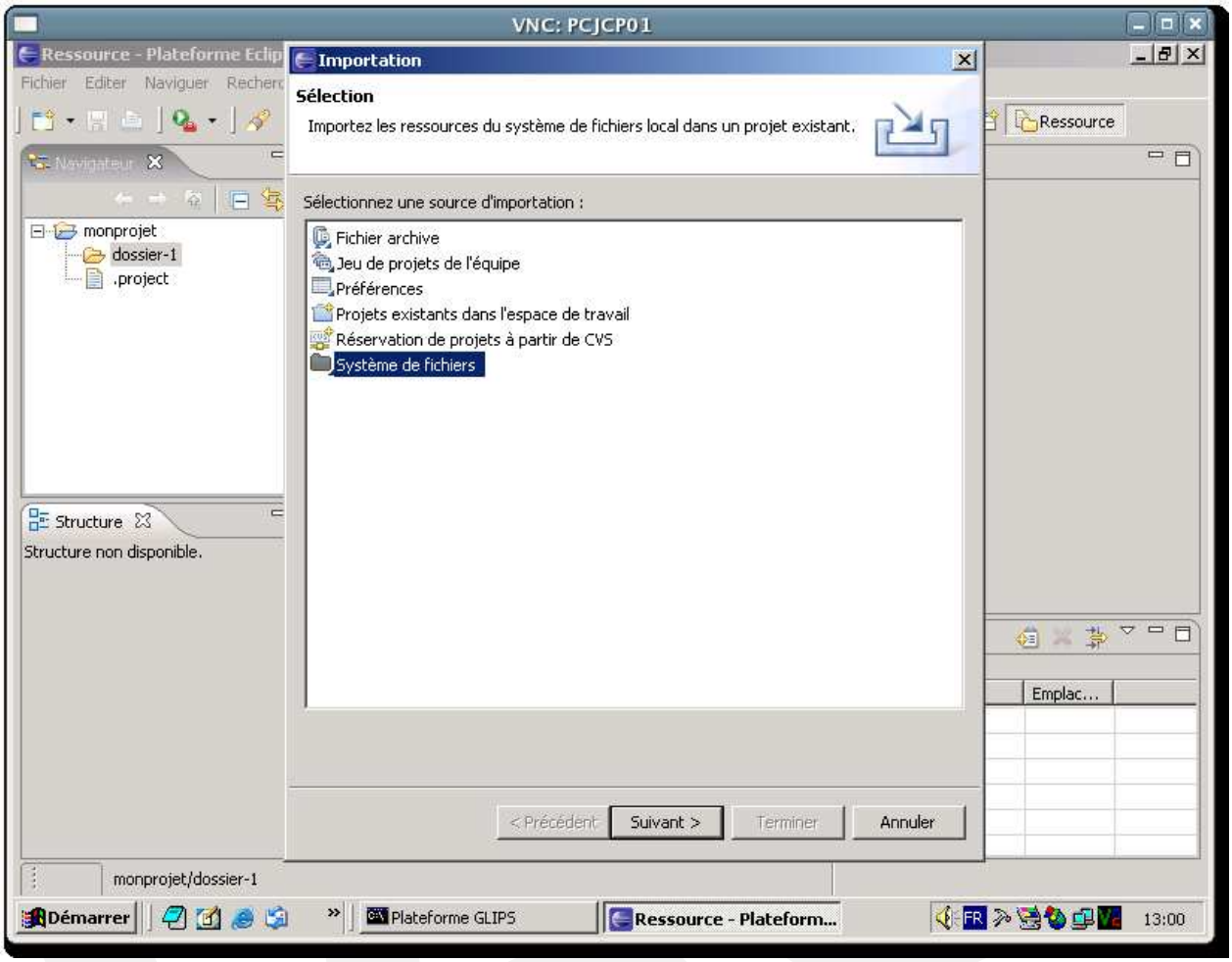
3.4.2 Importation

L'importation en utilisant les dialogues est plus complexe à utiliser mais s'adapte à des cas où seuls certains fichiers d'un répertoire doivent être importés. Leur sélection en est facilitée.

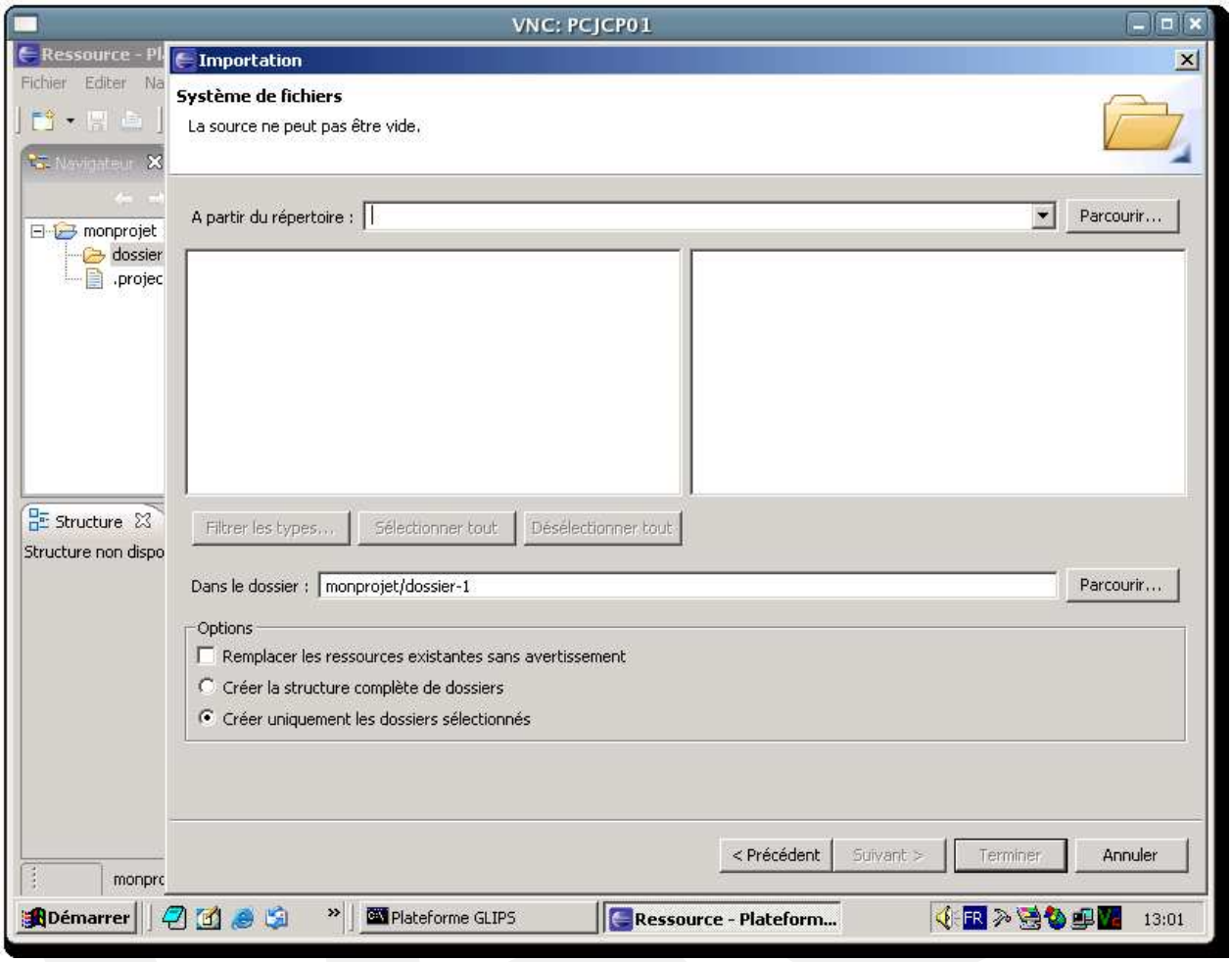
Dans notre exemple, on souhaite importer des fichiers dans le dossier1. Il faut donc sélectionner le dossier_1, ouvrir le menu contextuel en cliquant sur le bouton de droite de la souris et sélectionner "Importer"...



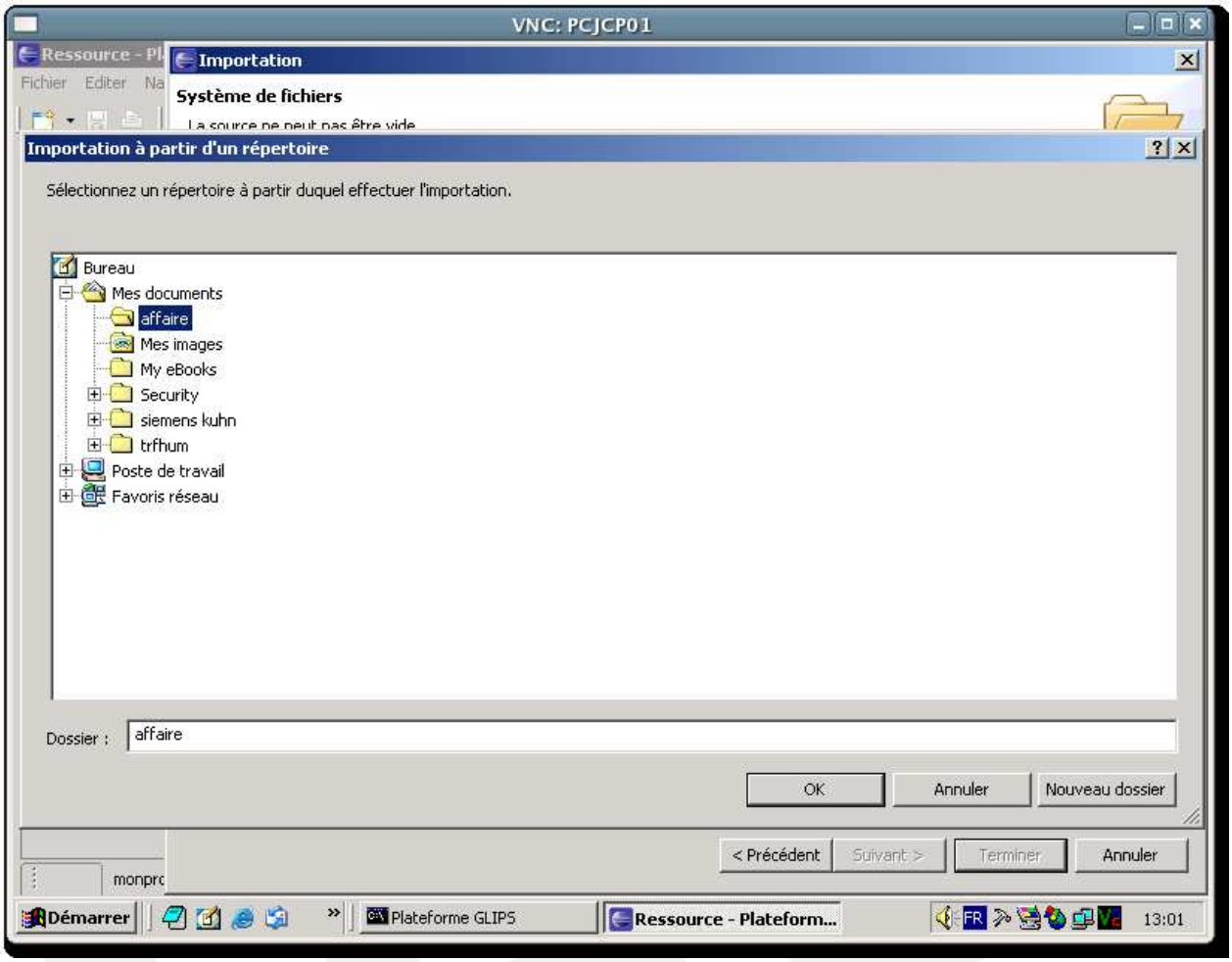
De toutes les possibilités d'import offertes, celle qui nous intéresse le plus est l'importation depuis le système de fichiers.



Ensuite, il convient de sélectionner le répertoire sur le disque qui contient les fichiers à importer.

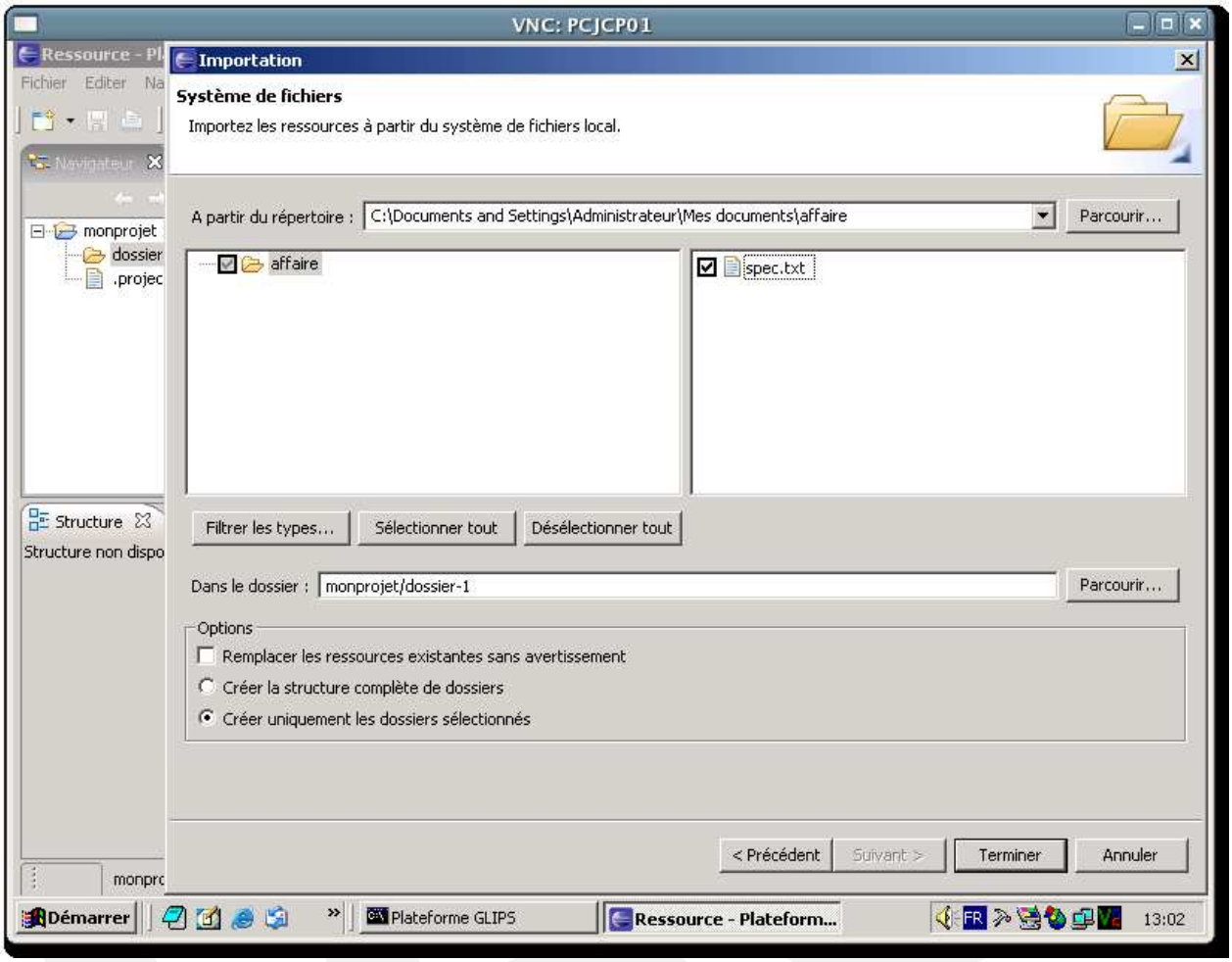


Ceci se fait en cliquant sur "Parcourir"…. Une boite de sélection du répertoire s'ouvre.



Une fois le choix du dossier source effectué, on valide et apparait la liste des fichiers importables. Il convient de sélectionner les désirés. Dans notre cas, on va importer le fichier spec[2].txt.

[2] specification



Enfin on valide et le fichier souhaité est importé dans l'espace de travail.

3.5 Comparaison de fichiers

Il est très facile de comparer des fichiers dans l'environnement d'Eclipse. Il suffit de sélectionner les deux fichiers que l'on souhaite comparer. Ensuite, le menu contextuel nous propose de comparer les fichiers réciproquement.

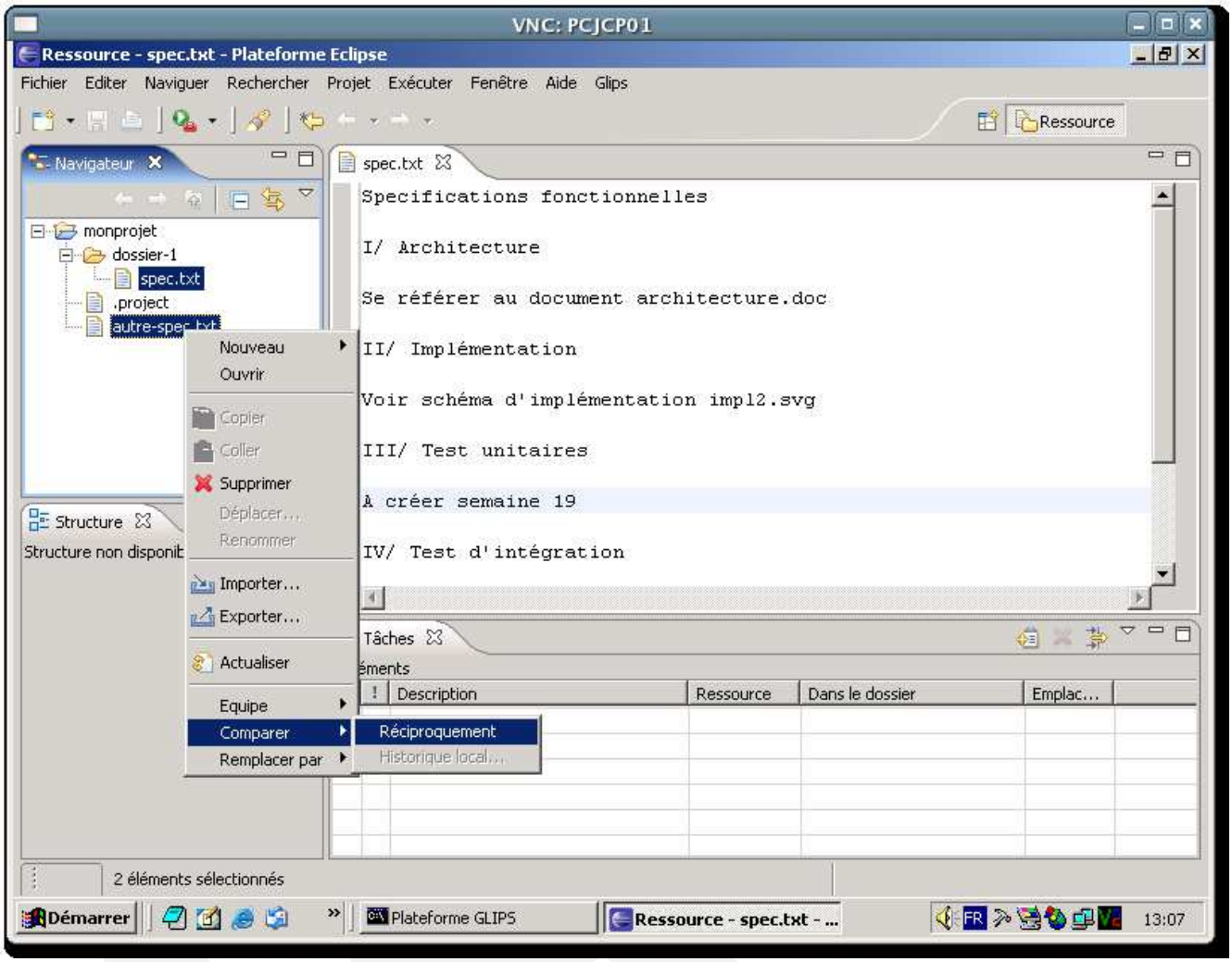


Figure 3.23: Demande de comparaison de deux fichiers

La sélection de l'entrée de menu permet l'ouverture de l'éditeur de différences:

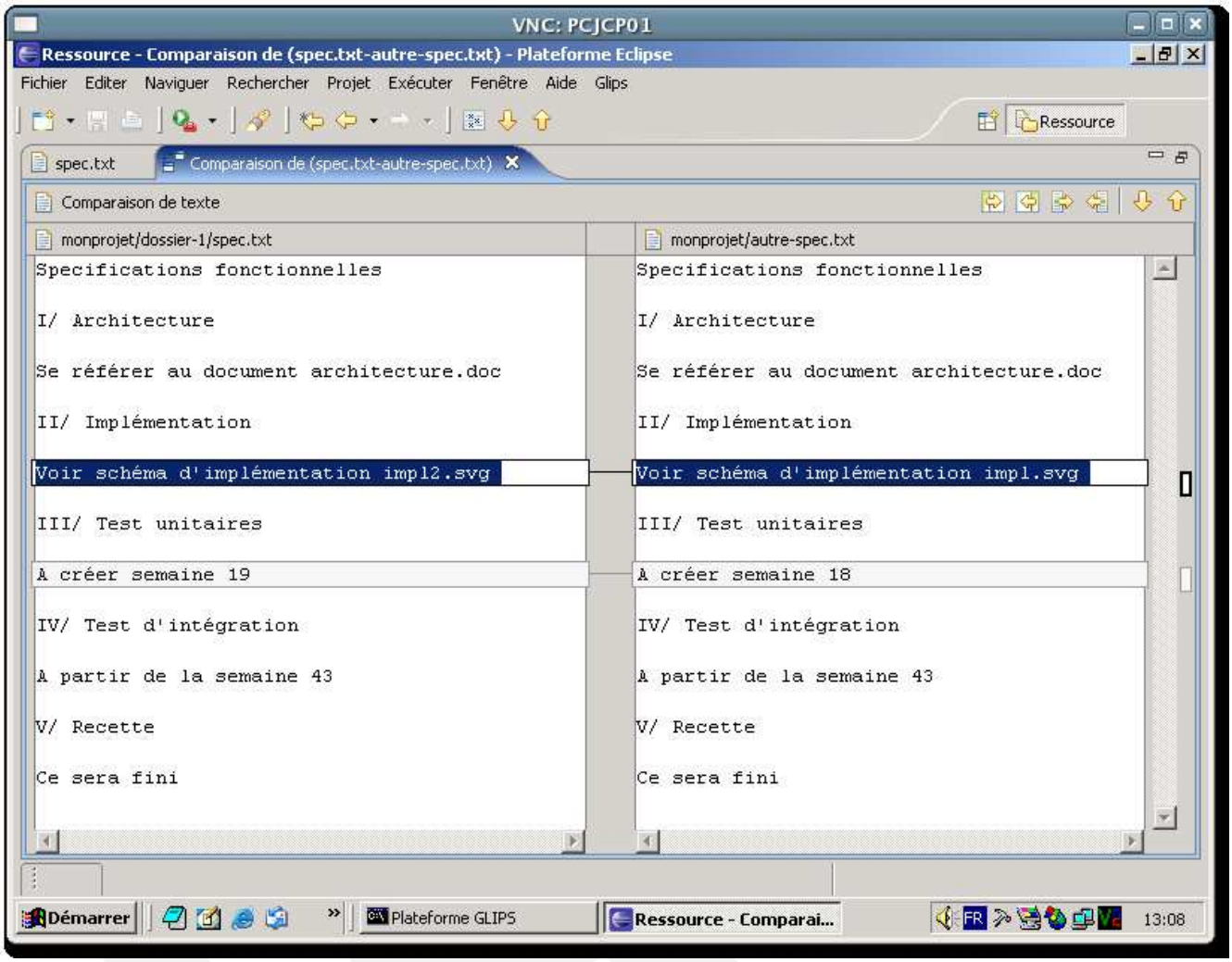


Figure 3.24: Editeur de différences

Il est possible de naviguer dans les différences en cliquant sur les petits rectangles blancs à la droite de la fenêtre.

Pour sortir du mode de comparaison, il suffit de fermer la fenêtre principale.

3.6 Historique local

Pour tous les fichiers texte qui se trouvent à l'intérieur d'un projet, Eclipse propose la fonction d'historique local. Celui-ci permet de garder des traces sur les modifications successives des fichiers texte. L'utilisateur peut à tout moment étudier les différences entre la version actuelle et les versions précédentes et revenir à une version précédente très rapidement.

3.6.1 Comparer avec une version précédente

Il suffit de sélectionner un seul fichier dans l'environnement Eclipse. Puis avec le menu contextuel, sélectionner "Comparer réciproquement".

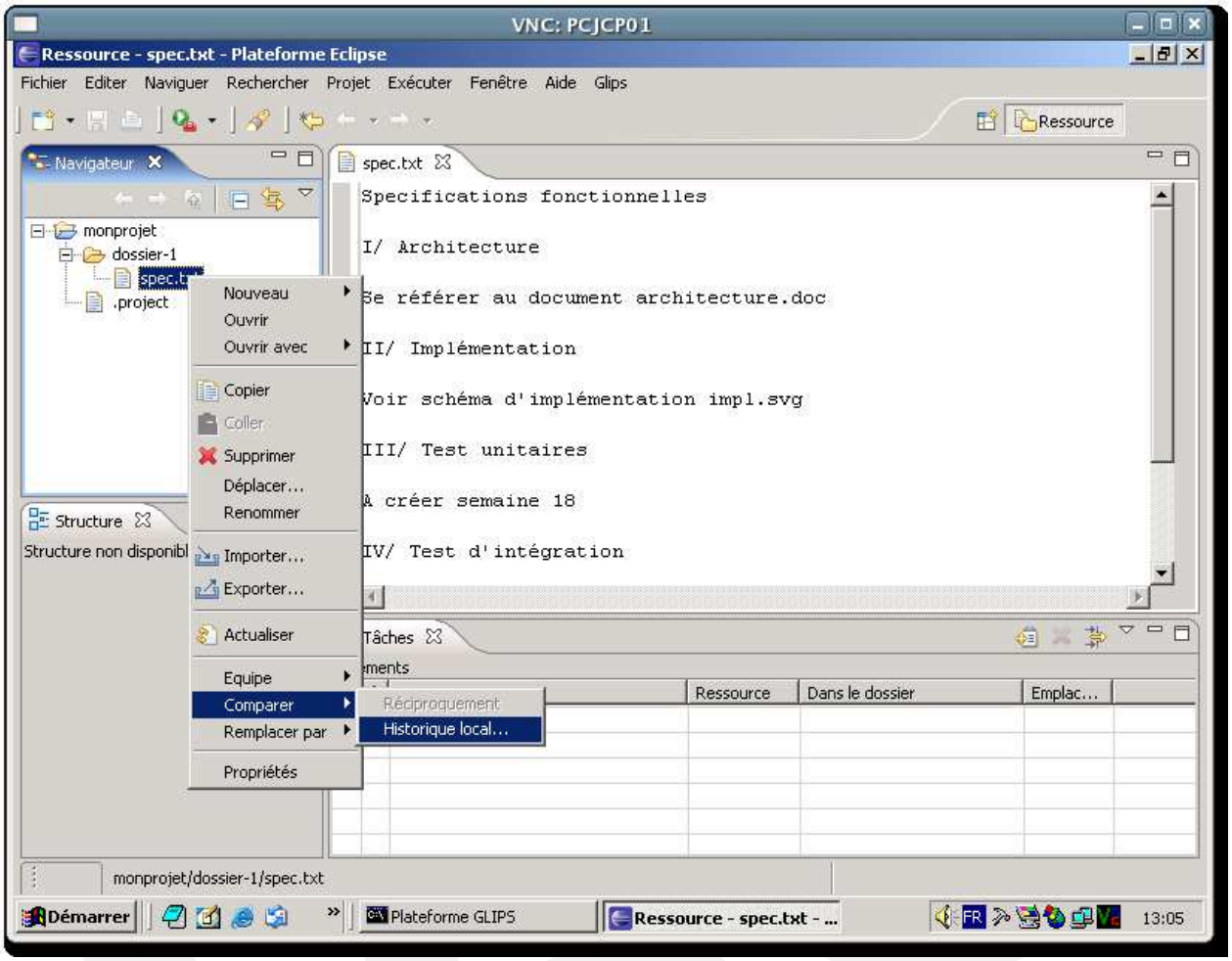


Figure 3.25: Demander un historique local

La fenêtre qui s'ouvre ressemble fortement aux fenêtres de comparaison entre fichiers au dessus de laquelle se trouve l'historique local du fichier. En cliquant sur les différentes versions, on obtient les différences entre la version actuelle et la version sélectionnée.

3.6.2 Remplacer par une version précédente

Dans le menu contextuel, en sélectionnant Remplacer par historique local, il est possible tout en visualisant les différences, de remplacer la version actuelle par une version plus ancienne.

Chapter 4

Création et utilisation d'un projet PLC Checker

La procédure synthétique pour la création d'un projet, l'utilisation de PLC Checker et la visualisation des résultats sont les suivantes :

- **Préambule** :
  1. Création d'un compte utilisateur sur le site d'Itris Automation Square
  2. Achat de droits auprès du commercial@automationsquare.com d'Itris Automation Square
  3. Obtention du fichier de règle ou Création d'un fichier de règles, suivi de son édition.
  4. Export, à partir de l'atelier automate, des différents fichiers nécessaires.
- **Création** d'un projet et d'un programme :
  1. Création d'un projet et d'un programme de type PLC Checker à partir de la plateforme Eclipse
     (a) Nommage du projet à partir de la plateforme Eclipse
     (b) Association d'un droit au projet à partir de la plateforme Eclipse
  2. Création d'un projet et d'un programme de type PLC Checker sur le site d'Itris Automation Square
     (a) Nommage du projet à partir du site web (étape 1)
     (b) Association d'un droit au projet à partir du site web (étape 4)
- **Exécution** du PLC Checker :
  - Méthode 1 : à partir de la plateforme Eclipse :
    1. Création d'un projet PLC Checker dans la plateforme Eclipse, reprenant le contenu et les caractéristiques du projet précédemment créé sur le site web.
    2. Import des fichiers de l'application automate dans l'espace de travail.
    3. Import du fichier de règle dans l'espace de travail.
    4. Exécution de PLC Checker à partir de la plateforme Eclipse.
  - Méthode 2 : à partir du site web :
    1. Chargement du/des fichier(s) des applications automates sur le serveur à partir du site web (étape 2).
    2. Chargement du fichier de règles sur le serveur à partir du site web (étape 3).
    3. Exécution de PLC Checker sur le serveur à partir du site web (étape 5).
- **Visualisation** des résultats sur la plateforme Eclipse.
  1. Si à l'exécution vous avez choisi la méthode 2, il faut créer un projet PLC Checker dans la plateforme Eclipse, reprenant le contenu et les caractéristiques du projet précédemment créé sur le site web (voir la section : Création d'un projet existant sur la plate-forme Eclipse)
  2. Visualisation du fichier résultat (*.gqre) sous forme d'arbre, sur la plateforme Eclipse.

4.1 Création d'un projet PLC Checker

1. Création sur IAS avec assistant (projet PLC Checker par défaut)
2. Création sur IAS avec assistant (projet PLC Checker à définir)
3. Création sur IAS d'un programme existant
4. Création d'un nouveau projet sur la plate-forme Eclipse
5. Création d'un projet existant sur la plate-forme Eclipse

4.1.1 Création avec assistant d'une application PLC Checker par défaut sur le site d'Itris Automation Square (Projet Plc Checker par défaut)

Pour créer une application PLC Checker avec l'assistant, veuillez cliquer sur le bouton "+ Nouveau PLC Checker" ou cliquer sur "Outils" puis "PLC Checker" dans le menu à gauche de la page. L'application est créée par défaut dans le projet par défaut, vous pouvez la déplacer dans un autre projet après avoir fini la création.

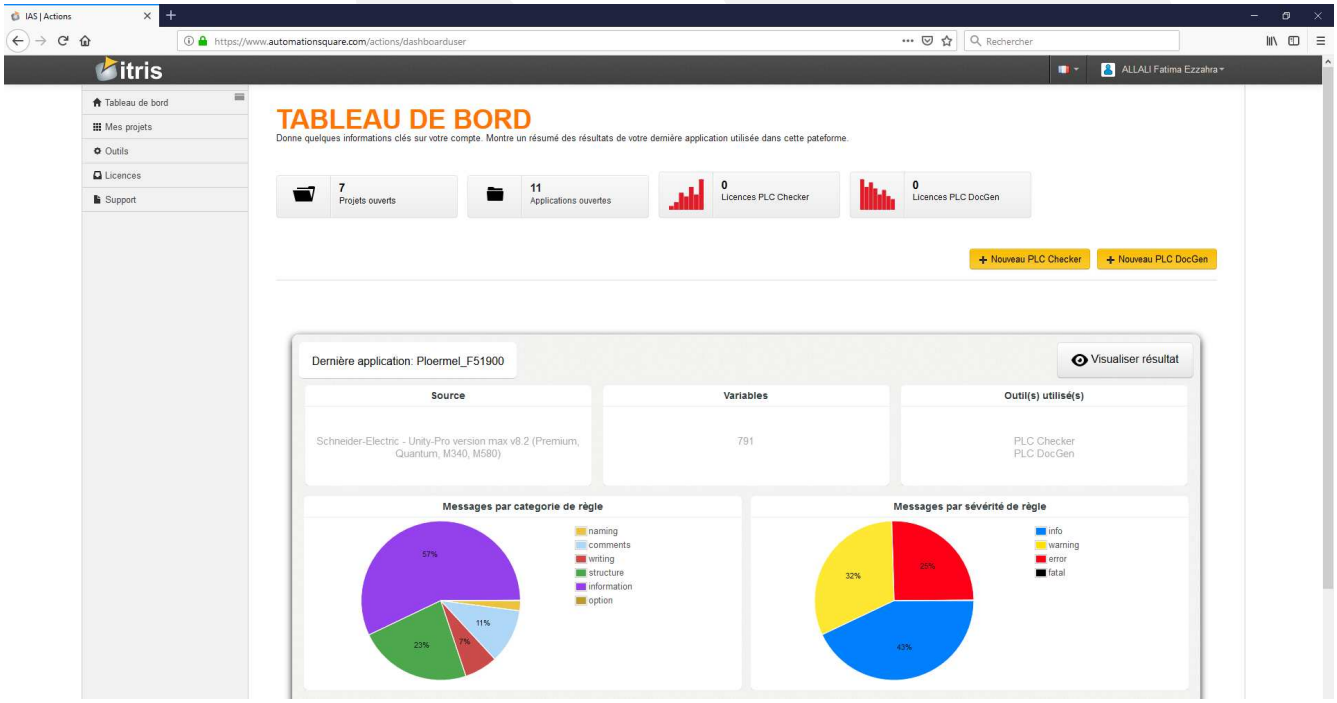


Figure 4.1: Tableau de bord

- Nommage de l'application à partir du site web (étape 1) :
  1. Donner un nom de l'application et un nom du client.
  2. Des informations complémentaires peuvent être saisies pour l'application.
  3. Finir la saisie en cliquant sur le bouton "Suivant".

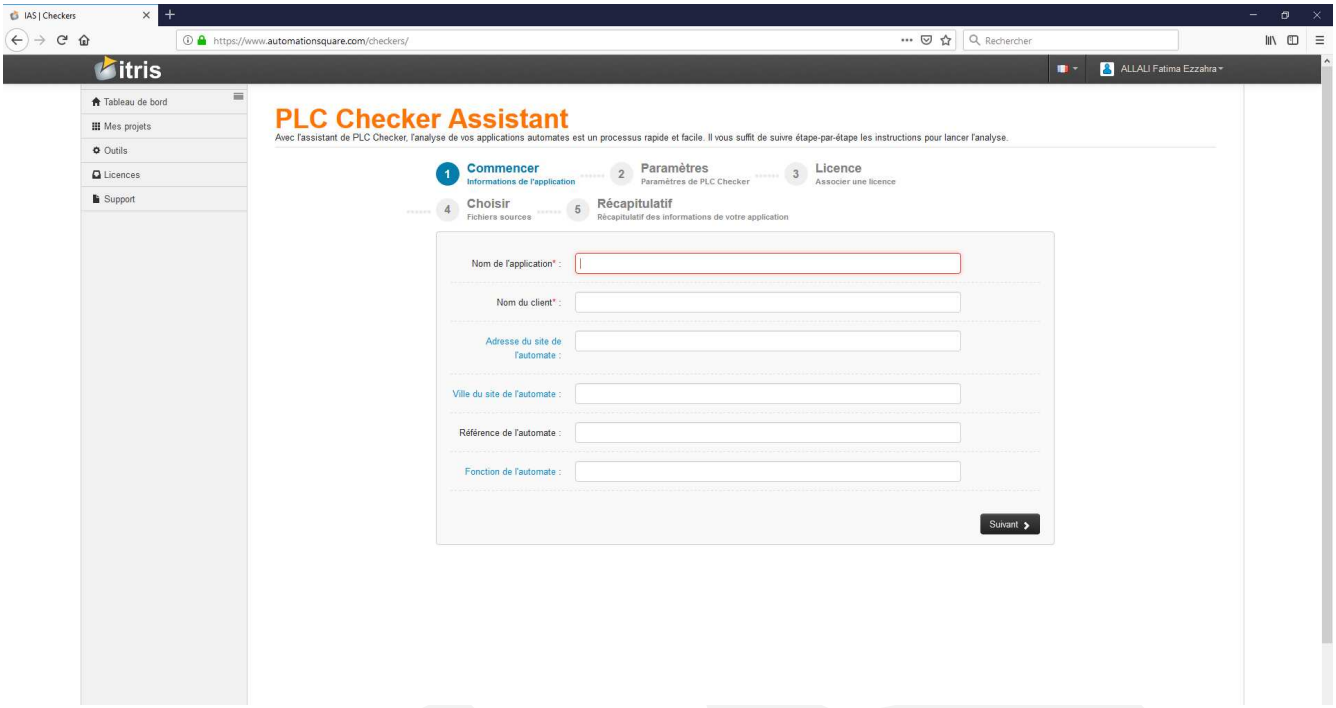


Figure 4.2: Informations du projet

- Choix du type d'automate et du fichier de règles (étape 2).

Veuillez sélectionner le type d'automate parmi la liste.

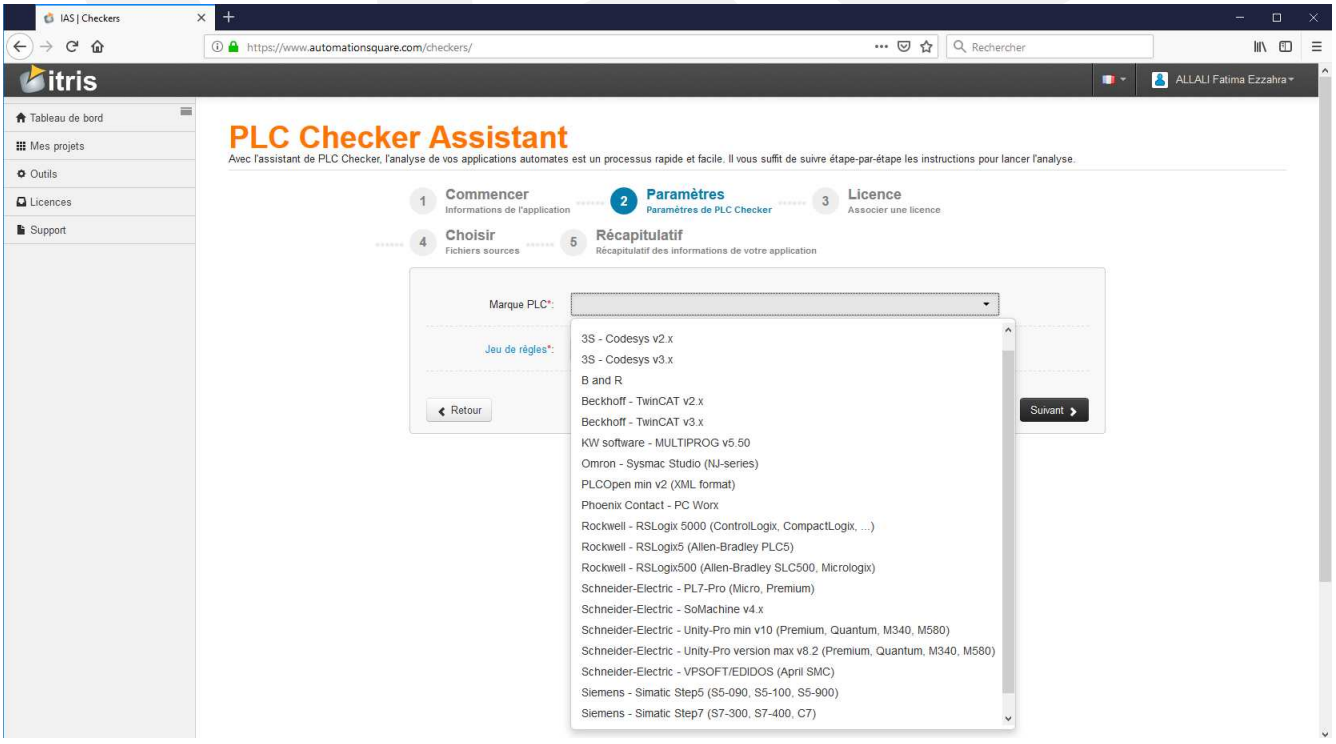


Figure 4.3: Choix de l'automate

Veuillez sélectionner le fichier de règles à utiliser ou choisir "Téléchargement dans l'étape 4" pour ajouter un nouveau fichier de règles dans l'étape 4.

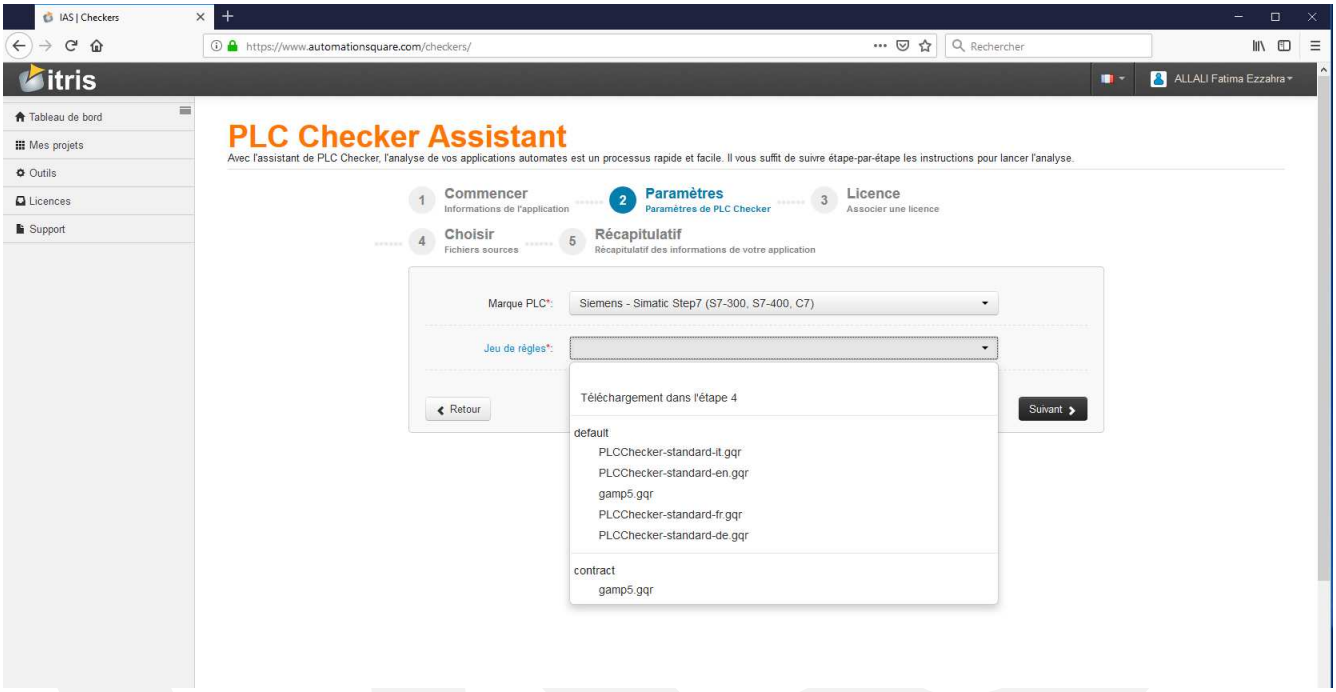


Figure 4.4: Sélection de fichier de règles

- Association d'un droit au projet à partir du site web (étape 3).

Si vous n'avez pas encore de droit disponible pour cette action, vous pouvez quand même continuer la création de l'application en appuyant sur "Suivant".

Si vous avez des droits disponibles pour cette action, veuillez sélectionner le droit à associer à cette application dans le menu déroulant et appuyer sur "Suivant".

Si vous avez une licence disponible libre, veuillez saisir son identifiant dans la case "#Id licence" et cliquer sur le bouton "Suivant".

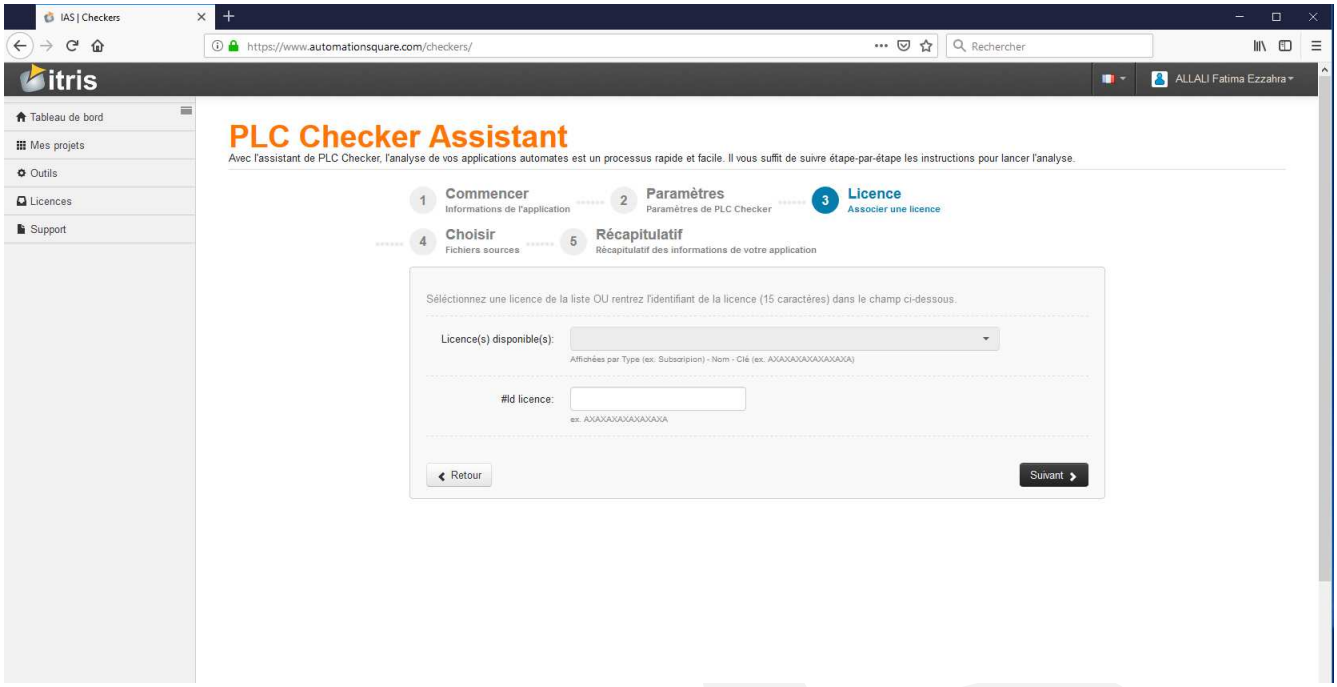


Figure 4.5: Association de la licence

- Chargement de fichiers de règles et fichiers sources (étape 4).

Veuillez cliquer sur le bouton "SÉLECTIONNEZ LE(S) FICHIER(S)..." pour importer le fichier de règles ainsi que les fichiers sources de l'application puis cliquez sur "Suivant" pour valider. En fonction du type d'automate choisi, vous avez une liste des fichiers autorisés. Vous pouvez continuer sans ajouter de fichiers en cliquant sur "Suivant".

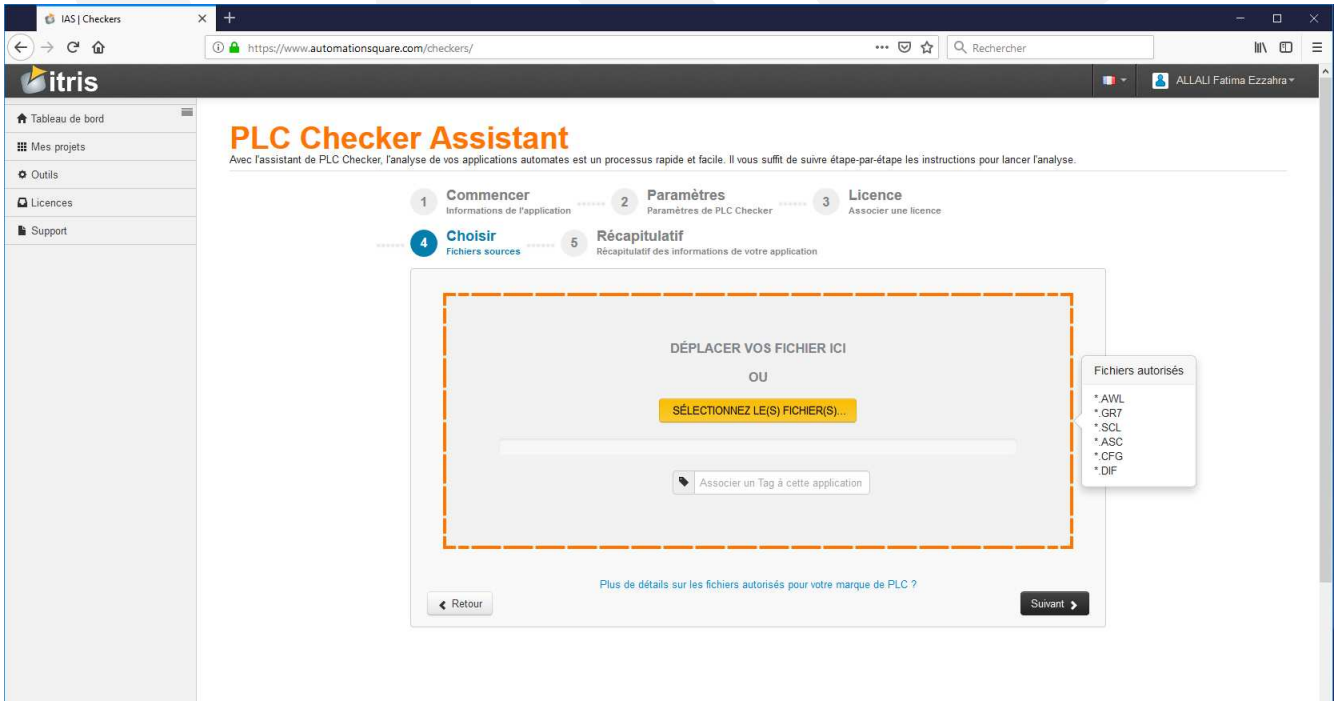


Figure 4.6: Sélection de fichiers

- Récapitulatif des informations de l'application (étape 5).

Dans cette étape, vous trouvez les informations que vous avez saisi sur l'application. Pour les modifier, vous pouvez cliquer sur le bouton "Retour" sinon vous cliquez sur "Créer" pour créer l'application ou "Exécuter" pour créer et exécuter Plc Checker.

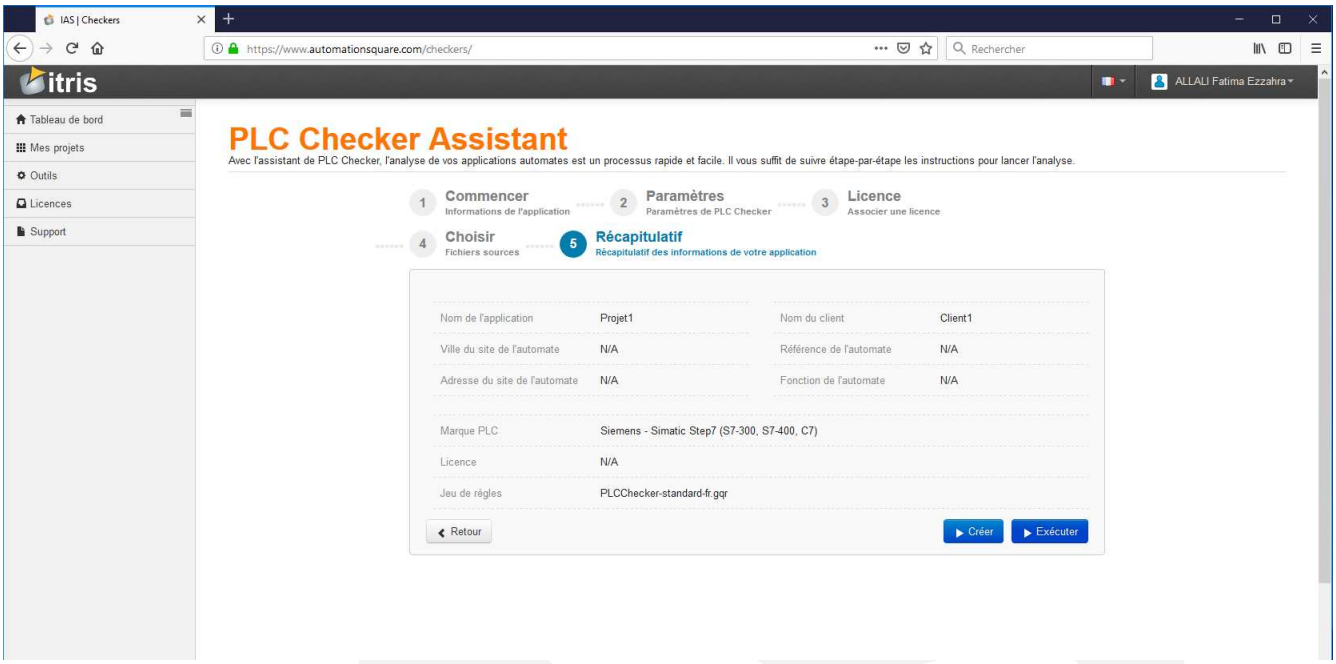


Figure 4.7: Récapitulatif

- Noter le code à 6 caractères de votre programme, il servira pour la récupération des informations du projet par un autre utilisateur sur le site web ou sur la plateforme Eclipse du PLC Checker.

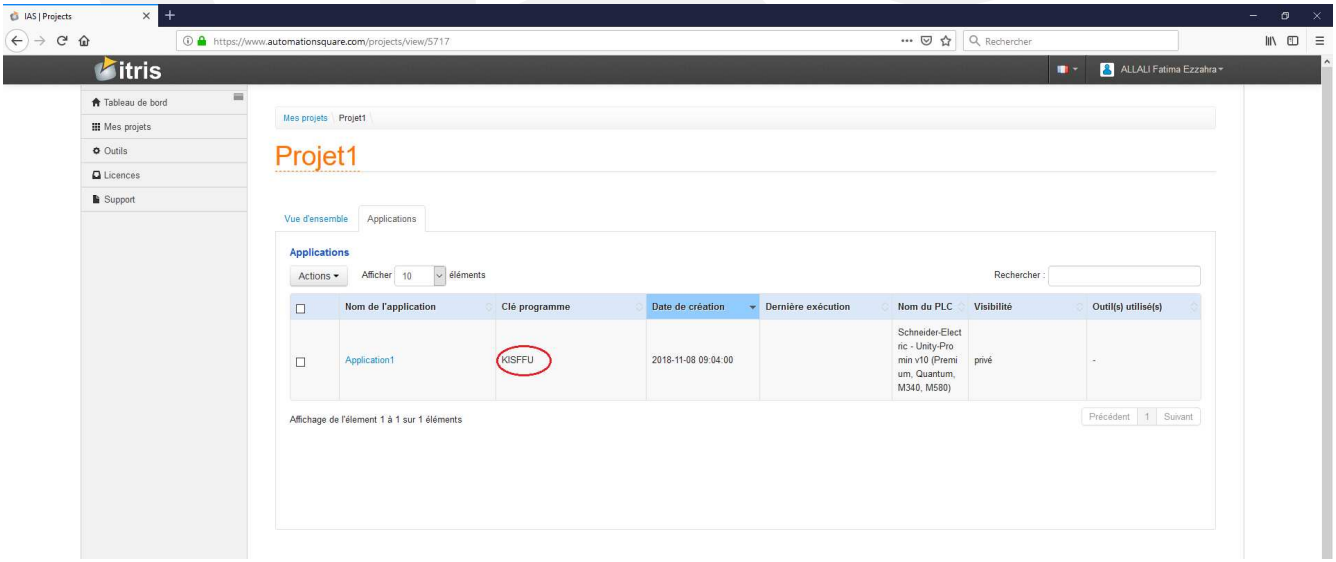


Figure 4.8: Code à 6 caractères

4.1.2 Création avec assistant d'une application de type PLC Checker sur le site d'Itris Automation Square (Projet Plc Checker à définir)

- Accéder à la page des projets (onglet "Mes projets" dans le menu à gauche).

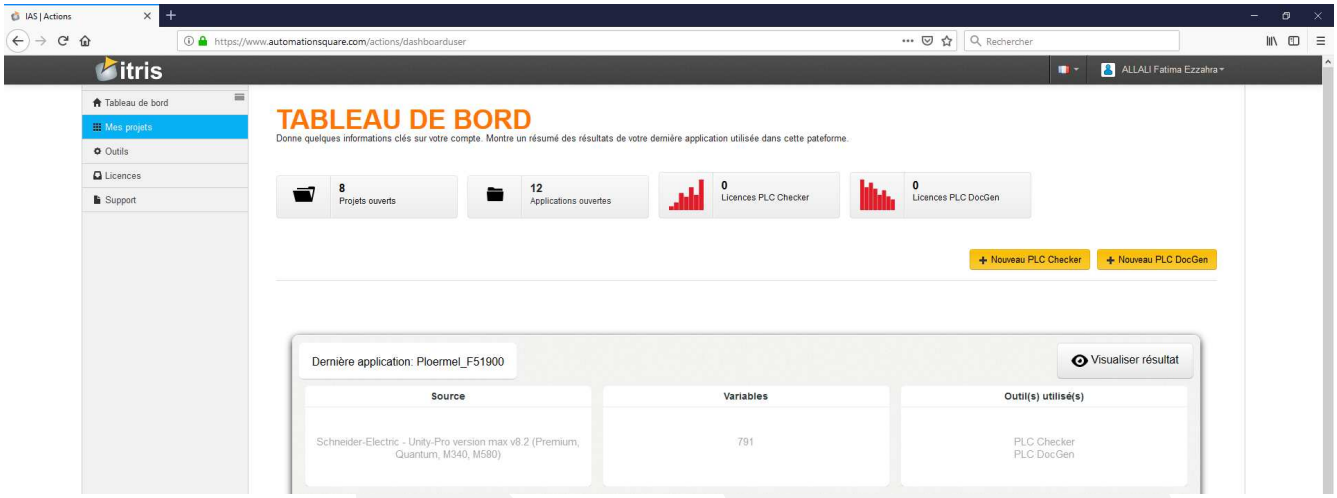


Figure 4.9: Accéder à la page des projets

- Créer un nouveau projet pour contenir les programmes: veuillez appuyer sur Actions puis Nouveau projet

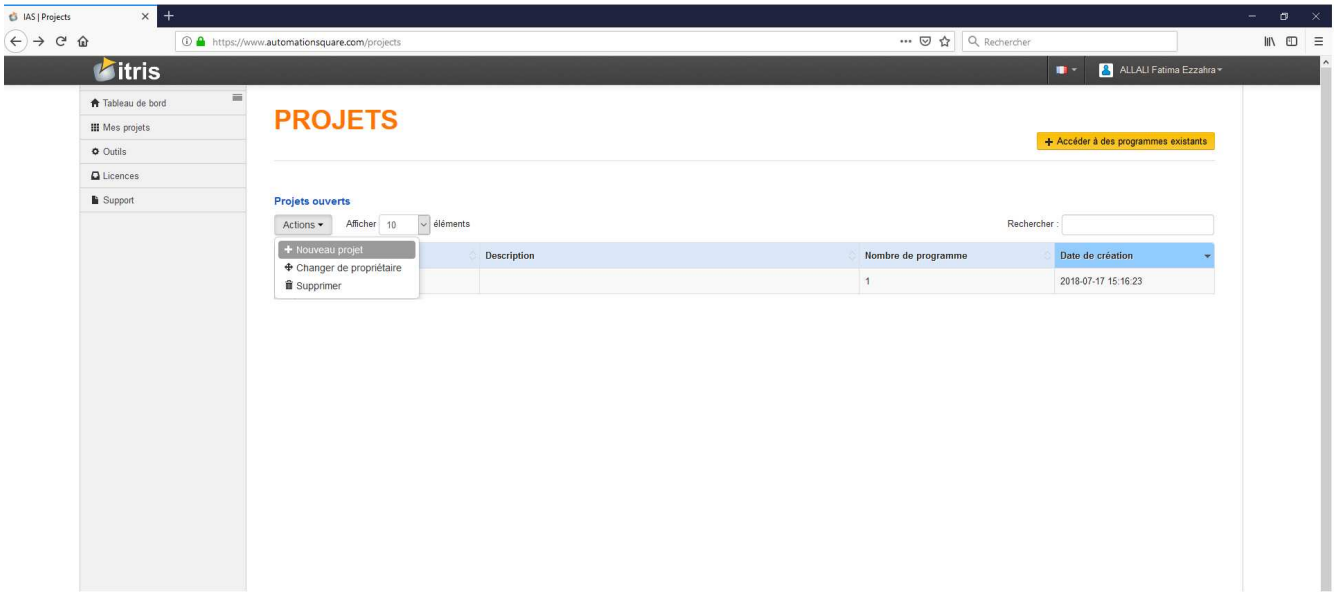


Figure 4.10: Créer un nouveau projet

- La page ci-dessous permet de saisir un nom et une description du projet. Veuillez cliquer sur "Ajouter" pour finir la saisie.

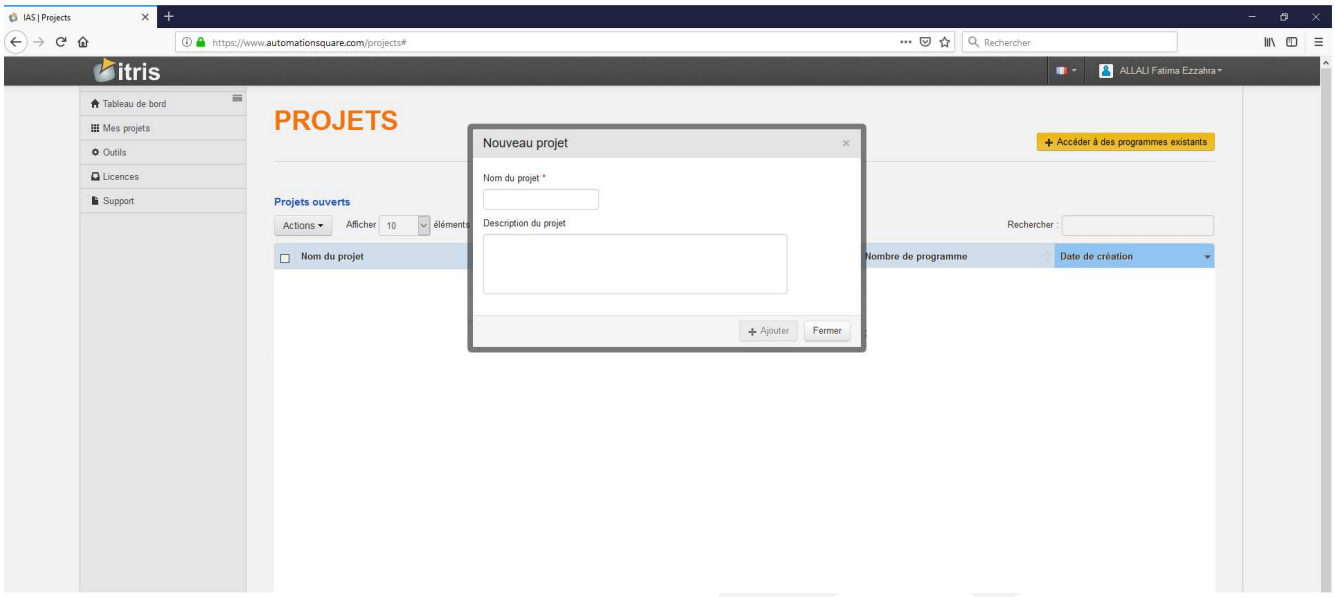


Figure 4.11: Saisie d'un nouveau projet

- Accéder à la page des applications (Onglet "Applications")

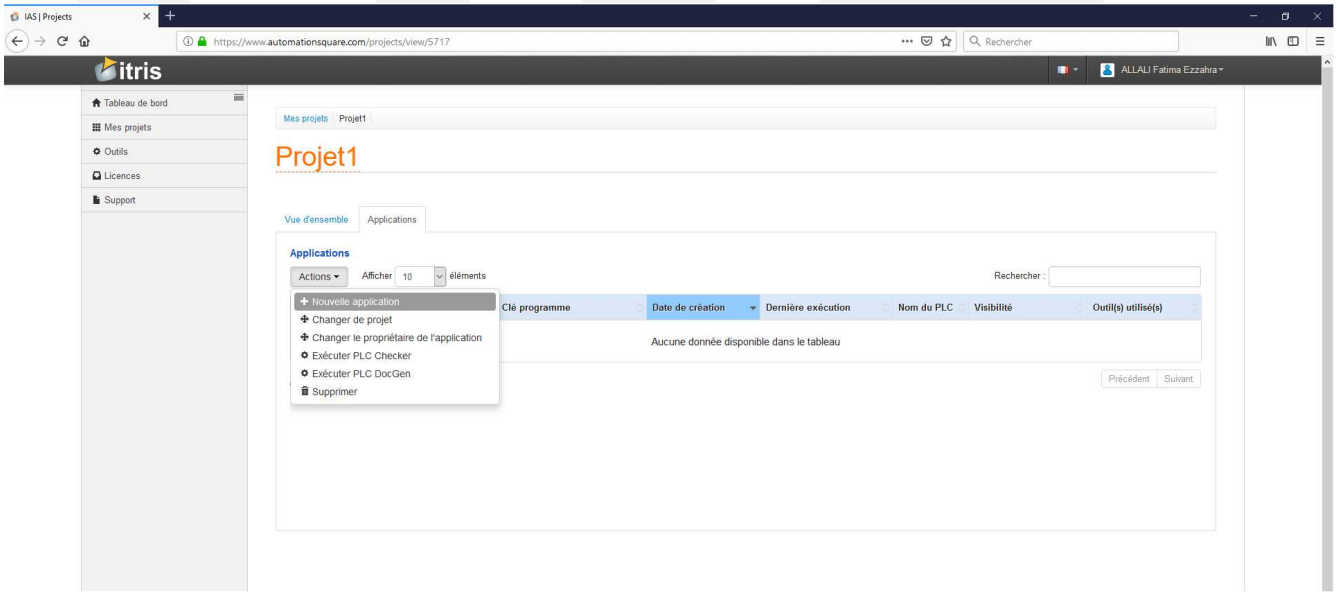


Figure 4.12: Nouvelle application

Quand vous cliquez sur Nouvelle application, veuillez choisir l'outil (dans ce cas Plc Checker) sur la page qui s'ouvre et valider en cliquant sur "Lancement de l'assistant".

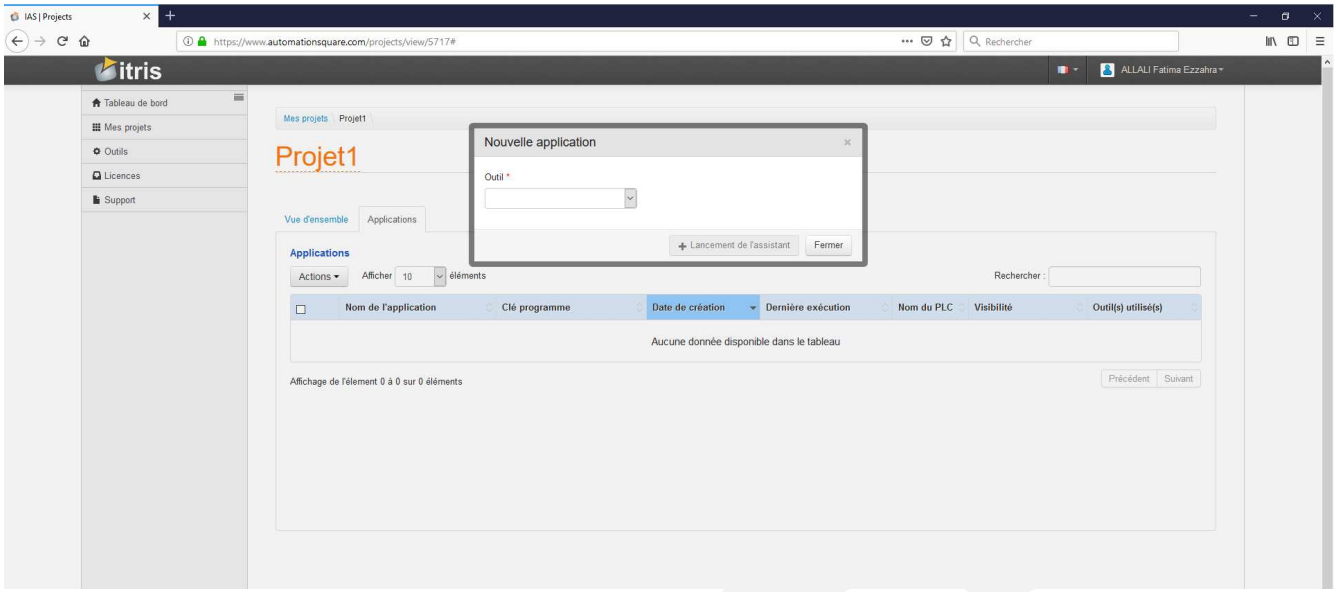


Figure 4.13: Création d'une nouvelle application

En cliquant sur "Lancement de l'assistant, vous serez dirigés vers la page de l'assistant Plc Checker détaillée dans le chapitre précédent Création avec assistant d'une application PLC Checker par défaut sur le site d'Itris Automation Square

4.1.3 Accès à un programme existant de type PLC Checker sur le site d'Itris Automation Square

- Accéder à la page des projets (onglet "Mes projets" dans le menu à gauche).

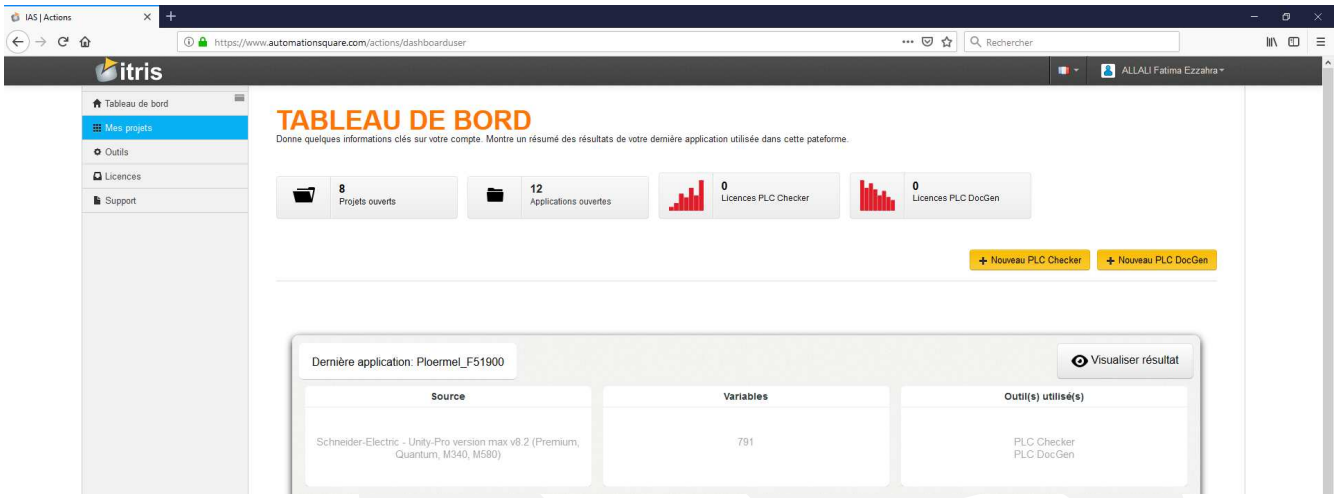


Figure 4.14: Accéder à la page des projets

- Saisir le code à 6 caractères du programme existant en cliquant sur "Accéder à des programmes existants"

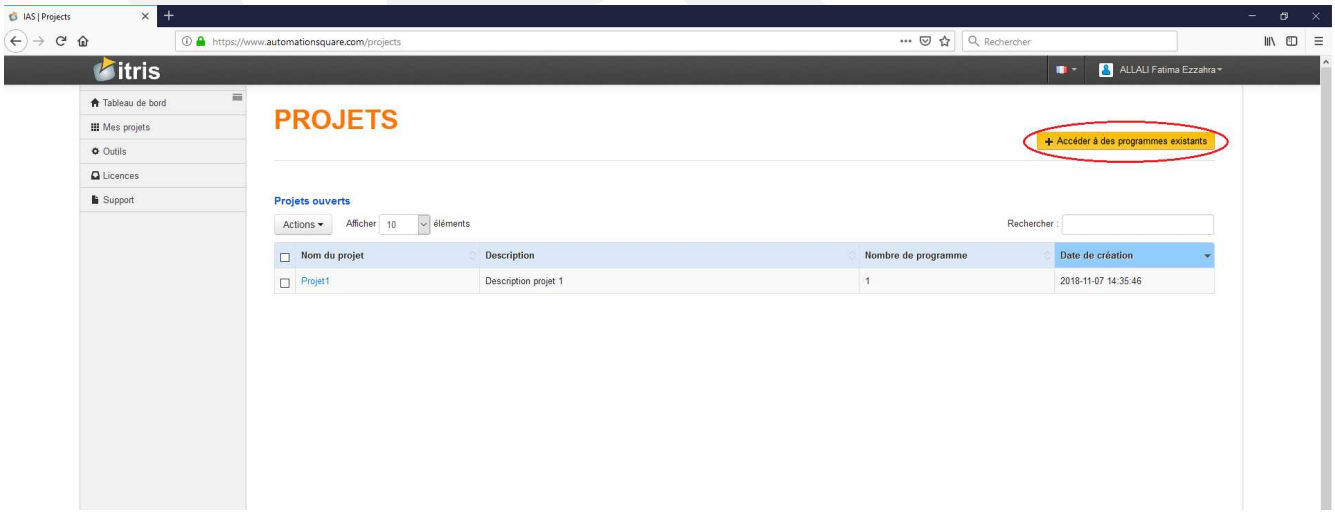


Figure 4.15: Accéder à un programme existant

Veuillez saisir le code à 6 caractères dans la case "clé programme" et cliquer sur "Accéder au programme". Le programme est ensuite ajouté dans le tableau "Applications invitées".

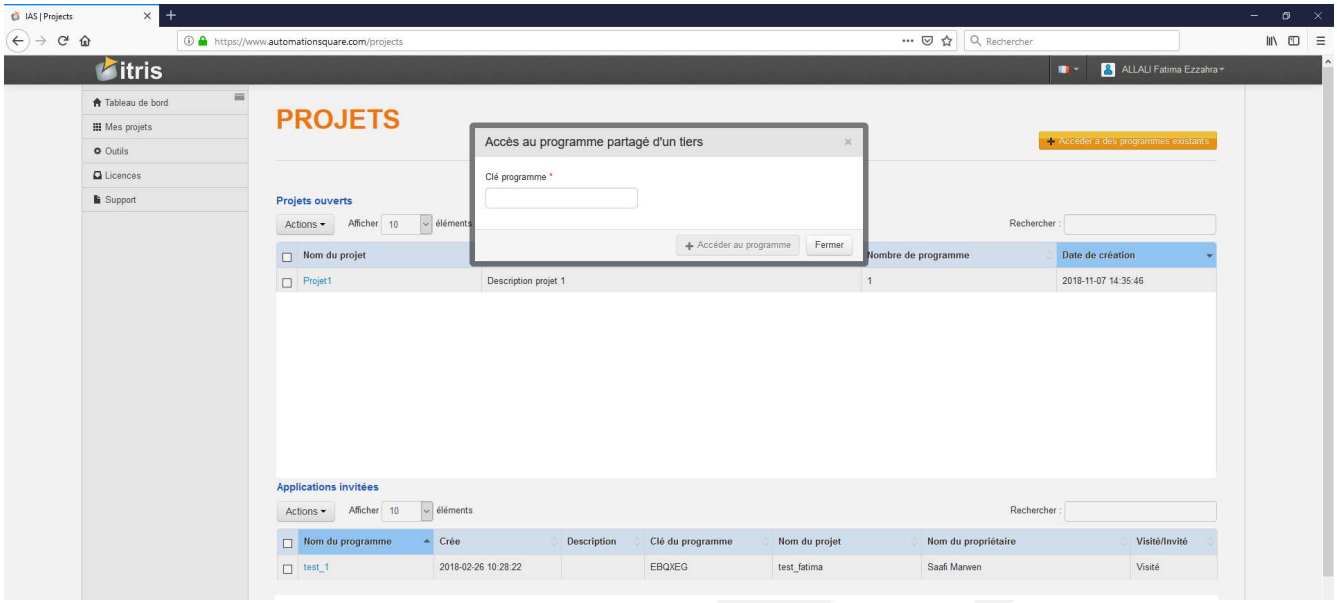


Figure 4.16: Saisie du code programme

4.1.4 Création d'un nouveau projet PLC Checker dans la plateforme Eclipse

Sur la plateforme Eclipse, sélectionner un nouveau projet de type PLC Checker puis cliquer sur "Suivant".

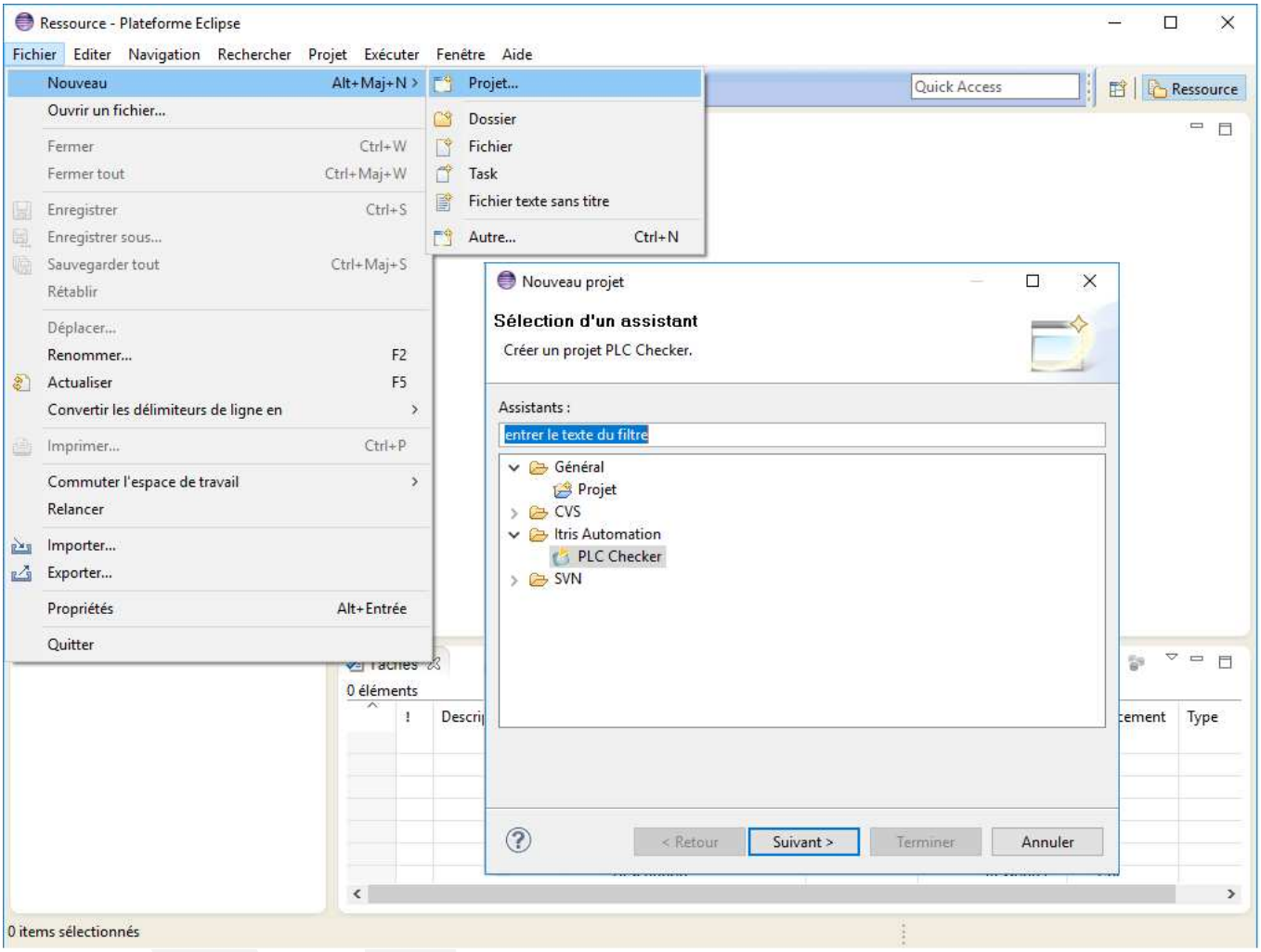


Figure 4.17: Nouveau projet

- Choisir le type d'automate (PL7-Pro, Unity, Step 7, RSLogix5000...) dans le champ "Nouveau programme API[1]".
- Saisir les caractéristiques du projet (nom, description, type d'automate) dans les champs "Projet", "Application API[1]" et "Description".
- Associer un droit au programme dans le champ "Droit IAS".
  - Si vous n'avez pas encore de droit disponible, veuillez contacter le commercial@automationsquare.com.
  - Si vous avez des droits disponibles, veuillez en sélectionner un, dans le champ "Droit IAS".

Vous pouvez laisser le champ "Droit IAS" vide afin de simplement créer la structure du projet et du programme. Mais un droit sera demandé ultérieurement si une analyse est requise par un utilisateur sur ce répertoire de programme.

- Choisir un fichier de règles dans le champ "Fichier de règles". Vous pouvez continuer et ajouter le fichier ultérieurement.

[1] Application Programming Interface

- Cliquer sur le bouton "Terminer". La plateforme créé sur votre espace de travail un répertoire avec le nom du projet défini.

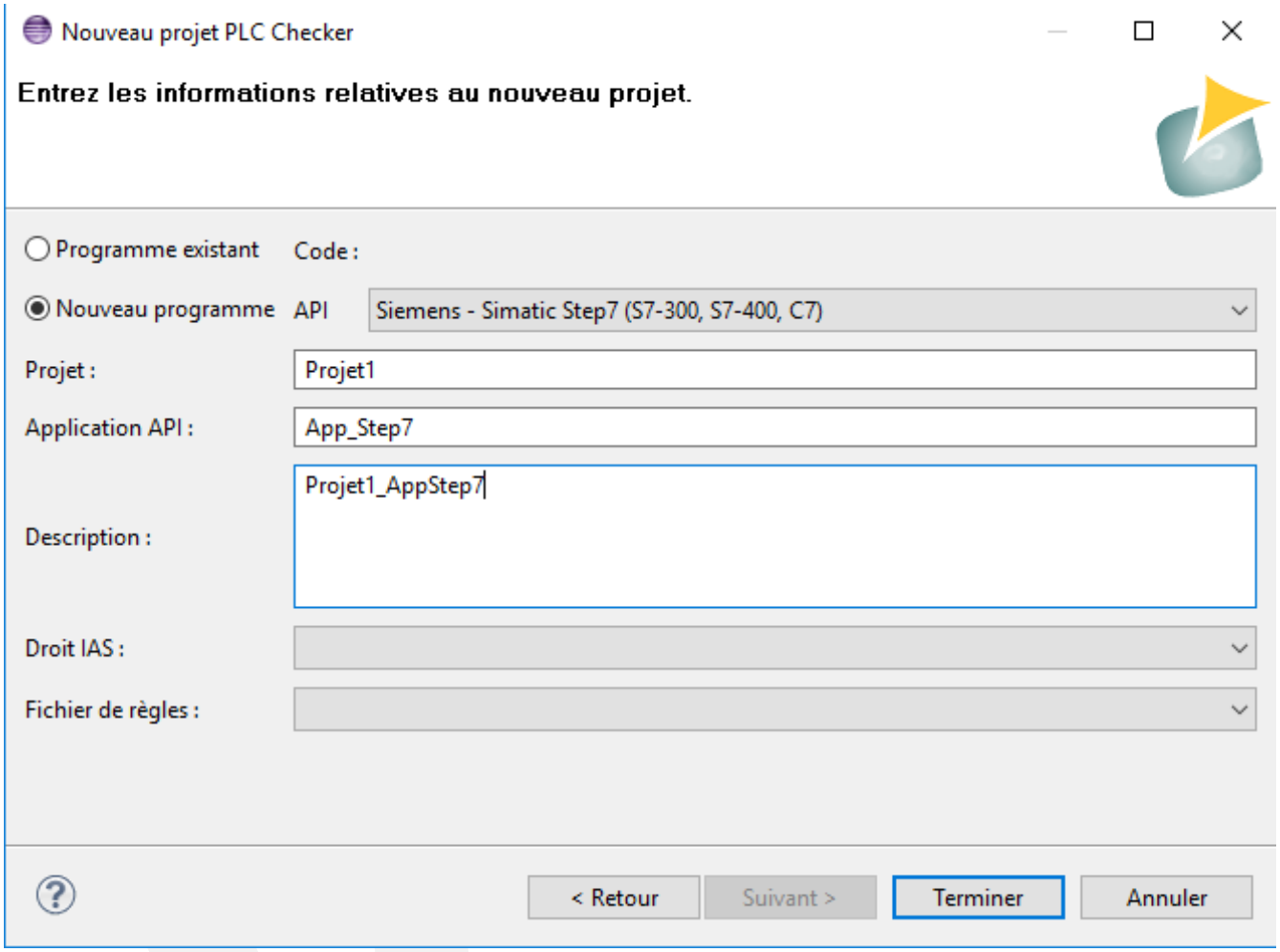


Figure 4.18: Nouveau programme

4.1.5 Création d'un projet PLC Checker existant à partir de la plateforme Eclipse

- Création d'un projet PLC Checker dans la plateforme Eclipse, reprenant le contenu et les caractéristiques du projet précédemment créés sur le site web.

Sur la plateforme Eclipse, sélectionner un nouveau projet de type PLC Checker puis cliquer sur "Suivant".

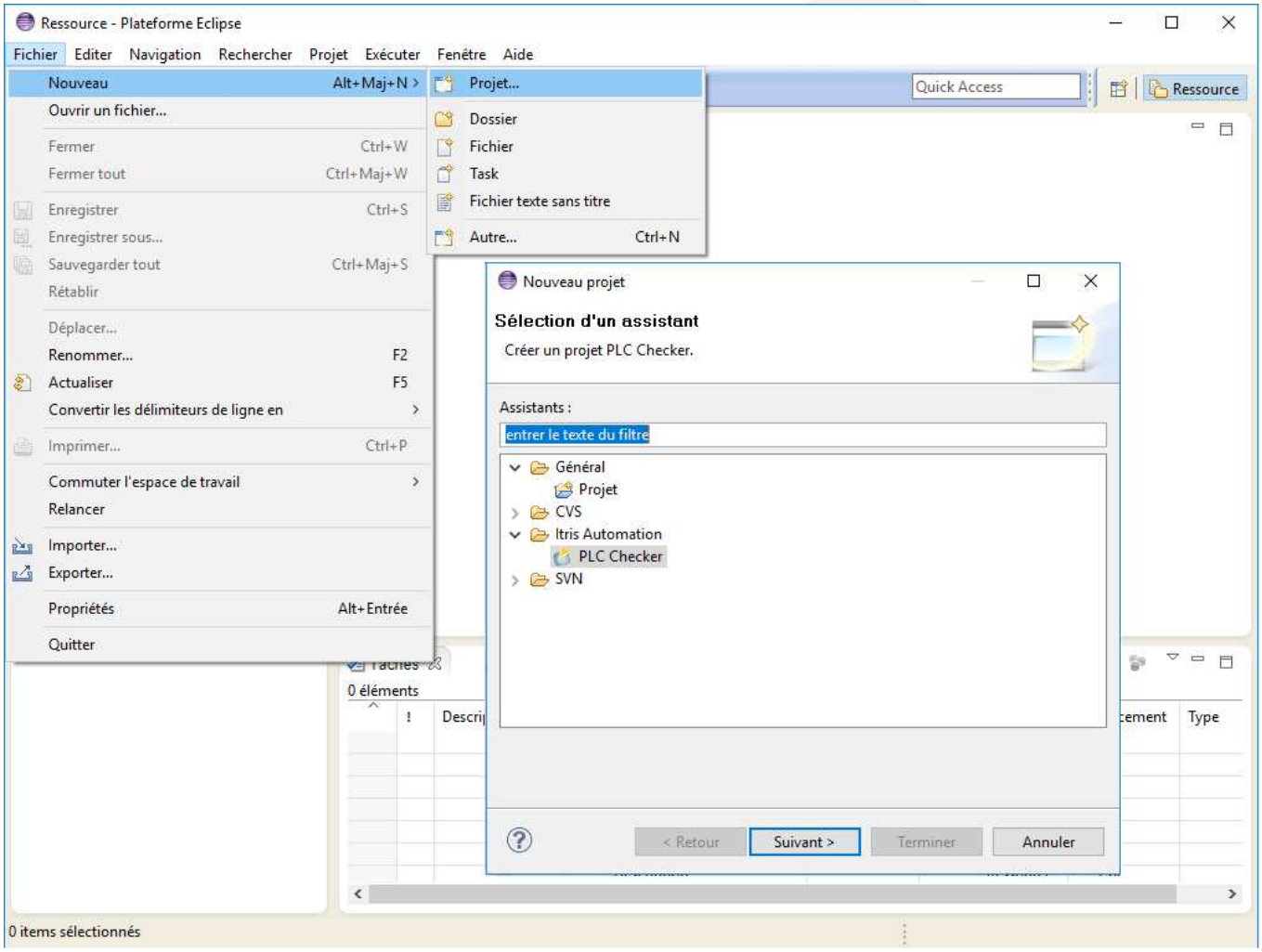


Figure 4.19: Nouveau projet

- Saisir le code programme de votre projet dans le champ "Code". Cliquer sur le bouton "Terminer". La plateforme créé sur votre espace de travail un répertoire reprenant le nom de votre projet défini sur le site web. Elle copie dans ce répertoire les fichiers contenus dans votre projet sur le serveur d'IAS.

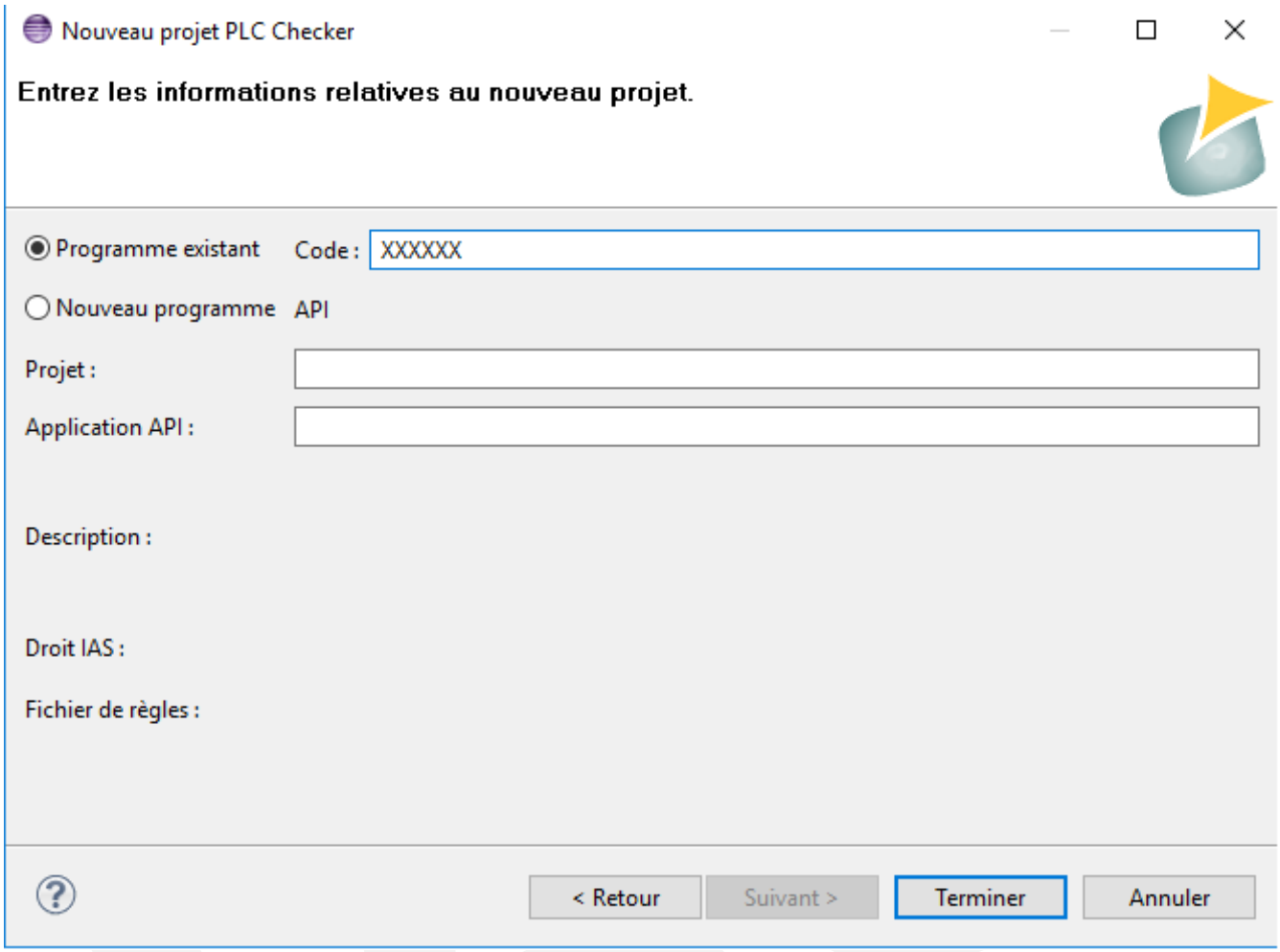


Figure 4.20: Saisie du code programme

4.2 Exécution de PLC Checker

1. Exécution de PLC Checker à partir de la plateforme Eclipse
2. Exécution de PLC Checker à partir du site web :

4.2.1 Exécution de PLC Checker à partir de la plateforme Eclipse

- Import du(des) fichier(s) de l'application automate dans l'espace de travail.
  1. Utiliser l'**explorateur** de fichier de votre OS[2] pour **copier** le(s) fichier(s) d'export automates à utiliser.
  2. **Coller** ce(s) fichier(s) sur la vue **Navigateur** de la plateforme Eclipse.
- Import du fichier de règle dans l'espace de travail.
  1. Utiliser l'**explorateur** de fichier de votre OS[1] pour **copier** le fichier de règles à utiliser.
  2. **Coller** ce fichier sur la vue **Navigateur** de la plateforme Eclipse.

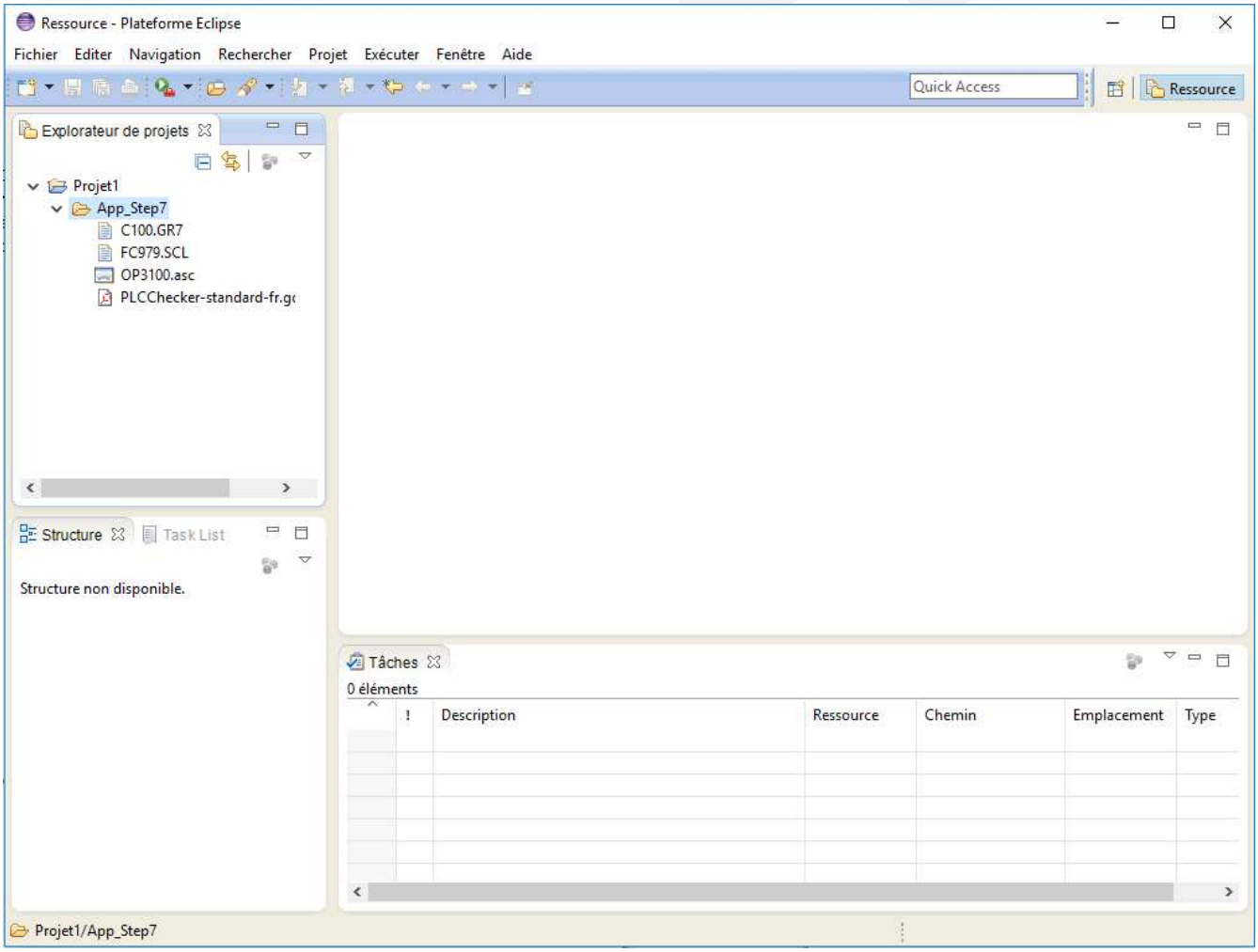


Figure 4.21: Ajout des fichiers dans l'espace de travail

- Exécution du PLC Checker à partir de la vue **Navigateur** de la plateforme Eclipse.

[2] Operating System

1. **Sélectionner les fichiers** automates (*.FEF pour PL7PRO, *.ASC, *.AWL, *.GR7 et *.SCL pour STEP7, *.XEF pour UNITY-PRO, *.L5K pour RSLogix5000) à vérifier ainsi que le fichier de règles (*.GQR) à utiliser lors de l'analyse.

2. Faire un **clic droit** pour afficher le menu contextuel

3. Lancer l'analyse à l'aide du menu contextuel : **Outils GLIPS / Exécuter le PLC Checker pour un programme . . .**

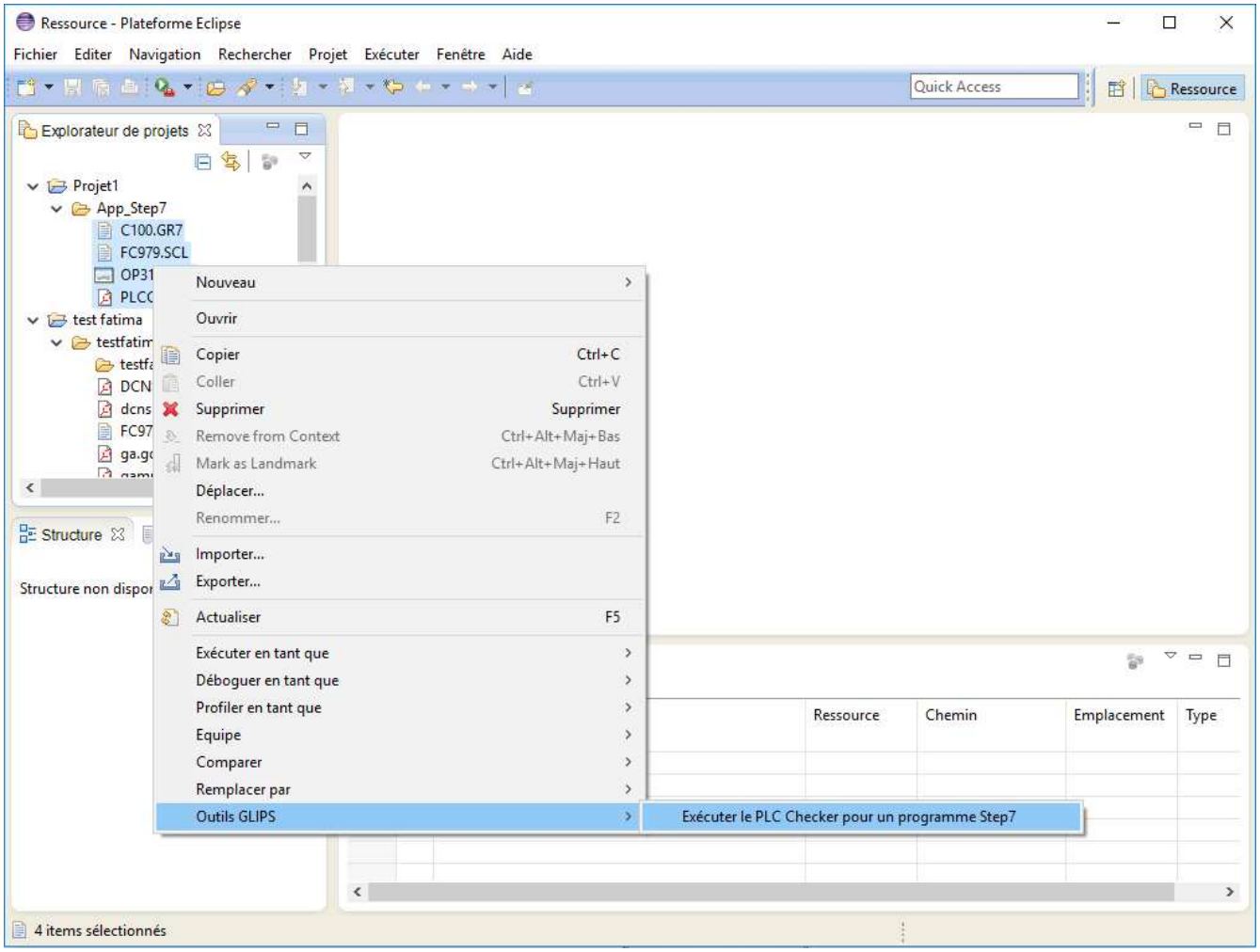


Figure 4.22: Lancer l'analyse Plc Checker

- Si des fichiers comportent des modifications non encore sauvegardées, cette boîte de dialogue apparaît.

```
<html>
<br/>
</html>
```

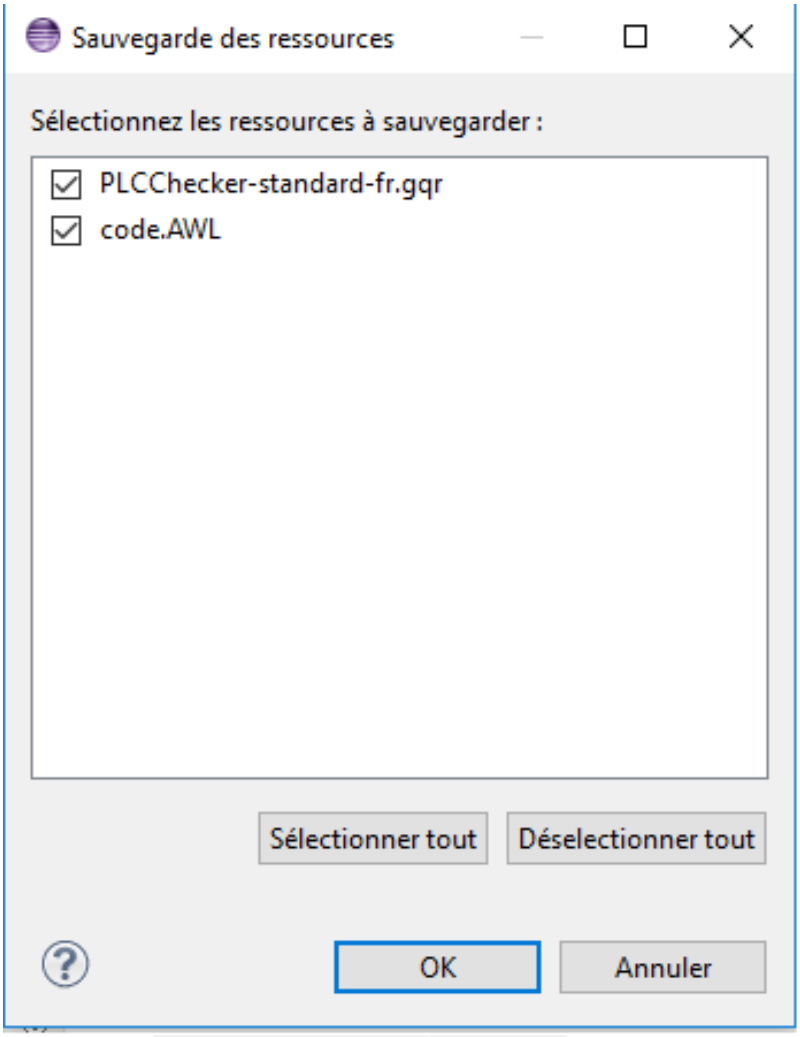


Figure 4.23: Sauvegarde des fichiers modifiés

– Par défaut cliquez sur **Désélectionner tout**, pour ne pas tenir compte des modifications non encore sauvegardées.

– Si vous êtes en train d'éditer le fichier de règles sélectionné, vous pouvez, en demandant la sauvegarde du fichier de règles, prendre en compte vos dernières modifications lors de cette exécution de PLC Checker.

– Cliquez sur OK pour continuer. Cliquez sur Annuler pour abandonner l'exécution.

• Une boîte de dialogue s'ouvre pour la sauvegarde de fichier résultat (.gqre). Nous avons la possibilité de le renommer ou changer son emplacement.

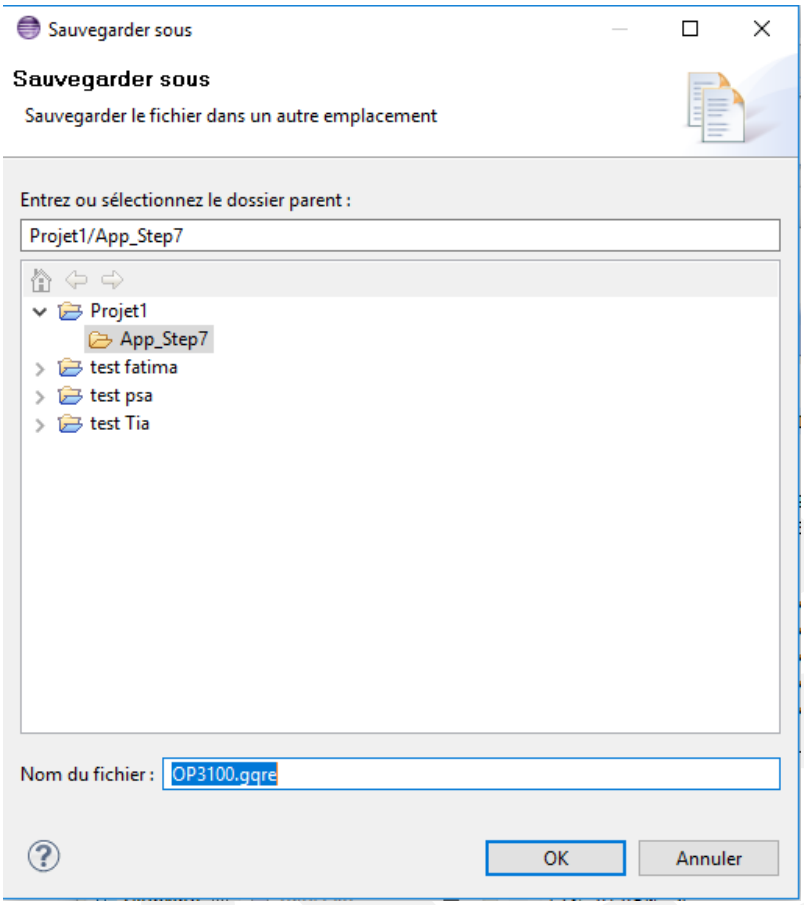


Figure 4.24: Sauvegarde des résultats

- Une boîte de dialogue vous indique le degré d'avancement de l'analyse de votre code. Vous pouvez cliquer sur **Annuler** pour abandonner l'exécution. L'analyse comporte plusieurs phases:
  - Envoi des fichiers (code automate et règles) au serveur
  - Lecture du programme automate
  - Analyse du programme automate selon le fichier de règle
  - Mise à jour du fichier résultat
  - Écriture du fichier résultat
  - Ouverture du fichier résultat

4.2.2 Exécution de PLC Checker à partir du site web :

Veuillez cliquer sur "Exécuter PLC Checker"

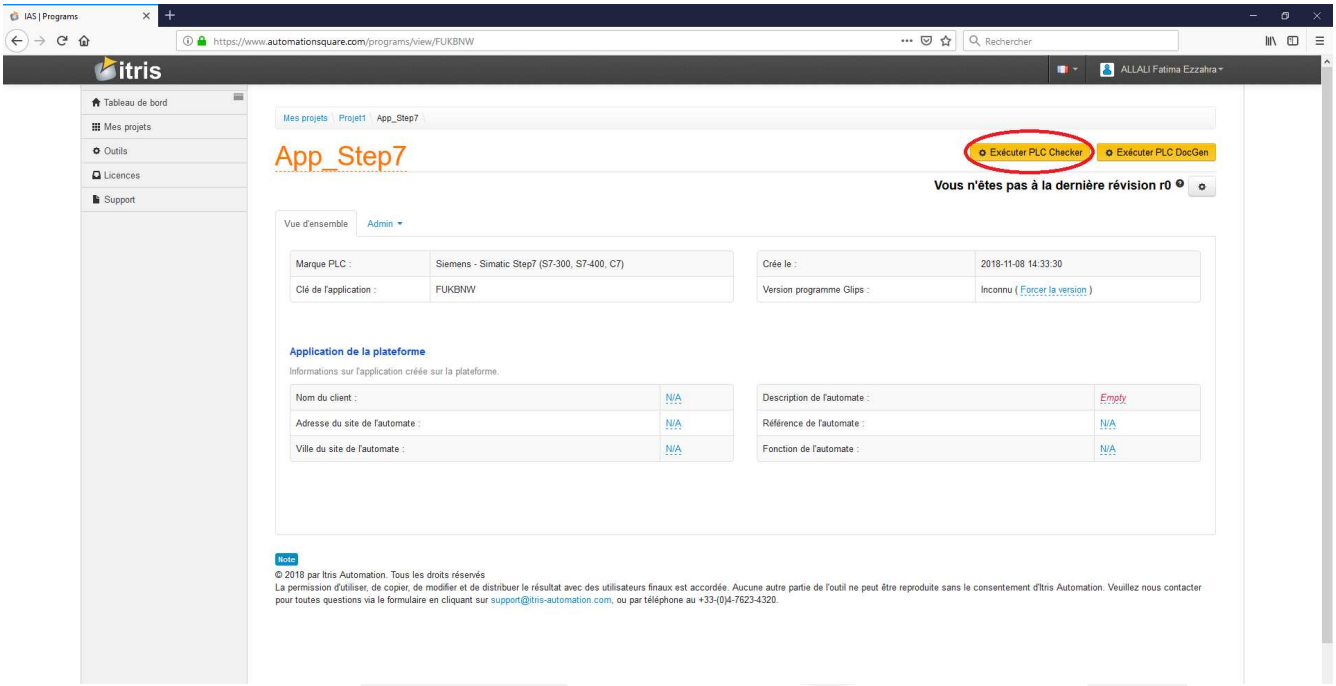


Figure 4.25: Bouton exécution Plc Checker

- Si vous avez déjà téléchargé les fichiers nécessaires pour l'exécution (Fichiers sources et fichier de règles), veuillez cliquer sur "Exécuter" pour lancer l'analyse Plc Checker.

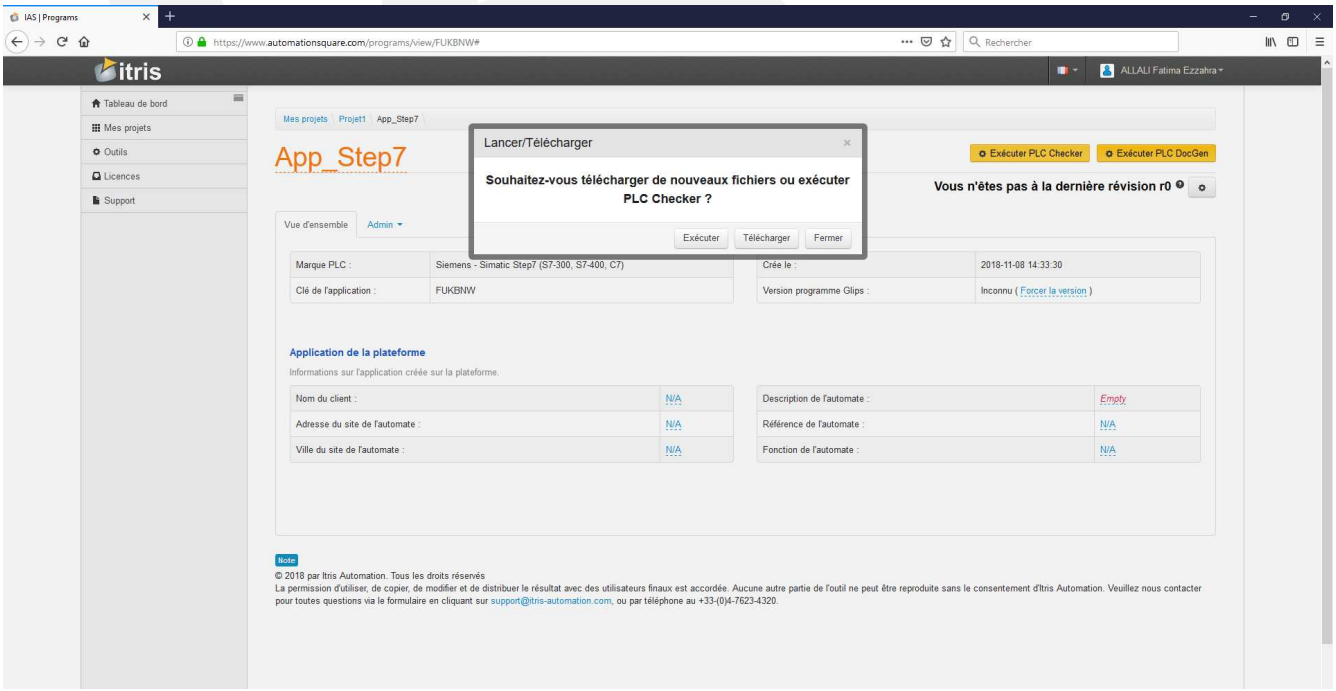


Figure 4.26: Page exécution Plc Checker

- Sinon,veuillez cliquer sur "Télécharger" pour aller à l'étape 4 de l'assistant Plc Checker. A la dernière étape de l'assistant (étape 5), vous pouvez lancer l'analyse en cliquant sur "Exécuter"

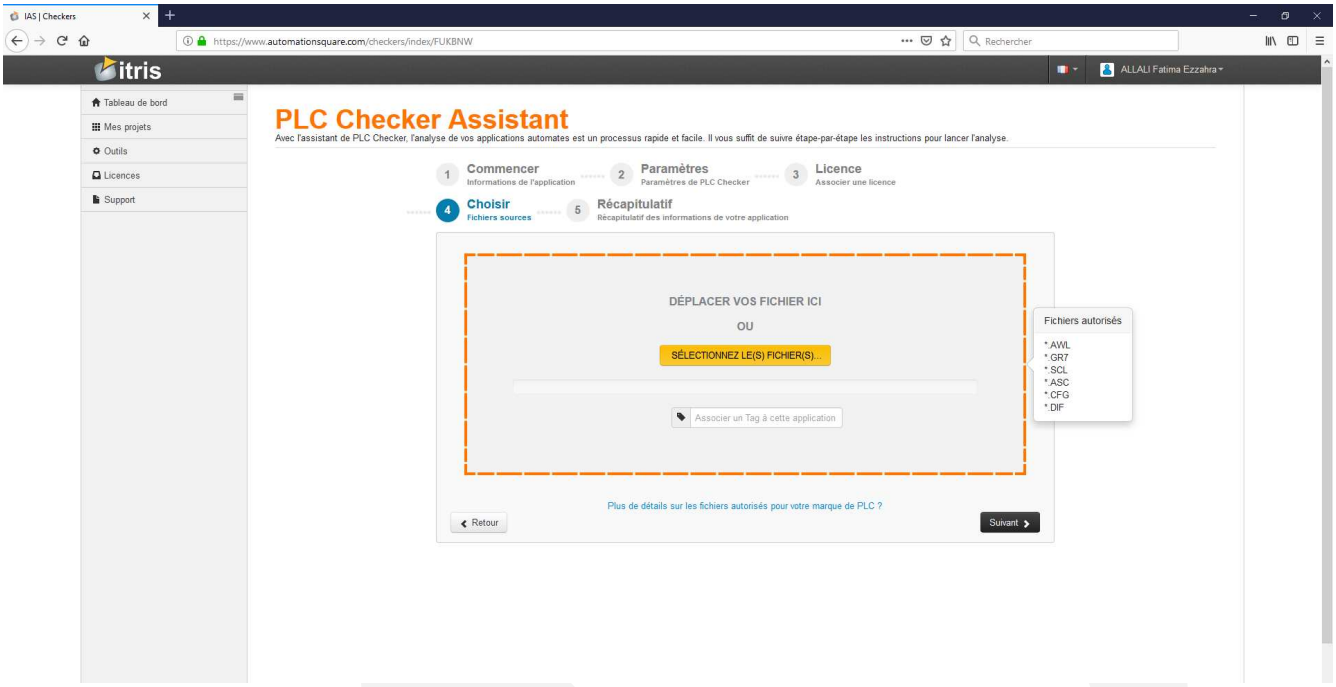


Figure 4.27: L'étape 4 de téléchargement

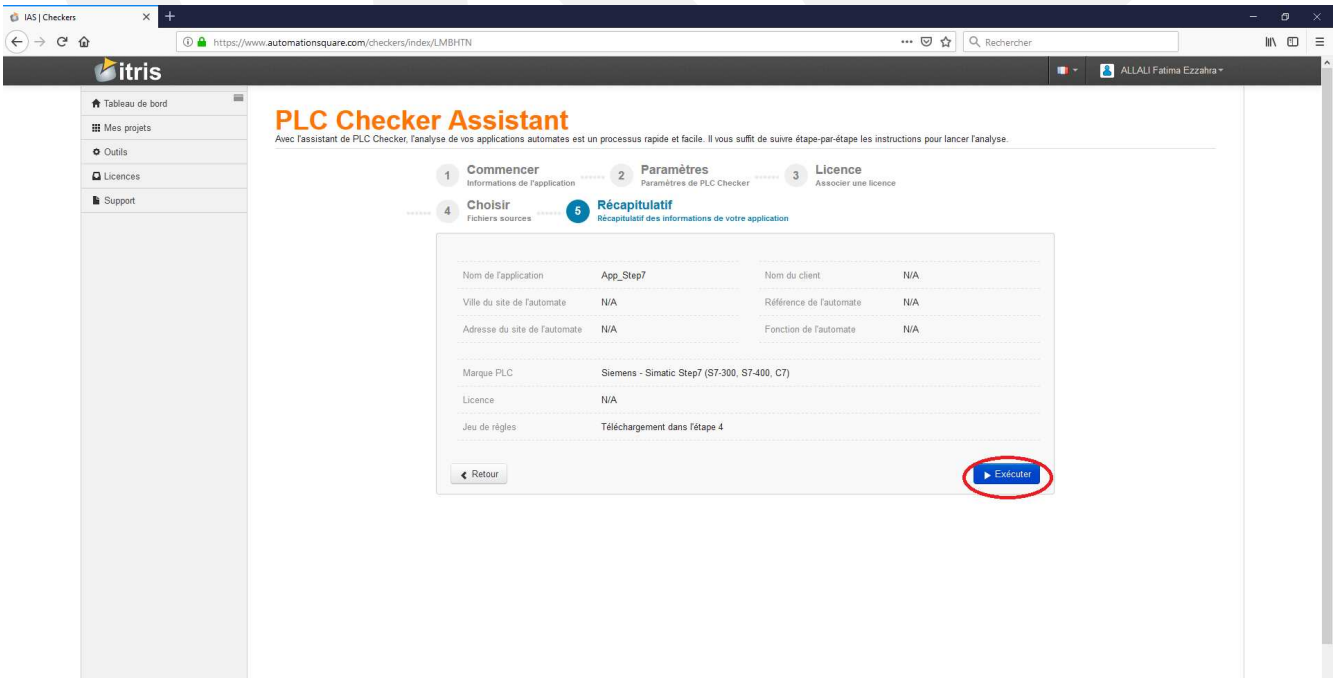


Figure 4.28: Exécution étape5 IAS

4.3 Visualisation des résultats PLC Checker

1. Visualisation des résultats sur la plateforme Eclipse
2. Visualisation des résultats sur le site web

4.3.1 Visualisation des résultats sur la plateforme Eclipse

- Obtention des résultats (**UNIQUEMENT si vous avez utilisé un projet PLC Checker existant (Création d'un projet reprenant les fichiers et les caractéristiques du projet précédemment créés sur le site web)**) :
  1. **Créer un projet** sur la plateforme Eclipse (voir : - Création d'un projet PLC Checker existant)
  2. **Sélectionner le fichier de règle**
  3. Faire un **clic droit** pour afficher le menu contextuel
  4. Lancer la récupération des résultats à l'aide du menu contextuel : **Outils GLIPS / Obtenir les résultats de la dernière opération**

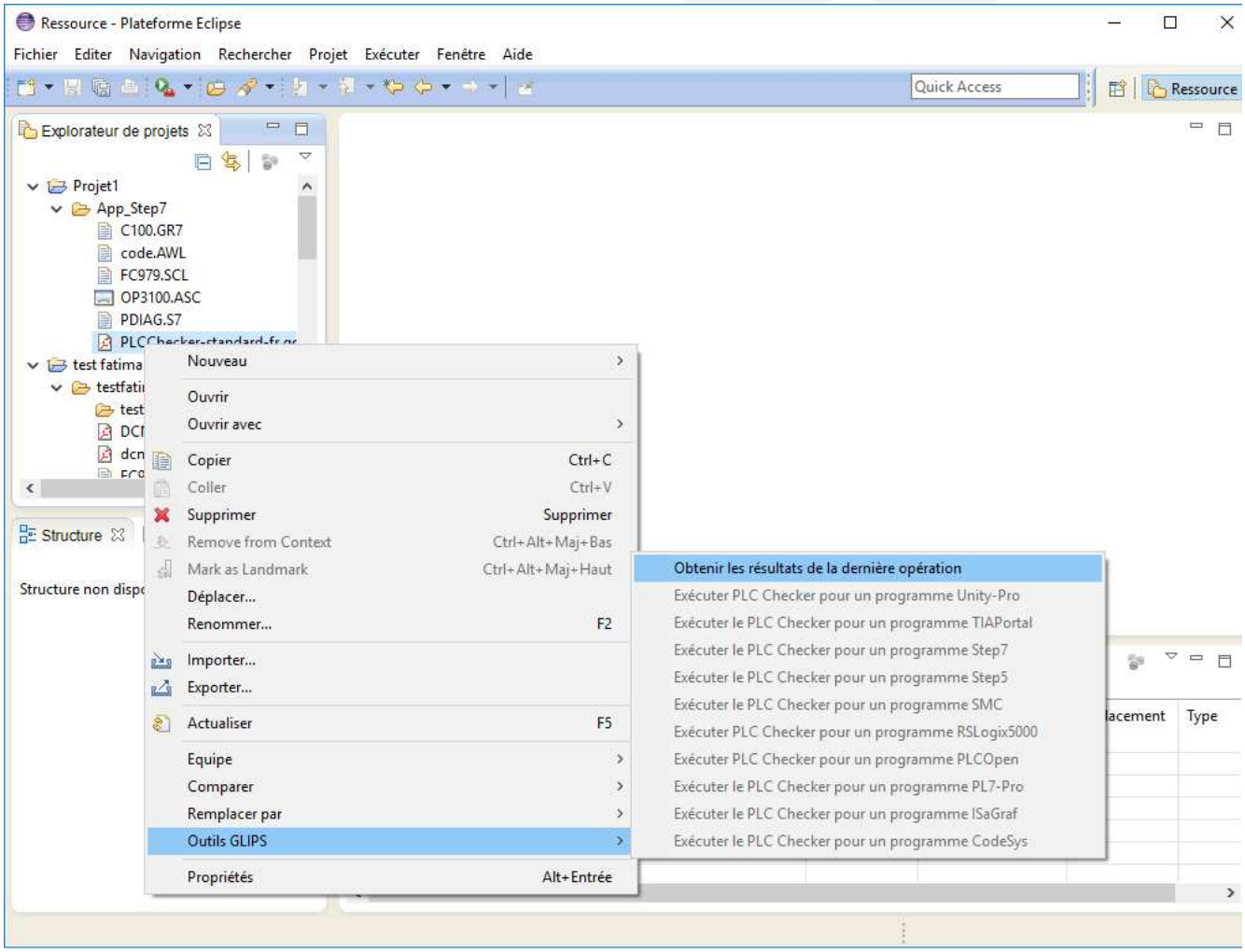


Figure 4.29: Obtenir les résultats de la dernière opération

- Visualisation du fichier résultat (*.gqre) sous forme d'arbre, sur la plateforme Eclipse.
  - La plateforme vous demande sous quel nom vous voulez enregistrer le fichier de résultat. Cliquer sur **OK**.

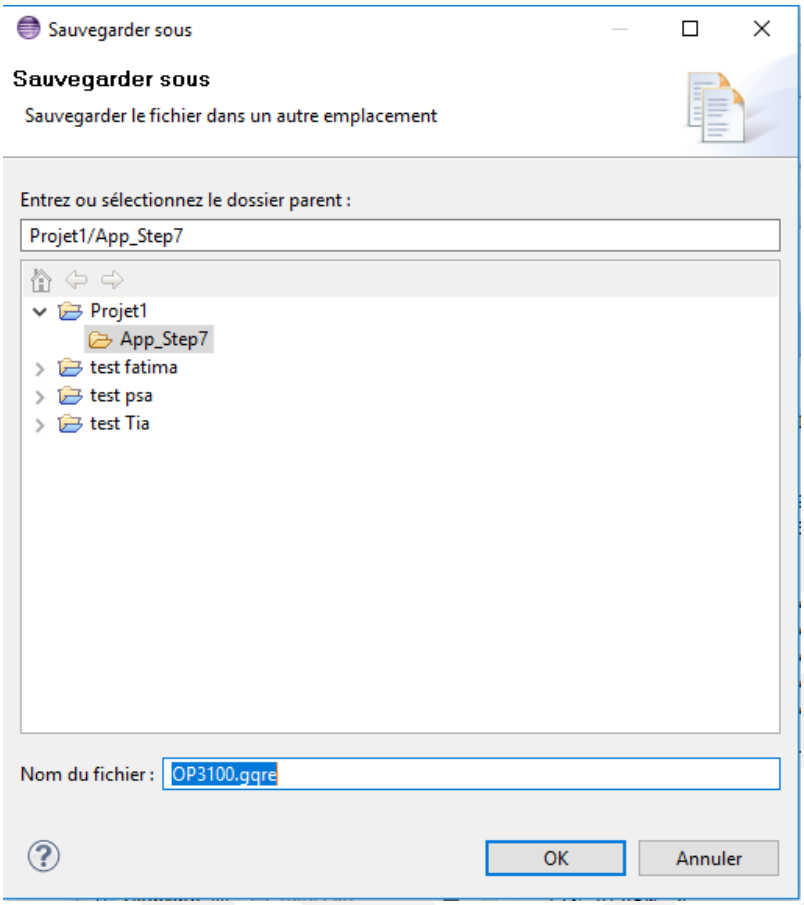


Figure 4.30: Sauvegarde du fichier de résultats

- Si le fichier résultat existe déjà, cette boîte de dialogue apparaît.

  – Cliquez sur **Oui** pour écraser le fichier résultat existant et lancer l'exécution de PLC Checker.

  – Cliquez sur **Non** pour revenir au choix du nom du fichier résultat.

  – Cliquez sur **Annuler** pour abandonner l'exécution.

- Le fichier de résultat (*.gqre) s'affiche dans l'éditeur de résultat de la plateforme Eclipse.

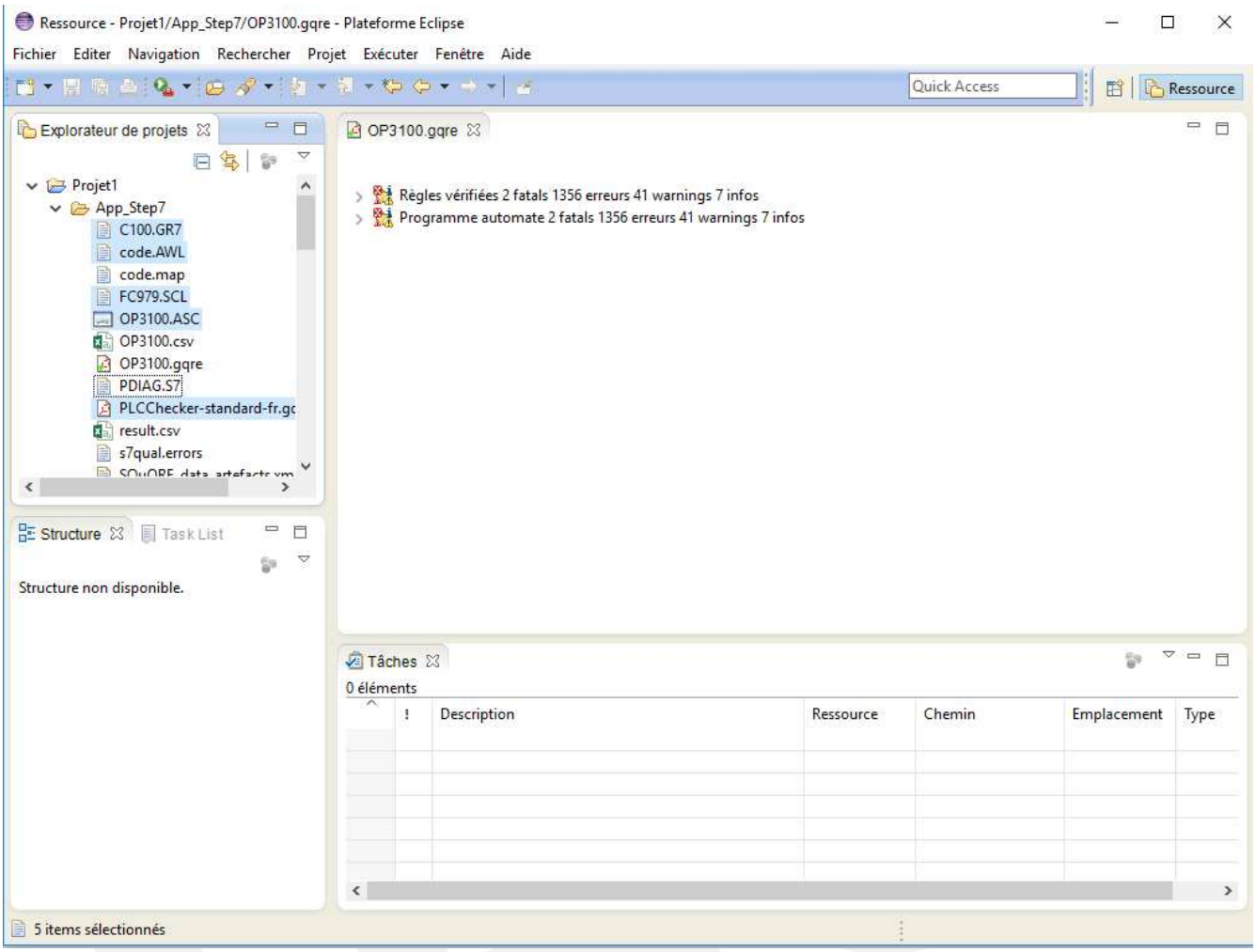


Figure 4.31: Fichier gqre de résultats

4.3.2 Visualisation des résultats sur le site web

- Après le lancement de Plc Checker, un onglet avec le nom "PLC Checker" s'ajoute dans la page de l'application.

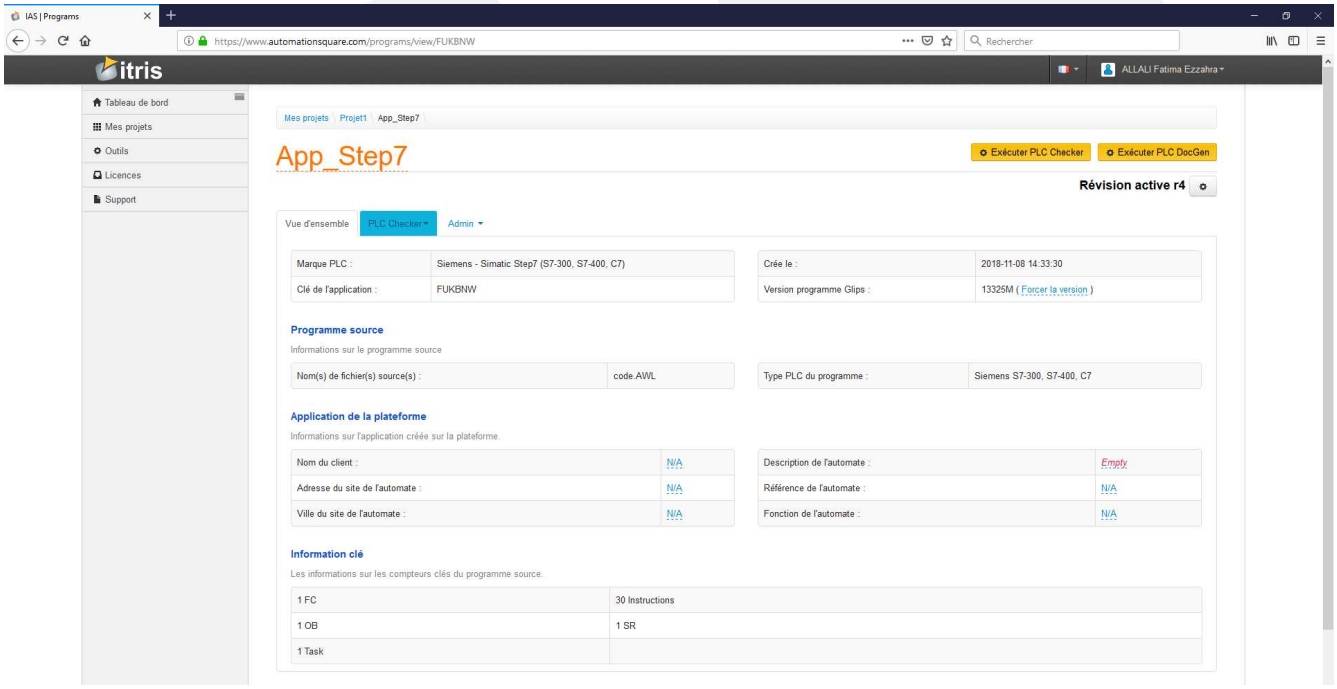


Figure 4.32: Onglet PLC Checker

- Pour obtenir une analyse des résultats, veuillez cliquer sur PLC Checker/Tableau de bord. Vous trouvez des graphiques montrant l'évolution des résultats entre les différentes analyses et une classification des résultats par catégorie de règle et par sévérité.

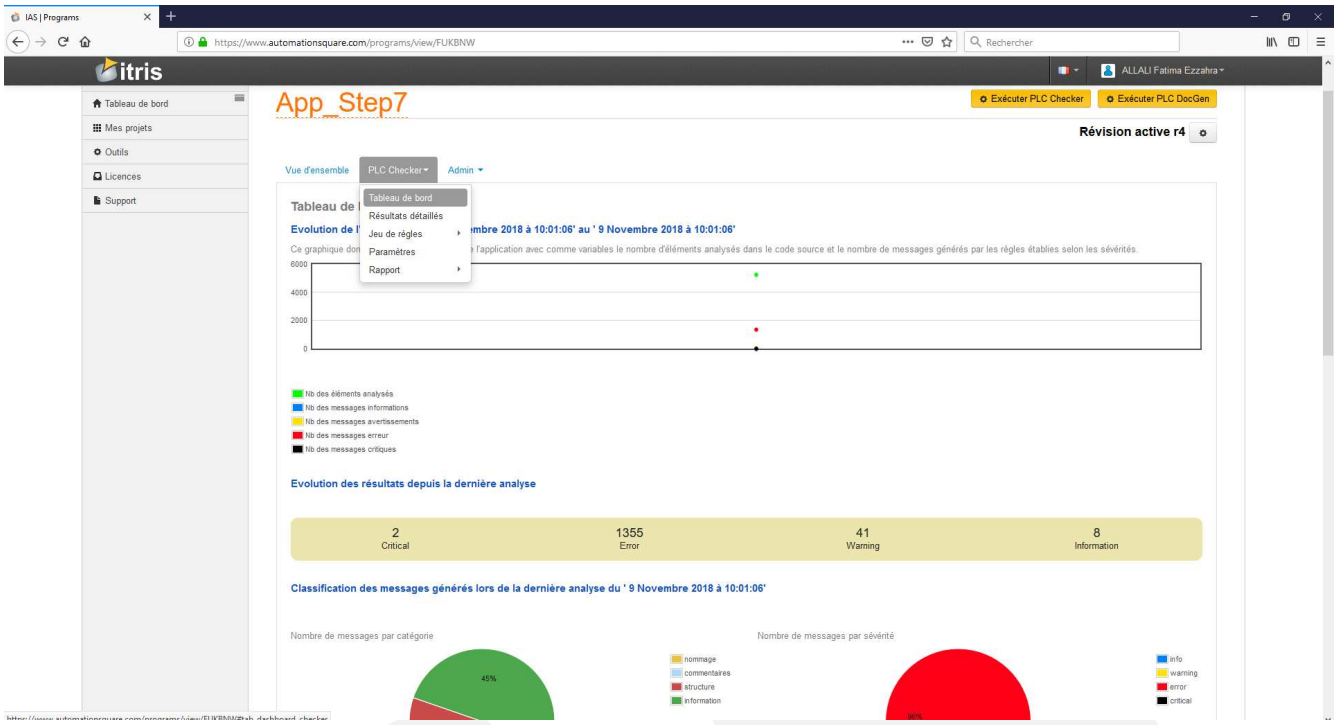


Figure 4.33: Tableau de bord

- Pour visualiser les résultats détaillés, veuillez cliquer sur PLC Checker/Résultats détaillés. vous trouvez la liste des résultats par catégorie de règle.

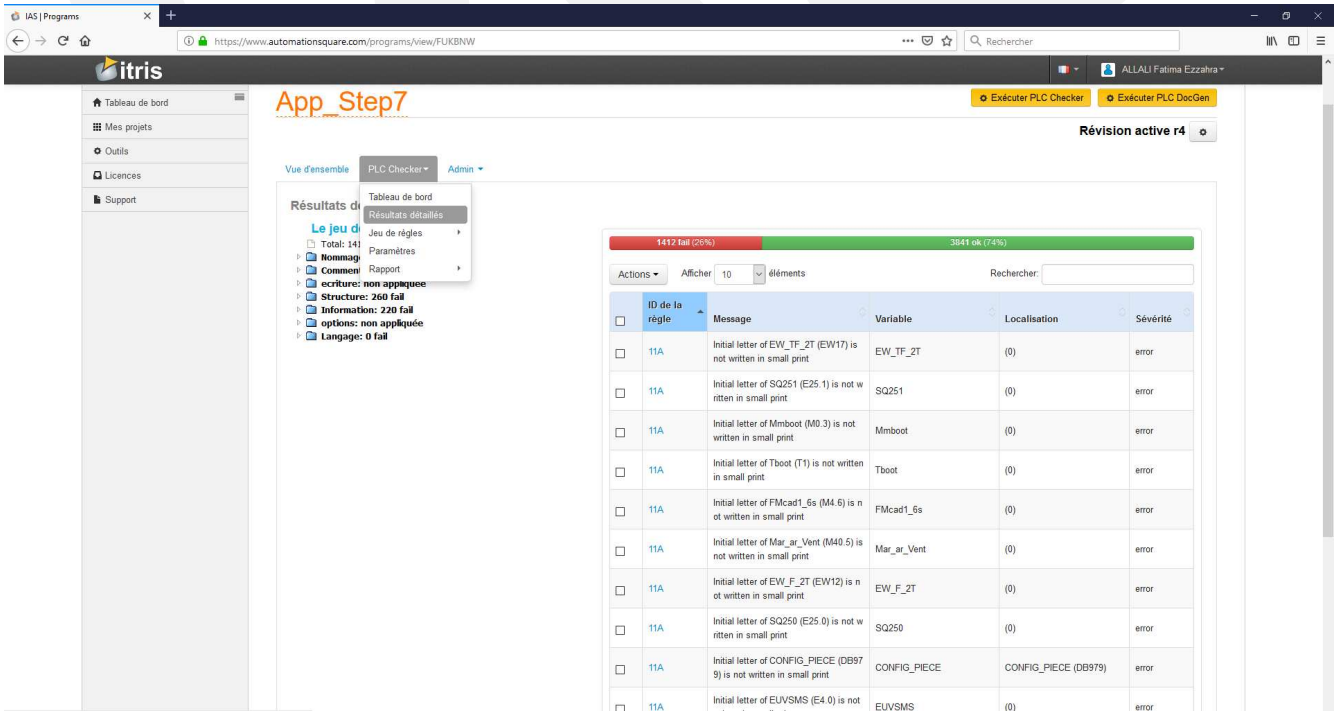


Figure 4.34: Page des résultats détaillés IAS

Chapter 5

Export des fichiers automates

5.1 Création d'un fichier FEF PL7-Pro

Pour analyser une application PL7-Pro avec les outils d'Itris Automation (ex. PLC Checker, PLC docGen), vous devez exporter cette application dans un fichier FEF.

IMPORTANT

Avant de pouvoir exporter votre application, vous devez configurer votre logiciel PL7 PRO de telle manière que les DFB non accessibles en lecture, soient correctement exportées.

Pour ce faire, ajoutez la clé "ExportPl7Converter=True" à la section [PL7TOOL132] du fichier "Pl7sys.ini" que vous trouverez dans le répertoire Windows (en général C:\WINNT ou C:\WINDOWS). La modification ne prendra effet que lors du prochain lancement du logiciel PL7 PRO.

Note: Cette possibilité n'est effective qu'à partir de la version PL7 PRO 4.5.

Vous pouvez créer un fichier FEF à l'aide du menu **Fichier** > **Exporter l'application...** dans la barre de menus du logiciel PL7 PRO.

1. Dans la barre de menus, sélectionnez **Fichier** > **Exporter l'application...**

2. Dans l'assistant **Exporter une application**, donnez un nom au fichier FEF

3. Ne changez pas le répertoire où le fichier FEF sera créé, cliquez sur "Enregistrer"

Avant de pouvoir l'analyser avec PLC Checker, vous devrez l'importer dans l'espace de travail de la plateforme Eclipse.

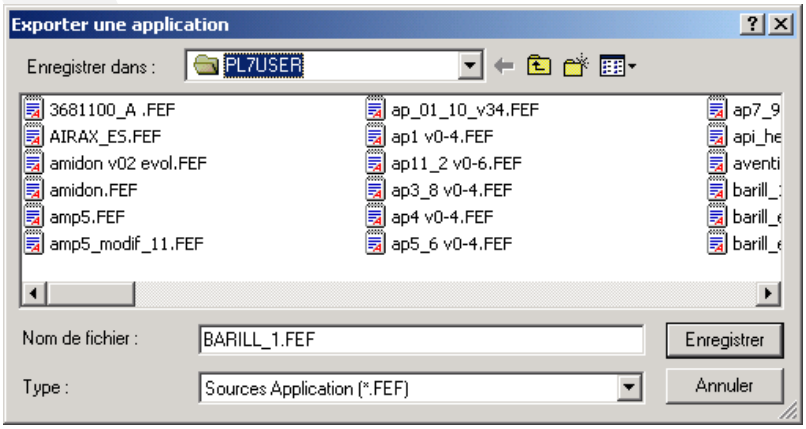


Figure 5.1: Export du fichier FEF

5.2 Création d'un fichier L5K RSLogix5000

Pour analyser une application RSLogix5000 avec PLC Checker, vous devez exporter cette application dans un fichier L5K.

Vous pouvez créer un fichier L5K à l'aide du menu **Fichier** > **Enregistrer sous ...** dans la barre de menus du logiciel RSLogix5000.

1. Dans la barre de menus, sélectionnez **Fichier** > **Enregistrer sous...**
2. Dans l'assistant "Enregistrer sous", sélectionner le type de fichier **RSLogix 5000 Import/Export File (*.L5K)** et donner un nom au fichier L5K
3. Ne changez pas le répertoire où le fichier L5K sera créé ; cliquez sur **Enregistrer**

Avant de pouvoir l'analyser avec PLC Checker, vous devrez l'importer dans l'espace de travail de la plateforme Eclipse.

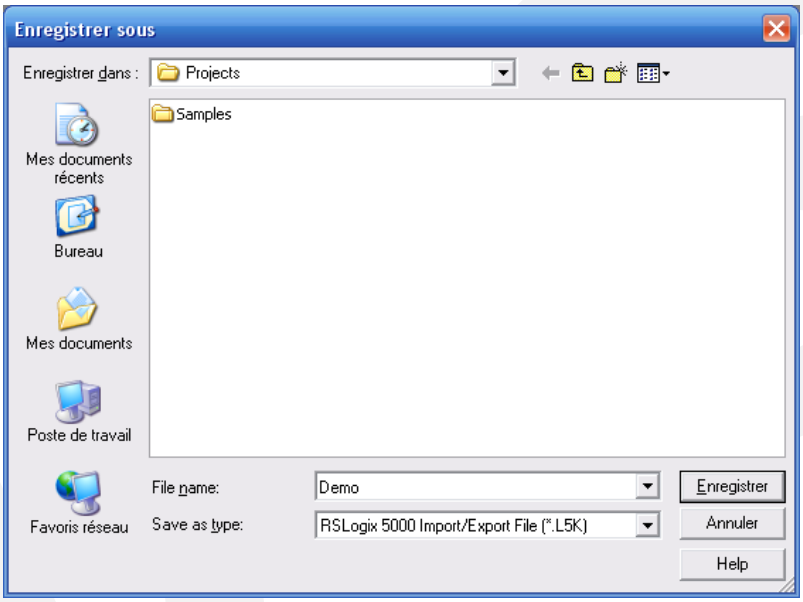


Figure 5.2: Export du fichier L5K

5.3 Création des fichiers .ASC, .AWL, .GR7 et .SCL Step7

Pour analyser une application STEP 7 avec PLC Checker, vous devez :

- exporter les mnémoniques de cette application dans un fichier **.ASC**, en utilisant les abréviations allemandes,
- générer et exporter son code CONT/LIST/LOG dans un fichier **.AWL**, en utilisant les abréviations allemandes,
- générer et exporter les blocs GRAPH dans des fichiers **.GR7**, en utilisant les abréviations allemandes,
- et exporter son code SCL dans des fichiers **.SCL**, en utilisant les abréviations allemandes.
- Pour créer les fichiers **.ASC**, **.AWL**, **.GR7** et **.SCL** vous devez tout d'abord vous assurer que votre atelier STEP7 utilise les abréviations allemandes.(menu **Outils** > **Paramètres...**)

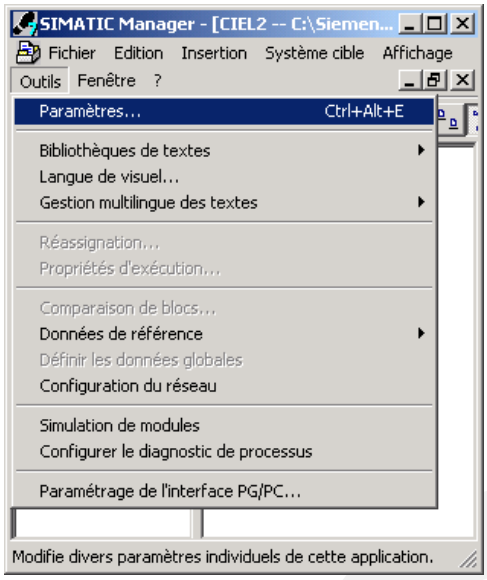


Figure 5.3: Paramétrer l'atelier STEP7

- Sélectionnez l'onglet **Langue**, puis les abréviations allemandes et cliquez sur **OK**.

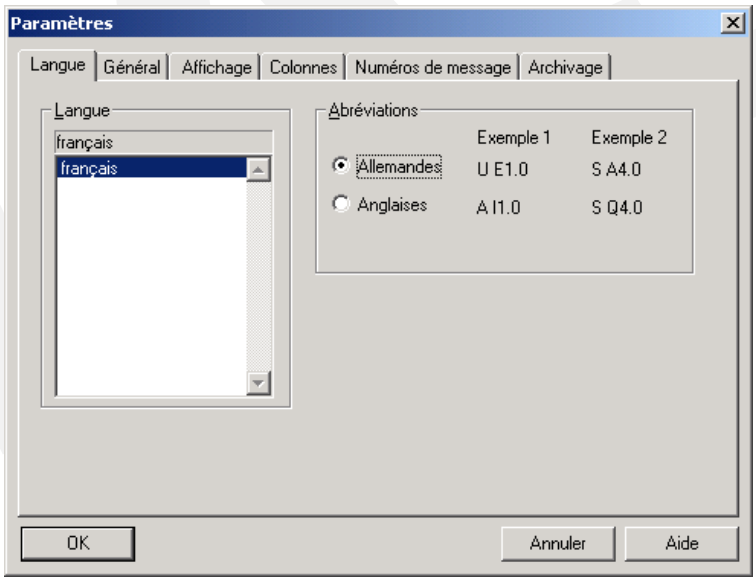


Figure 5.4: Sélectionner les abréviations allemandes

5.3.1 Export du Fichier .ASC

- Pour créer un fichier **.ASC**, vous devez tout d'abord ouvrir l'éditeur de mnémoniques.

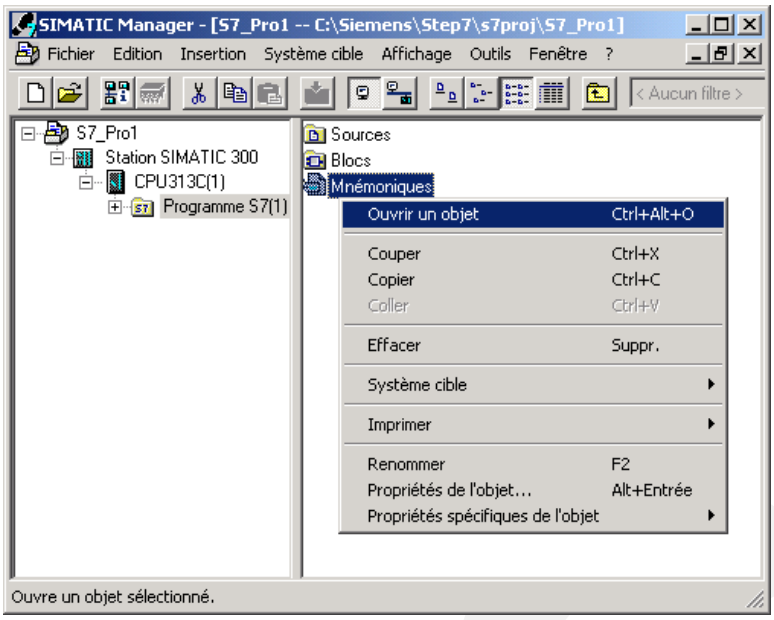


Figure 5.5: Ouvrir la table des mnémoniques

- Dans l'éditeur de mnémoniques sélectionnez le menu **Table** > **Exporter…**

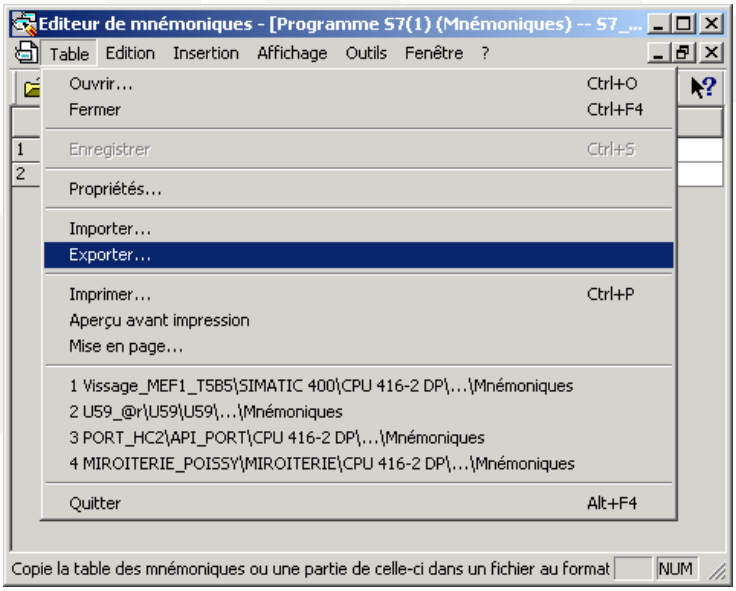


Figure 5.6: Exporter la table des mnémoniques

- Choisissez le type Format ASCII[1] (*.ASC), le nom et le répertoire du fichier **.ASC** à créer, et cliquez sur le bouton **Enregistrer**.

[1] American Standard Code for Information Interchange

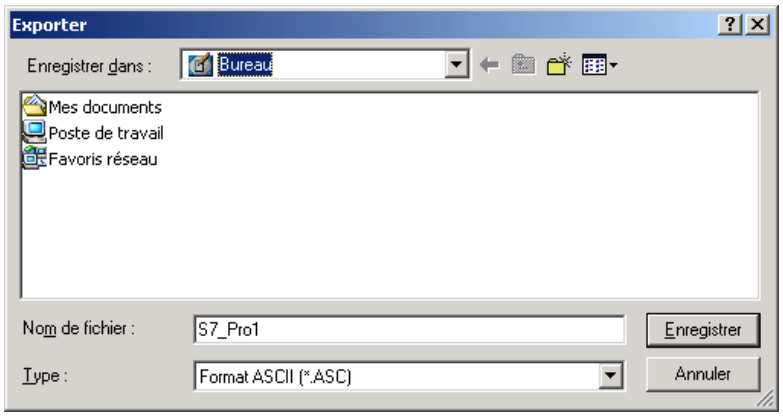


Figure 5.7: Donner le nom du fichier .ASC

5.3.2 Génération du Fichier .AWL

- Pour créer le fichier **.AWL** vous devez générer le source LIST. Ouvrez un éditeur de code CONT/LIST/LOG.

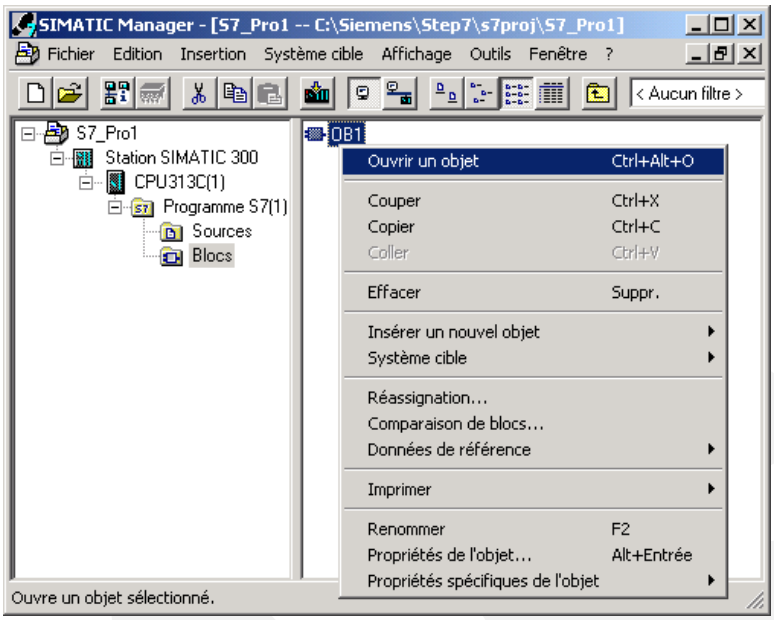


Figure 5.8: Ouvrir l'éditeur CONT/LIST/LOG

- Fermez tous les objets en édition (Menu **Fichier** > **Fermer**)

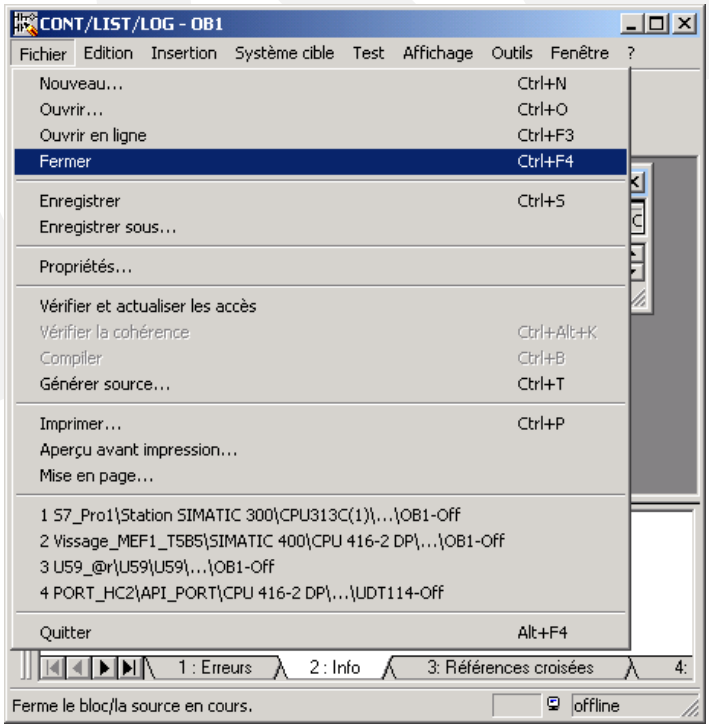


Figure 5.9: Fermer tous les objets en édition

- Demandez la génération du source du programme (Menu **Fichier** > **Générer source...**)

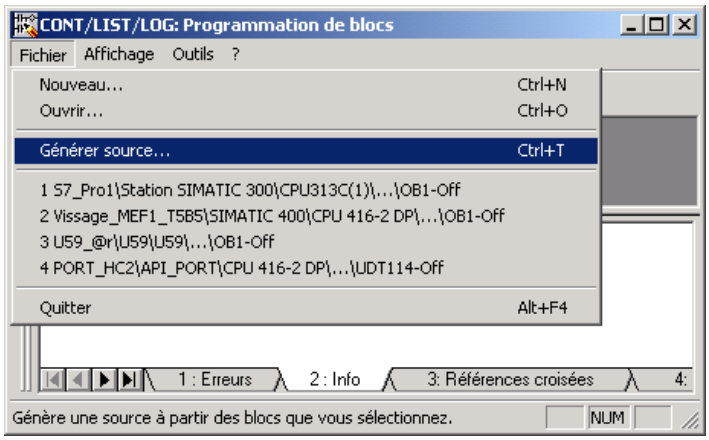


Figure 5.10: Générer le source LIST de l'application

- Donnez le nom de l'objet à générer.

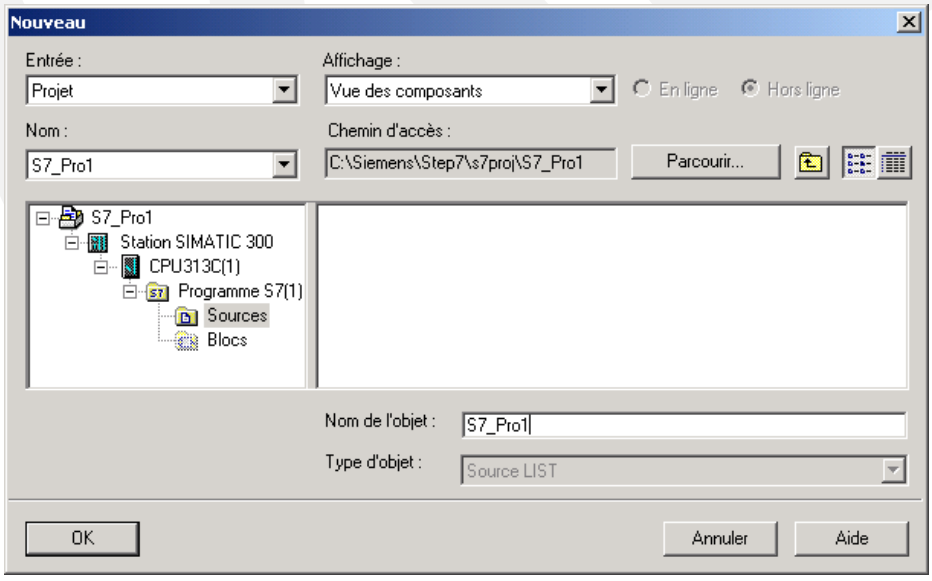


Figure 5.11: Donner le nom du fichier .AWL à générer

- A l'aide du bouton **Tous** > **sélectionnez tous les blocs**, cochez uniquement la case **Tri selon structure de programme**. Sélectionnez les opérandes symboliques et pressez le bouton **OK**.

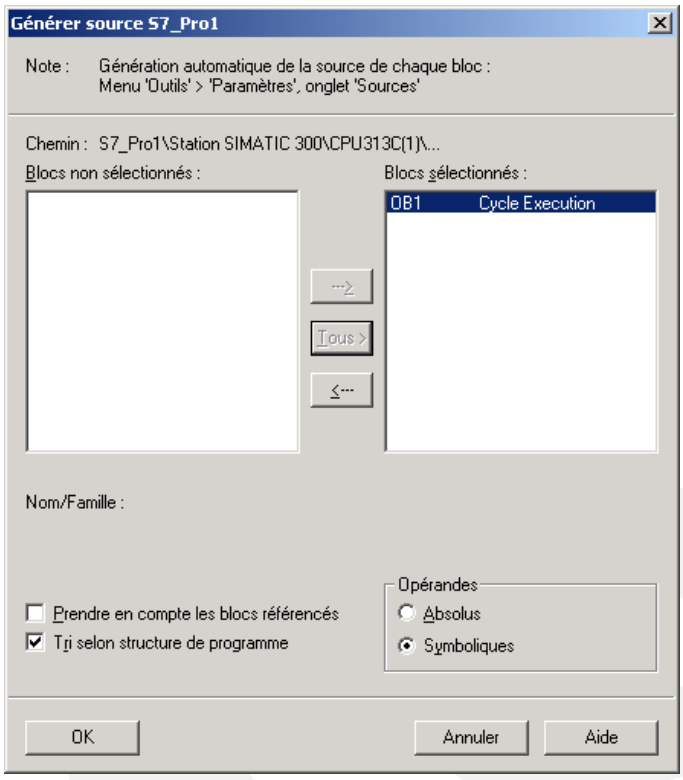


Figure 5.12: Configurer la source à générer

5.3.3 Génération des Fichiers .GR7

- Création des sources GR7. Sélectionnez tous les blocs GRAPH et ouvrez l'éditeur GRAPH

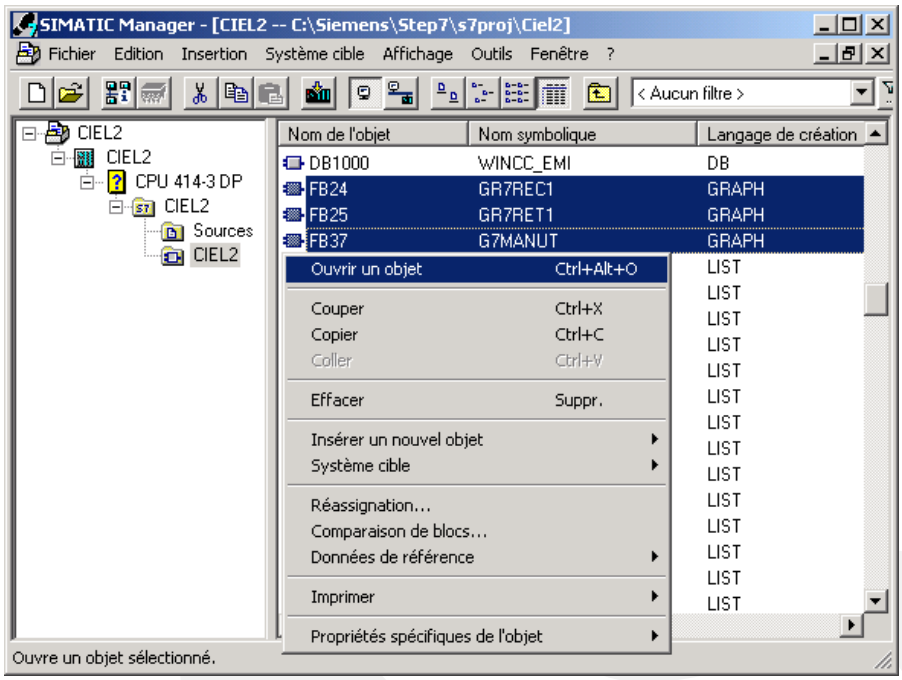


Figure 5.13: Séléctionner tous les blocs GRAPH

- Pour chaque bloc, demandez la génération du source.

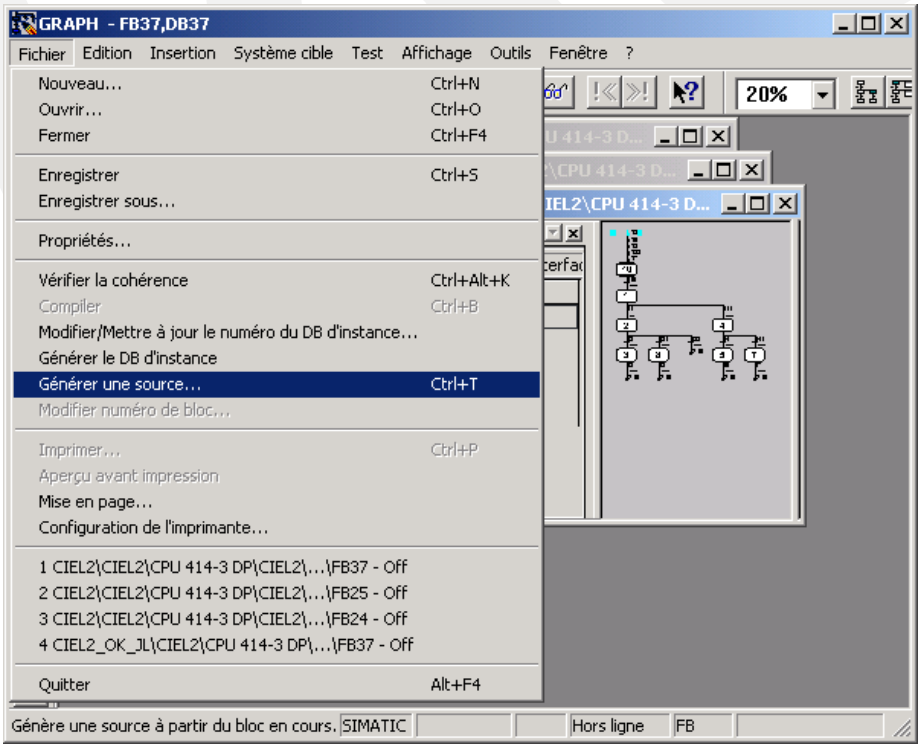


Figure 5.14: Générer le source du bloc GRAPH

- Donnez un nom au fichier source GR7, et pressez **OK**.

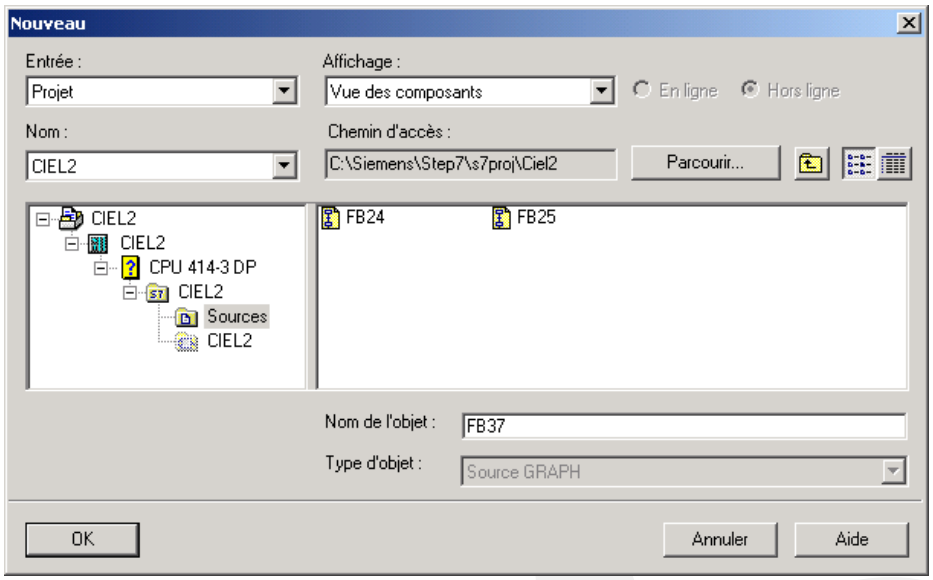


Figure 5.15: Donnez le nom du source à générer

5.3.4 Génération sources SCL pour codes CFC

- Si l'application contient des codes CFC, ils doivent être compiler pour générer les sources SCL correspondantes. Ouvrir l'éditeur CFC

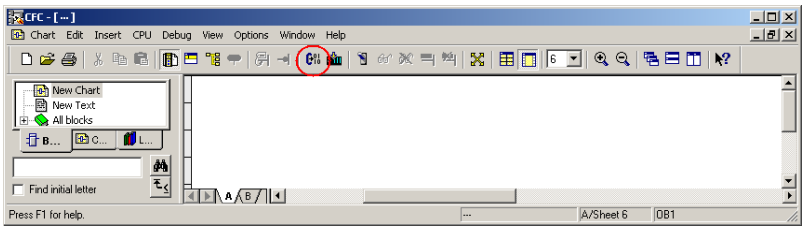


Figure 5.16: Compiler les codes CFC

- Compiler l'ensemble des codes CFC avec les options **Programme complet** et **Générer sources SCL**

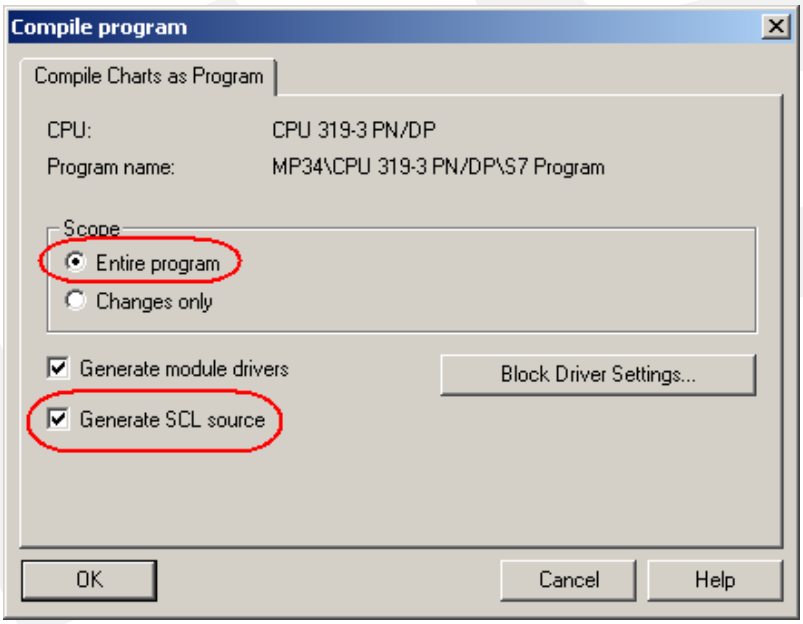


Figure 5.17: Compiler avec génération des sources SCL

- Les fichiers source en SCL ainsi générés peuvent alors être exportés comme les autres fichiers sources. Voir export_des_fichiers_awl_gr7_et_scl.

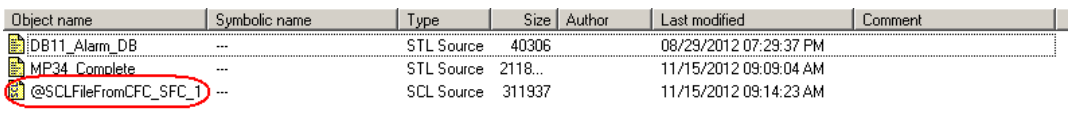


Figure 5.18: Clic droit

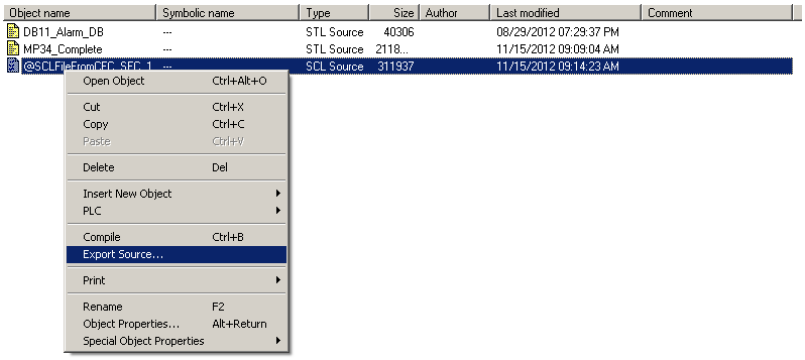


Figure 5.19: Exporter les fichiers sources

5.3.5 Cas d'une application Safety

Attention ! Seule la version Simatic Step7 5.5 supporte l'export de la partie Safety d'une application. Il faudra également que l'outil Step7Safety soit installé.

Une application safety est composée de deux parties :

- une partie standard
- une partie safety

L'export de la partie standard se fait de la manière standard. Veuillez ne pas cocher la case "FC F, DB F, FB F".

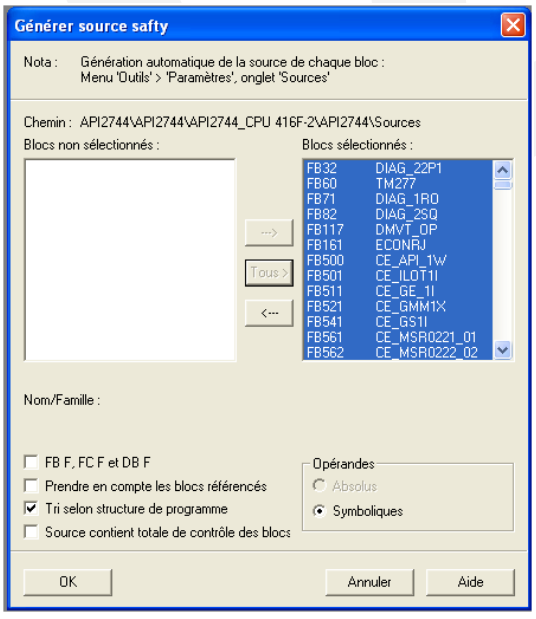


Figure 5.20:

En revanche, l'export de la partie safety nécessite d'autres étapes :

- l'export de la partie safety qui génère un AWL correspondant aux modules et DB safety

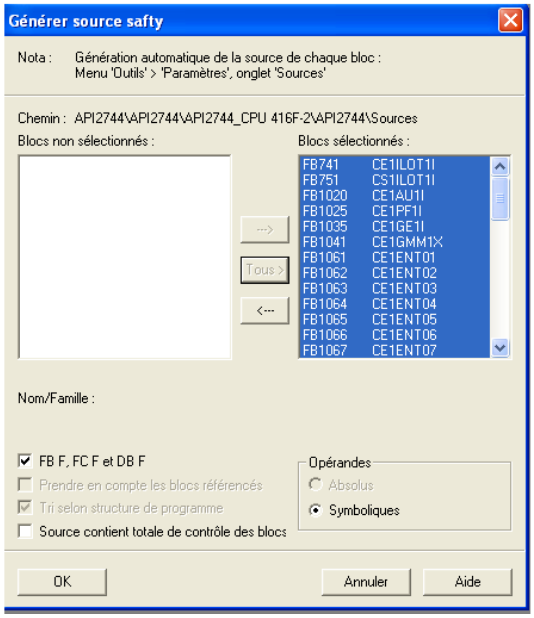


Figure 5.21:

- l'export de la structure du programme qui génère un fichier *.dif (fichier texte)

C'est à partir de ce fichier que l'on peut déterminer si un module est ou non safety.

Export du fichier .dif

Sous l'atelier Step7 faire **Outils** → **Données de référence** → **Afficher**.

Dans la fenêtre "Paramètre", cocher "Structure du programme" puis OK.

La fenêtre contenant la structure du programme s'affiche.

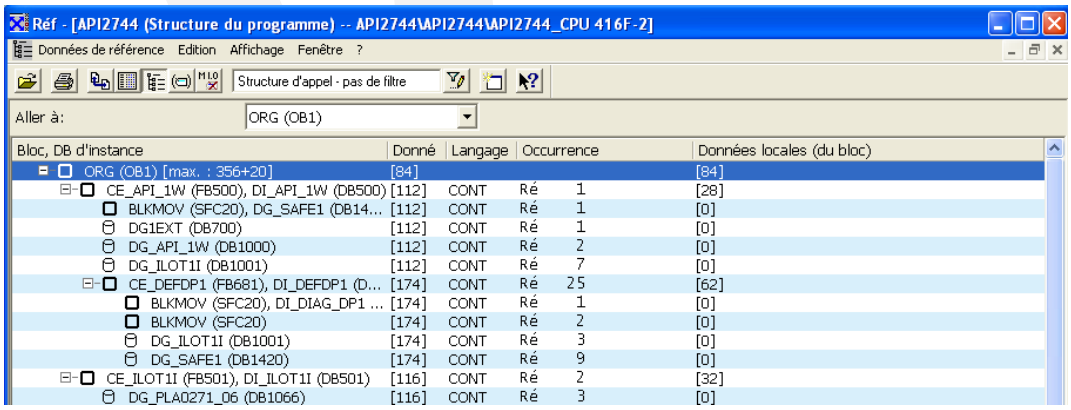


Figure 5.22:

Sélectionner : **Affichage** → **Exportation**.

Donner un nom, par exemple "Structure", puis enregistrer. Le fichier Structure.dif est généré.

5.3.6 Export des fichiers .AWL, .GR7 et .SCL

- Sélectionnez les sources LIST et GRAPH que vous venez de générer, mais aussi vos sources SCL. A l'aide du menu contextuel **Exporter source…** demandez la copie des sources.

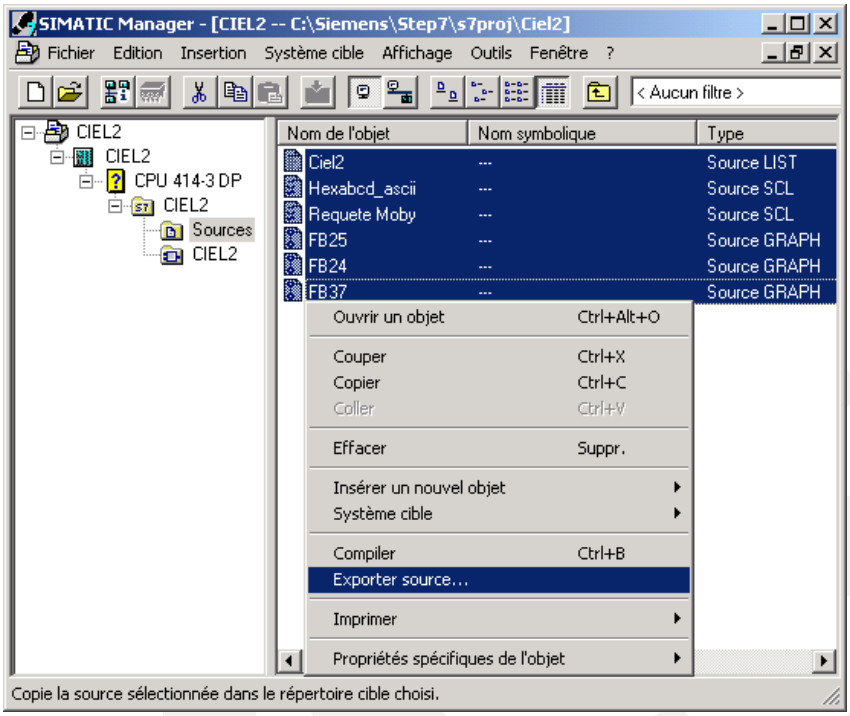


Figure 5.23: Sélectionner les sources à exporter

- Définissez le répertoire où vous voulez exporter vos fichiers.

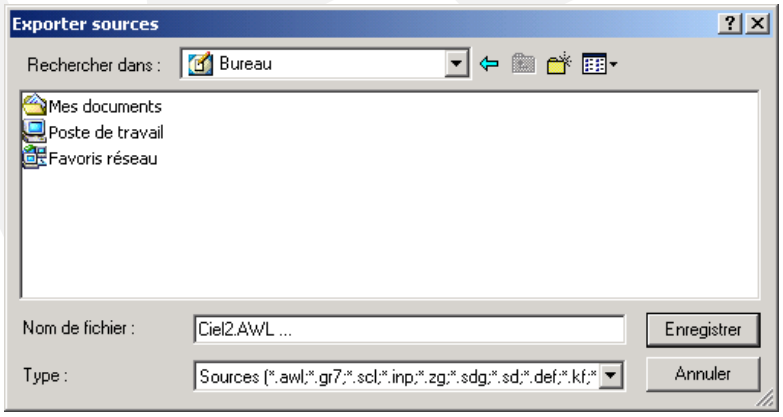


Figure 5.24: Sélectionner le répertoire destination

5.3.7 Import des Fichiers

Avant de pouvoir analyser le programme STEP 7 avec PLC Checker vous devrez importer (copier/coller) dans l'espace de travail, les fichiers **.ASC**, **.AWL**, **.GR7** et **.SCL**. Cet espace de travail peut être la plateforme Eclipse lorsqu'elle est installé sur le poste client ou bien un site web hébergé sur le serveur de PLC Checker (ex: www.automationsquare.com).

5.4 Création d'un fichier XEF Unity

Pour analyser une application Unity avec PLC Checker, vous devez exporter cette application dans un fichier XEF.

Vous pouvez créer un fichier XEF à l'aide du menu **Fichier** > **Exporter l'application...** dans la barre de menus du logiciel Unity.

1. Dans la barre de menus, sélectionnez **Fichier** > **Exporter l'application...**
2. Dans l'assistant **Exporter une application**, donnez un nom au fichier XEF
3. Ne changez pas le répertoire où le fichier XEF sera créé ; cliquez sur "Enregistrer"

Avant de pouvoir l'analyser avec PLC Checker, vous devrez l'importer dans l'espace de travail de la plateforme Eclipse.

5.5 Récupération du fichier SMC pour OMRON Sysmac Studio

Aucune opération d'export n'est requise pour OMRON PLC Checker. PLC Checker supporte nativement le format d'application Sysmac Studio depuis la version v1.0.3.

Le répertoire exact de vos applications Sysmac Studio peut dépendre de votre version de Microsoft Windows et des choix faits pendant l'installation. Vous le trouverez certainement dans le dossier nommé "Mes Documents". Cherchez un fichier d'extension "SMC" et portant le nom du projet.

5.6 Création d'un fichier d'export Codesys v3.x

Pour analyser une application Codesys v3.x avec les outils d'Itris Automation Square (PLC Checker, PLCDocgen, ...), vous devez exporter cette application dans un fichier XML au format *.export.

Attention certains ateliers basés sur Codesys v3.x ne dispose que d'un export au format PLCOpen XML (*.xml) et pas au format Codesys XML (*.export).

Vous pouvez créer un fichier d'export à l'aide du menu **PROJET** > **Exporter...** dans la barre de menus du logiciel Codesys v3.x.

1. Dans la barre de menus, sélectionnez **Projet** > **Exporter...**

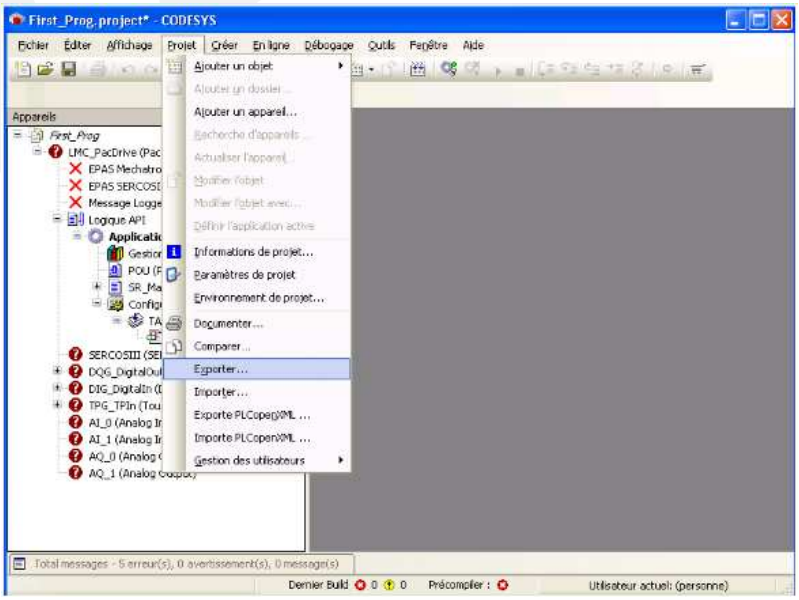


Figure 5.25: Export du fichier XML

2. Choisissez la version d'export suivant votre besoin

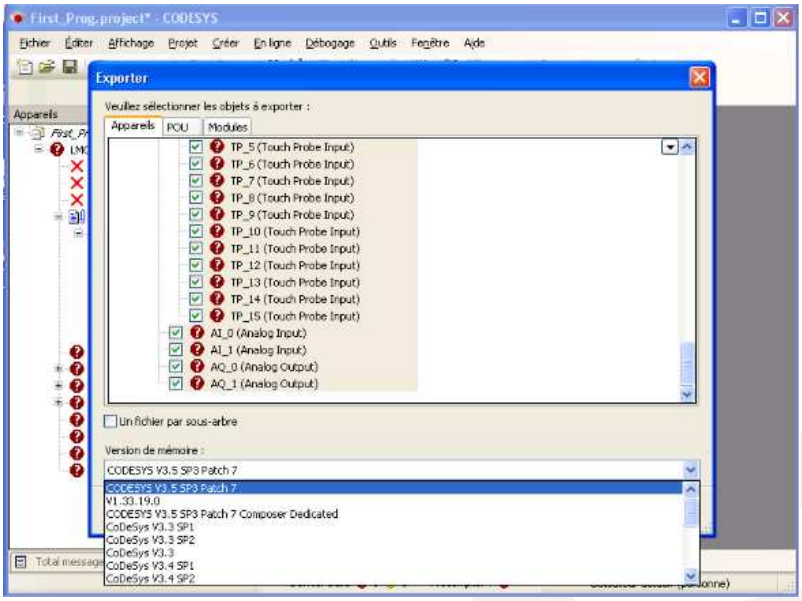


Figure 5.26: Choix de la version de stockage

3. Dans l'assistant **Exporter**, vous allez trouver le type du fichier (*.export)

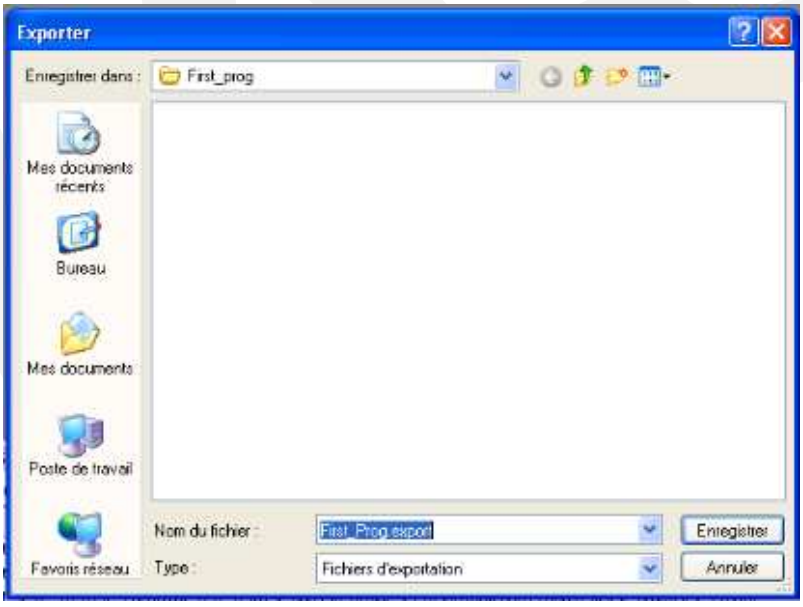


Figure 5.27: Extension actuelle

4. Enregistrez le fichier en lui donnant un nom et en gardant la même extension

Lors de l'analyse, il faut alors choisir le type d'automate 3s - Codesys v3.x.

5.7 Création d'un fichier EXP Codesys V2

Pour analyser une application Codesys V2 avec PLC Checker, vous devez exporter cette application dans un fichier EXP.

Vous pouvez créer un fichier EXP à l'aide du menu **Projet** > **Exporter...** dans la barre de menus du logiciel Codesys V2.

1. Dans la barre de menus, sélectionnez **Projet** > **Exporter...**

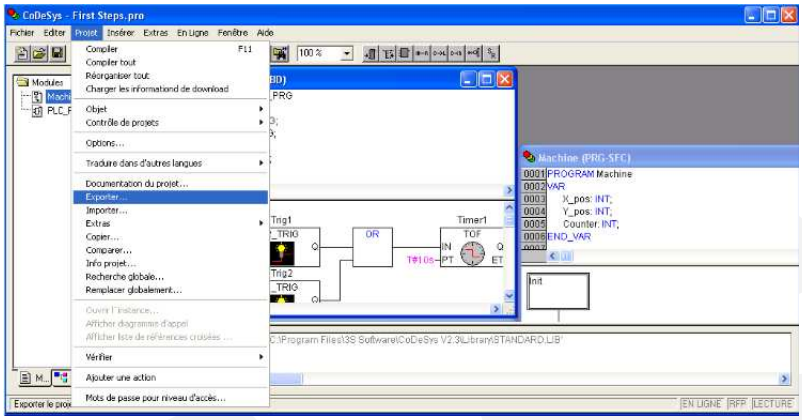


Figure 5.28: Export du fichier EXP

2. Dans l'assistant **Exporter**, donnez un nom au fichier
3. Cliquez sur "Enregistrer"

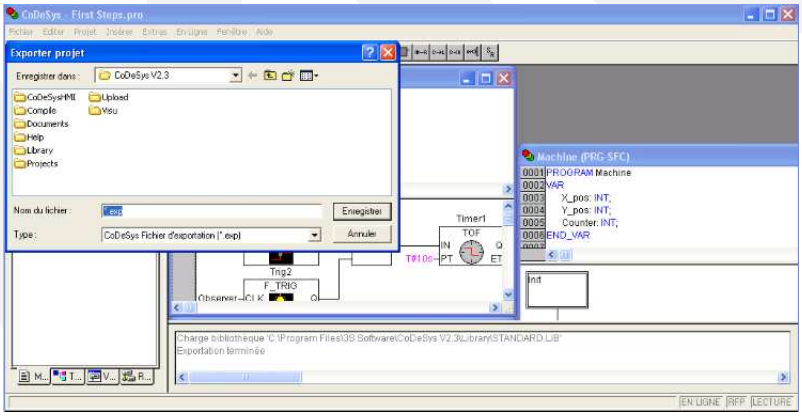


Figure 5.29: Sauvegarde du fichier EXP

Avant de pouvoir l'analyser avec PLC Checker, vous devrez l'importer dans l'espace de travail de la plateforme Eclipse.

5.8 Création d'un fichier EXPORT SoMachine

Pour analyser une application SoMachine avec PLC Checker, vous devez exporter cette application dans un fichier .EXPORT (format xml).

Vous pouvez créer un fichier EXPORT à l'aide du menu **PROJET** > **Exporter...** dans la barre de menus du logiciel SoMachine.

1. Dans la barre de menus, sélectionnez **Projet** > **Exporter…**

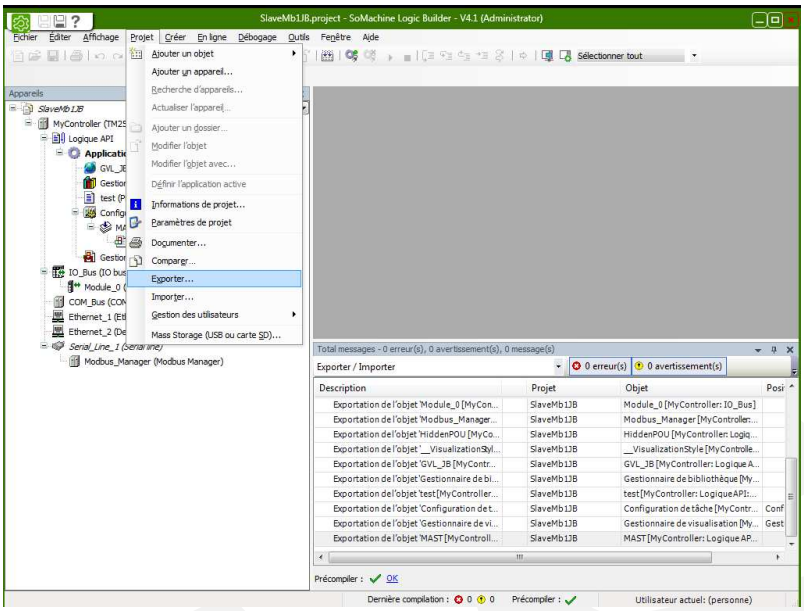


Figure 5.30: Export du fichier XML

2. Choisissez la version d'export suivant votre besoin

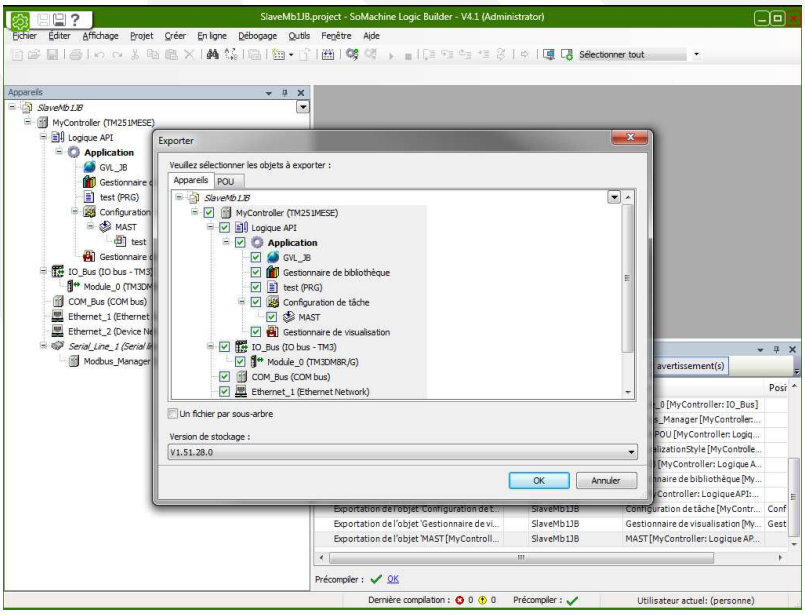


Figure 5.31: Choix de la version de stockage

3. Dans l'assistant **Exporter**, Donnez un nom au fichier (*.export)

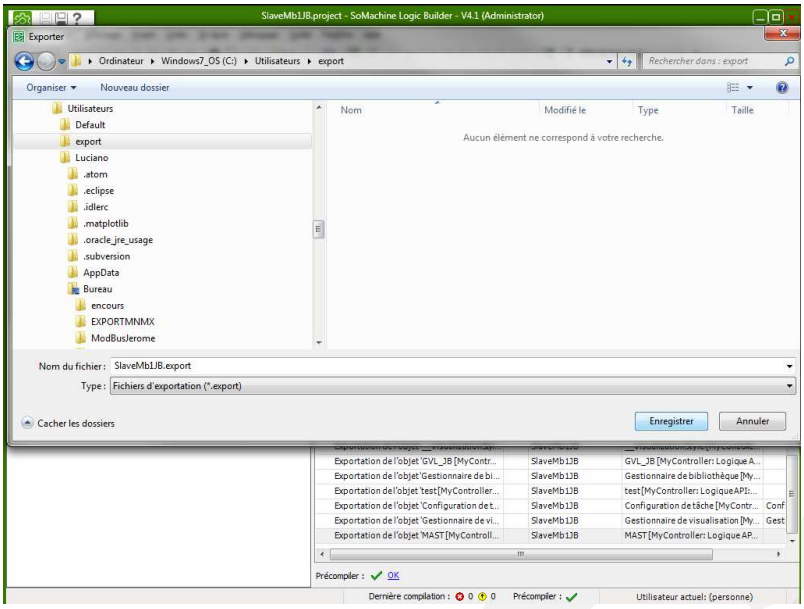


Figure 5.32: Extension actuelle du fichier

4. Enregistrer le fichier (*.**EXPORT**)

Avant de pouvoir l'analyser avec PLC Checker, vous devrez l'importer dans l'espace de travail de la plateforme Eclipse.

5.9 Création d'un fichier XML Beckhoff Twincat

Pour analyser une application Beckhoff Twincat avec PLC Checker, vous devez exporter cette application dans un fichier XML.

Vous pouvez créer un fichier XML à l'aide du **Explorateur de solution** > **Export PLCopenXML…**.

1. Dans la fenêtre explorateur de solution , sélectionnez le projet trouvé dans l'arborescence "PLC"

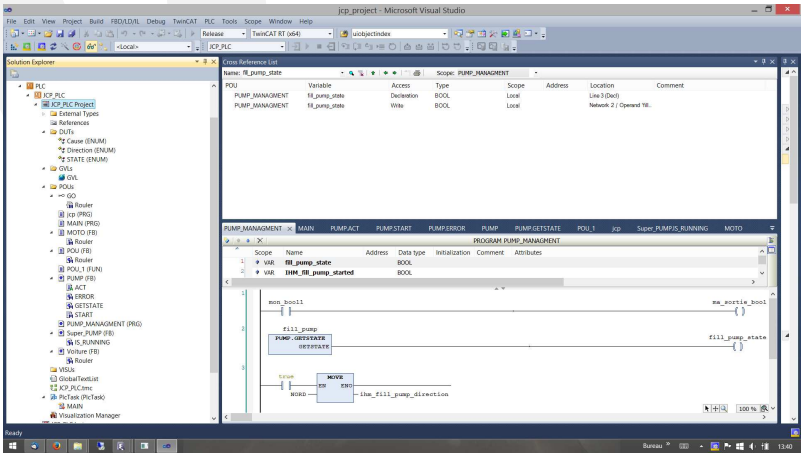


Figure 5.33: Sélection du projet

2. Clic droit de la souris et choisissez **Export PLCopenXml…**

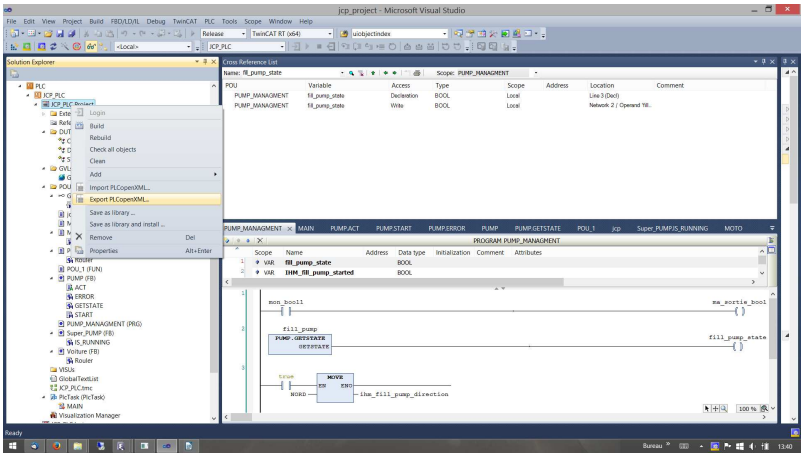


Figure 5.34: Export fu fichier XML

3. Cliquez sur "Enregistrer"

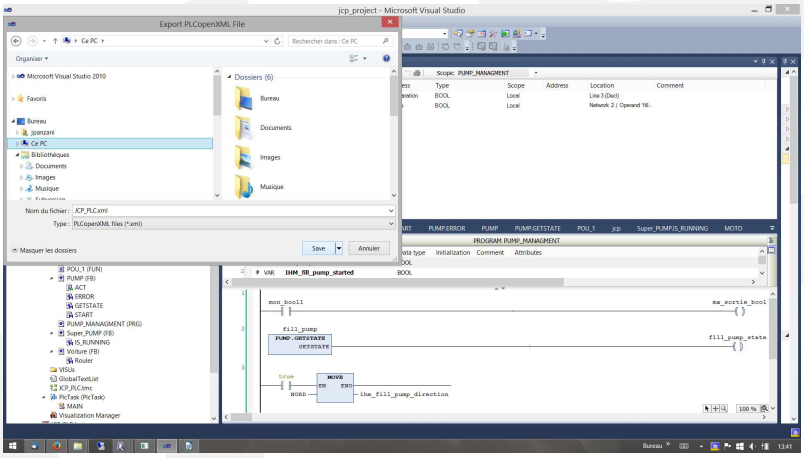


Figure 5.35: Sauvegarde du fichier XML

Avant de pouvoir l'analyser avec PLC Checker, vous devrez l'importer dans l'espace de travail de la plateforme Eclipse.

5.10 Création d'un fichier EXP Beckhoff twincat V2

Pour analyser une application Beckhoff twincat V2 avec PLC Checker, vous devez exporter cette application dans un fichier EXP. Vous pouvez créer un fichier EXP à l'aide du menu **Projet** > **Exporter…** dans la barre de menus du logiciel TwinCAT PLC Control.

1. Dans la barre de menus, sélectionnez **Projet** > **Exporter…**

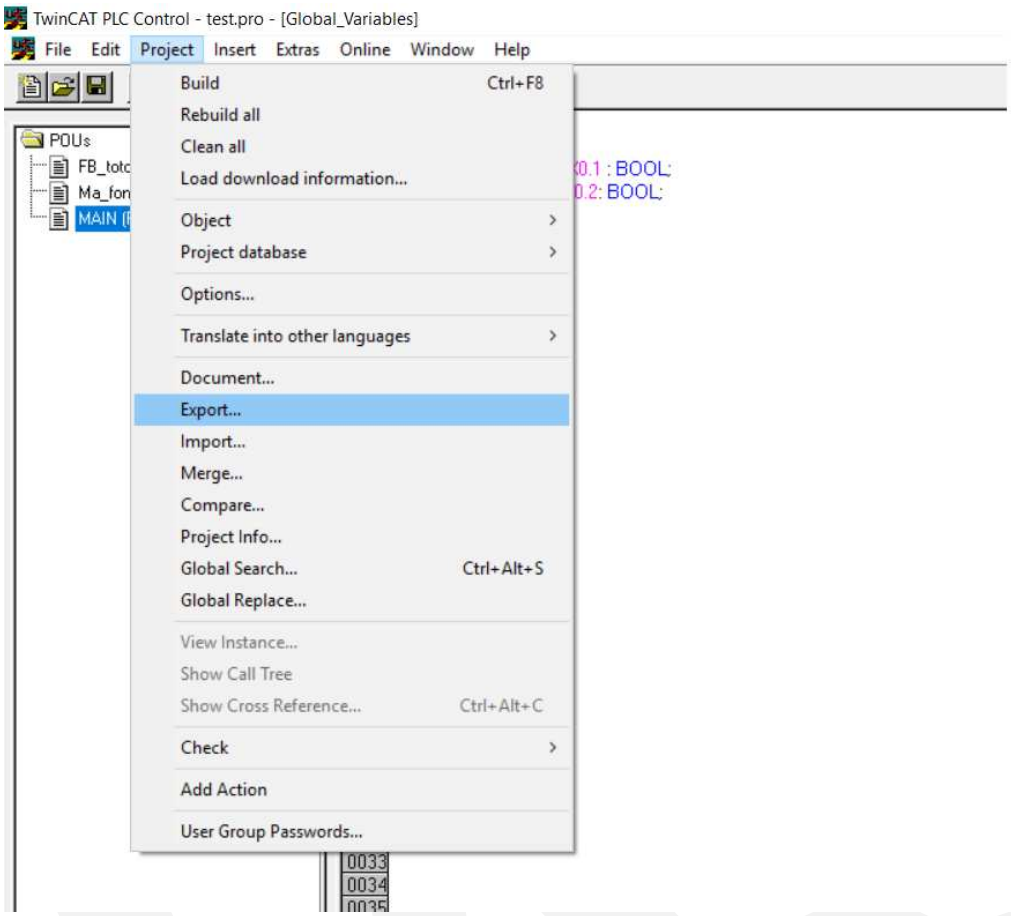


Figure 5.36:

1. Dans l'assistant **Exporter**, donnez un nom au fichier
2. Cliquez sur "Enregistrer"

Avant de pouvoir l'analyser avec PLC Checker, vous devrez l'importer dans l'espace de travail de la plateforme Eclipse.

5.11 Création d'un fichier PC5 RSLogix 5

Pour analyser une application RSLogix 5 avec PLC Checker, vous devez exporter cette application dans un fichier PC5. Vous pouvez créer un fichier PC5 en ouvrant l'application avec l'atelier RSLogix5 et : **Fichier→Enregistrer sous→Type choisir l'export Fichier Librairie (*.PC5)** .

Les fichiers générés sont :

- PC5 file
- SY5 file
- SY6 file
- FIX file
- FIX2 file

Seuls les fichiers PC5 et SY6 sont utilisés pour l'analyse Checker.

Avant de pouvoir l'analyser avec PLC Checker, vous devrez l'importer dans l'espace de travail de la plateforme Eclipse.

5.12 Export d'un projet Tia Portal

5.12.1 Présentation générale de l'application

Itris TIA Portal Import/Export vous permet d'importer et d'exporter du code vers ou depuis Siemens TIA Portal. Cet outil exploite les possibilités de l'outil Openness, fourni par Siemens pour effectuer ce genre d'opérations. Cet outil a été conçu de façon modulaire pour s'adapter au mieux à vos besoins : après avoir installé le système de base, vous installez uniquement les modules dont vous avez besoin.

5.12.2 Pré-requis pour l'installation

Itris TIA Portal Import/Export requière l'installation des composants suivants pour fonctionner :

- Microsoft .NET Framework en version 4.6.1 au minimum (Téléchargement)
- Openness, que vous trouverez :
  - Pour les versions 13 SP1, 13 SP2, 14 et 14 SP1 de TIA Portal : dans le dossier "Support" du disque d'installation de STEP 7,
  - Pour la version 15 de TIA Portal : lors du processus d'installation, dans la liste des modules optionnels
- Pour les version 13 SP1, 13 SP2 et 14 de TIA Portal uniquement : Fichiers d'autorisation fournis par Siemens

5.12.3 Unité de base

L'unité de base est le programme principal de Itris TIA Portal Import/Export, il vous permet d'effectuer simplement les opérations d'import et d'export de code. Pour pouvoir importer et exporter du code, vous devez compléter cette unité de base avec les modules d'import ou d'export dont vous avez besoin.



Figure 5.37:

Notez que l'installation des fichiers d'autorisation vous sera proposée s'ils ne sont pas déjà installés pour les versions 13 SP1, 13 SP2 et 14 de TIA Portal (si vous ne savez pas de quoi il s'agit, reportez-vous à la section présentant l'installation de Openness).

5.12.4 Modules

Les modules viennent se greffer à l'unité de base pour lui fournir les fonctions d'import et d'export de code dont vous avez besoin. Ils doivent être téléchargés et installés à part. Aucun pré-requis additionnel à ceux de l'unité de base n'est à prendre en compte.

Notez que pour être détectés, les modules doivent être installés dans le même dossier que l'unité de base ou dans un sous-dossier (il s'agit du cas de figure par défaut).

Une fois les modules installés, l'unité de base les détecte et vous permet de les exploiter. Vous pouvez avoir plusieurs modules pour plusieurs versions de TIA Portal, pour autant que la version d'Openness qui est compatible avec TIA Portal soit installée (si ce n'est pas le cas, l'unité de base émettra un message pour vous prévenir).

5.12.5 Fonctionnement

Une fois l'unité de base et les modules installés, vous pouvez procéder aux opérations de votre choix via l'interface graphique de l'unité de base (une icône a due être déposée sur votre bureau suite à l'installation de l'unité de base).

Notez que durant chacune de ces opérations, TIA Portal peut vous demander l'autorisation d'être connecté aux modules via la fenêtre suivante :

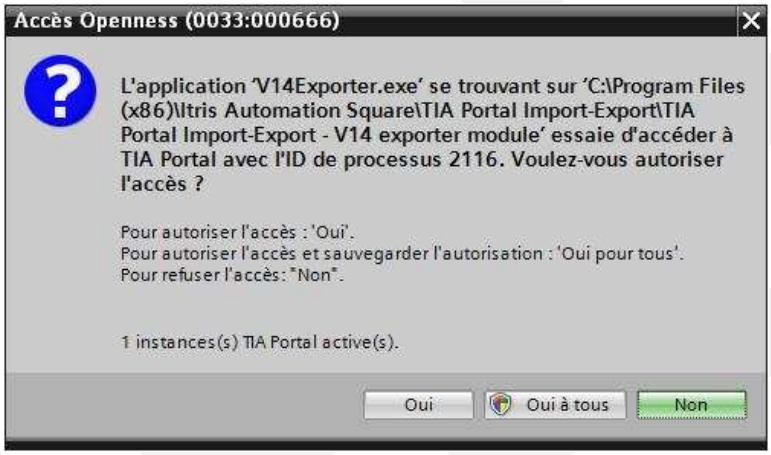


Figure 5.38:

Vous pouvez autoriser les modules à être acceptés de façon systématique ou ponctuelle.

Si vous souhaitez effectuer un export de code à partir de TIA Portal et pour un outil d'ITRIS Automation (PLC Checker, PLC DocGen, ...), vous devez localiser votre projet TIA Portal via la section "Export project from TIA Portal", puis cliquer sur "Proceed".

Une fois le processus terminé, l'interface vous indiquera où trouver l'export de code à fournir à l'outil d'ITRIS AUTOMATION ainsi que le pourcentage de code qui a pu être exporté de votre application.

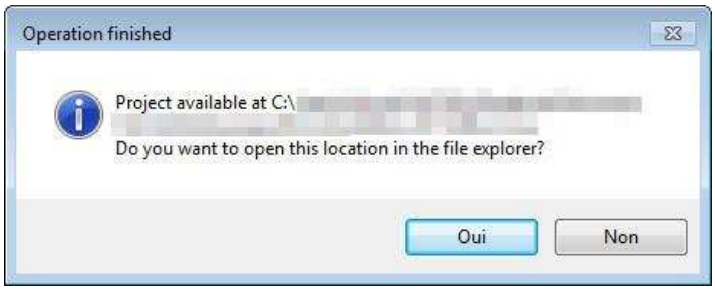


Figure 5.39:

INFO	**Il n'est pas possible d'exporter du code ne compilant pas ou étant inconsistant. Pour éviter de générer des erreurs d'export et vous assurer que les outils d'ITRIS recevront tout votre code, vous devez le compiler vous-même via TIA Portal et sauvegarder le résultat avant de procéder à l'export du code. Certaines versions de Openness ne permettent également pas l'export de code protégé, dans ce cas, le pourcentage de code exporté sera forcément inférieur à 100%.**

INFO	**Dans le cas où vous souhaitez utiliser les outils d'import/export avec les version 13 SP1, 13 SP2 ou 14 SP1 de TIA Portal, vous devez vous assurer que le format de votre projet est bien V13 SP1, V13 SP1 ou V14 SP1 et non V13 ou V14. Plus d'informations à ce propos sont données dans le chapitre *Format du projet TIA Portal.***

Si vous souhaitez effectuer un import de code depuis un outil d'ITRIS Automation (PLC Converter, ...), vous devez localiser l'export de code au format ZIP, qui vous a été fourni par PLC Converter, via la section "Import project from TIA Portal" puis sélectionner le projet TIA Portal dans lequel vous souhaitez importer le code contenu dans le fichier ZIP. Enfin cliquez sur "Proceed". Une fois le processus terminé, l'interface vous indiquera le pourcentage de réussite d'import du code.

INFO	**Toute unité de code présente à la fois dans l'export de code au format ZIP et dans le projet que vous fournissez en tant que cible d'importation sera écrasée dans le projet cible par l'unité contenue dans le fichier ZIP.**

Utilisation avancée : Vous pouvez utiliser l'unité de base en mode invité de commande via le programme "ItrisTIAExporterCli". Vous pouvez en apprendre plus sur son fonctionnement avec le paramètre "?".

5.12.6 Installation de Openness

Présentation

Openness est la passerelle logicielle intervenant pour permettre le dialogue entre Itris TIA Portal Import/Export et TIA Portal. Sa présence est requise pour faire fonctionner Itris TIA Portal Import/Export.

Installation et préparation

Vous trouverez Openness avec le DVD d'installation de STEP 7, dans le dossier support pour les version 13 SP1, 13 SP2, 14 et 14 SP1 de TIA Portal, ou lors du processus d'installation de TIA Portal pour la version 15.

Veillez à installer la version de Openness qui correspond à la version de TIA Portal que vous utilisez.

INFO	**Il vous faudra peut-être mettre à jour Openness à l'aide de l'utilitaire de mise à jour de TIA Portal. Lisez attentivement les instructions fournies lors de l'installation, ce détail sera explicité en fin de processus d'installation.**

Une fois Openness installé, vous devez permettre au compte utilisateur de l'ordinateur qui va effectuer les exports et imports de code de pouvoir l'utiliser. Pour cela :

- Effectuez un clic droit sur Ordinateur (ou Ce PC suivant la version de Windows que vous utilisez)
- Sélectionnez Gérer



Figure 5.40:

- Dans la fenêtre qui vient de s'ouvrir, dépliez les éléments suivants :
  - Outils système
  - Utilisateurs et groupes locaux



Figure 5.41:

- Placez-vous sur le dossier Groupes, vous devez voir apparaître le groupe Siemens TIA Openness, double-cliquez dessus pour l'éditer
- Cliquez sur Ajouter...
- Tapez le nom de l'utilisateur de l'ordinateur et cliquez sur Vérifier les noms, pour vous indiquer que Windows a réussit à trouver le nom, ce dernier apparaîtra souligné
- Cliquez pour finir sur OK puis Appliquer et redémarrez votre ordinateur. Openness est prêt à être utilisé par l'utilisateur de l'ordinateur. Ces opérations sont à répéter pour chaque utilisateur susceptible d'utiliser Openness sur l'ordinateur.

Fichiers d'autorisation

Les versions 13 SP1, 13 SP2 et 14 de TIA Portal requièrent une autorisation de la part de Siemens pour être utilisées avec Openness. Vous devez, pour utiliser Openness, suivre la procédure officielle de Siemens, à l'adresse suivante : TIA Portal Openness Guidance Usagefile EnablerFile.

Une fois que vous disposez des fichiers, procédez à un export de code avec Itris TIA Portal Import/Export. L'assistant détectera l'absence des autorisations et vous proposera de les installer pour vous, cette opération ne sera effectuée qu'une seule fois pour tous les utilisateurs de l'ordinateur :

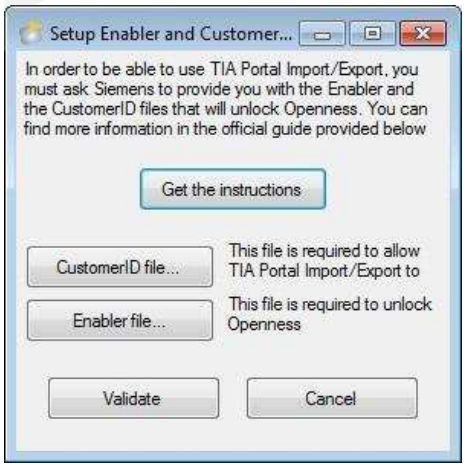


Figure 5.42:

5.12.7 Format du projet TIA Portal

Une erreur courante remontée par l'outil d'import/export est "*Your project is not saved in the TIA Portal V×× SP×× format...*". Ce message signifie que le projet est enregistré dans le format d'une version antérieure de TIA Portal. Si TIA Portal sait s'en accommoder, ce n'est pas le cas de Openness. Il vous faut donc mettre le format de votre projet à niveau pour qu'il soit dans la même version que TIA Portal.

Pour cela, ouvrez-le dans TIA Portal et sélectionnez "Projet" / "Mise à niveau" :

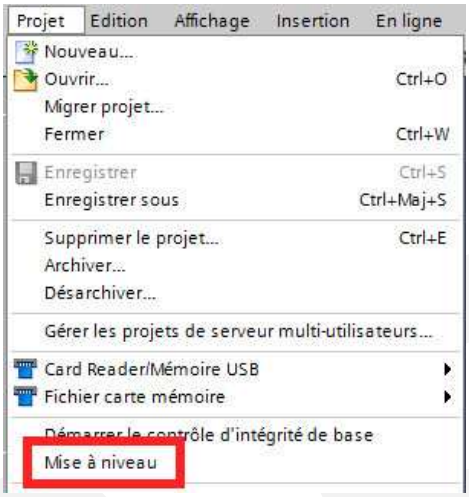


Figure 5.43:

Un message d'information vous indiquera que cette action est irréversible. Cela signifie qu'après avoir fait cela, la version antérieure de TIA Portal dans laquelle la version du projet se trouvait sera incapable d'ouvrir le projet. Vous devez valider ce message, recompiler et sauvegarder votre application pour être dans la mesure d'utiliser l'outil d'import/export.

5.13 Création d'un fichier XML Phoenix Contact Multiprog

Pour analyser une application Phoenix Contact Multiprog avec PLC Checker, vous devez exporter cette application dans un fichier XML.

Vous pouvez créer un fichier XML à l'aide du menu **Fichier** > **Exporter...** dans la barre de menus du logiciel Phoenix Contact Multiprog.

1. Dans la barre de menus, sélectionnez **Fichier** > **Exporter...**

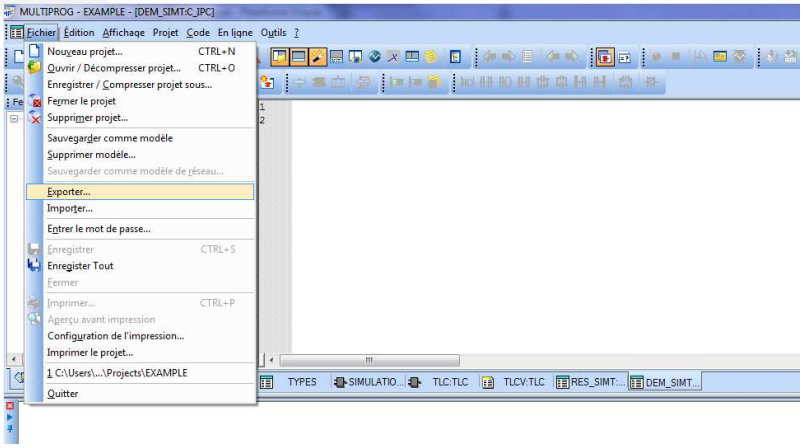


Figure 5.44: Export du fichier

2. Choisissez "Exporter fichier PLCopen XML"

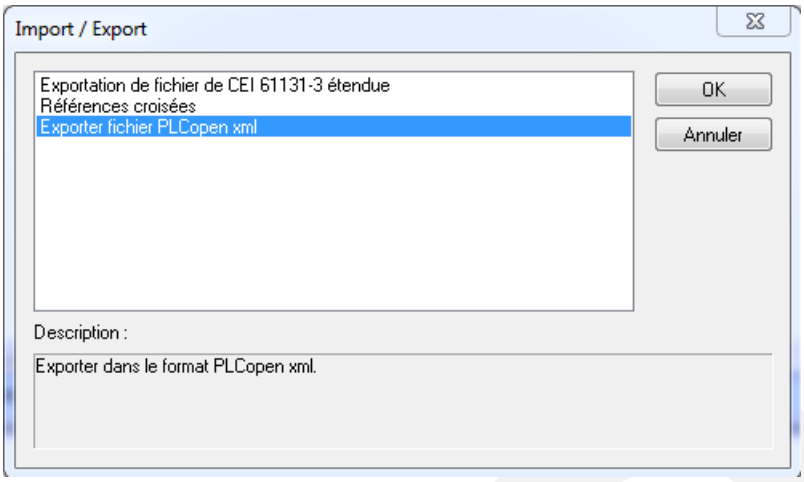


Figure 5.45: Choix du type d'export

3. Donnez un nom au fichier et enregistrez (*.**XML**)

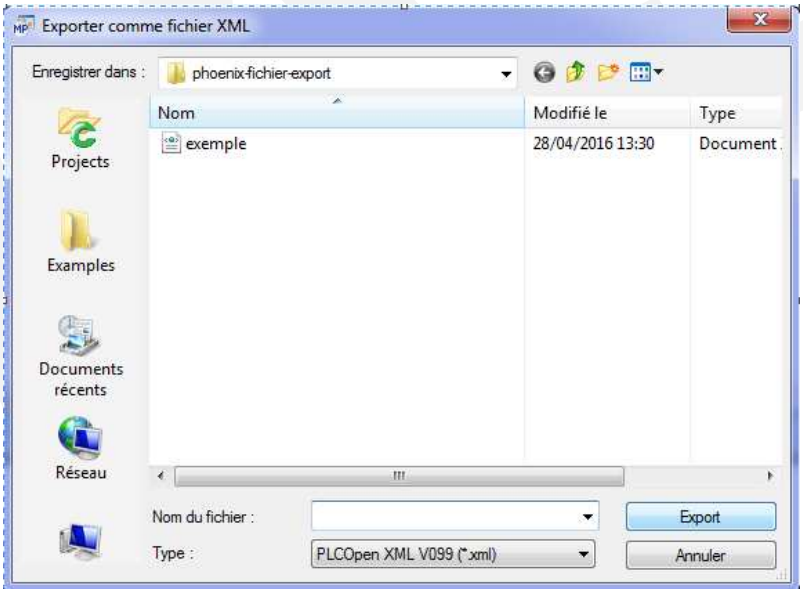


Figure 5.46: Sauvegarde du fichier

Avant de pouvoir l'analyser avec PLC Checker, vous devrez l'importer dans l'espace de travail de la plateforme Eclipse.

Note : Les résultats issus de l'analyse avec PLC Checker diffèrent suivant la version du fichier PLCopen XML choisie !

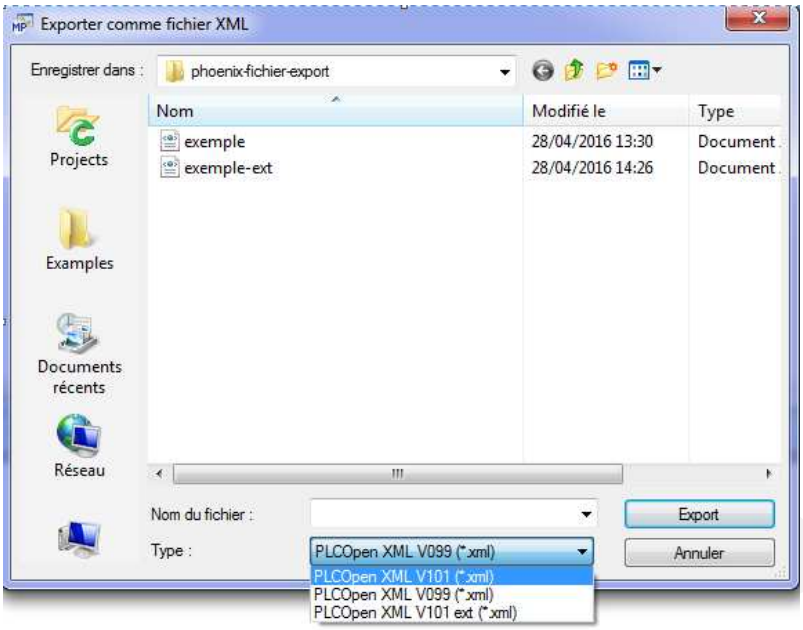


Figure 5.47: Choix de la version

Chapter 6

Itris Import/Export Tool

6.1 Présentation

L'outil de Itris Automation d'import/export de code vous permet de réaliser simplement l'export et l'import de vos codes automates, dans le but de les utiliser dans les outils d'Itris (PLC Checker, PLC Docgen, ICS Monitoring, PLC Converter, . . .).

Cet outil vient s'interfacer avec les ateliers automates avec lesquels vous programmez vos automates. Il est donc nécessaire de l'installer sur les ordinateurs avec lesquels vous programmez.

Dans un cadre plus avancé, il vous est également possible d'interfacer **Itris Import/Export Tool** avec des outils tiers ou avec votre système d'intégration continue puisqu'il offre également une interface en ligne de commandes.

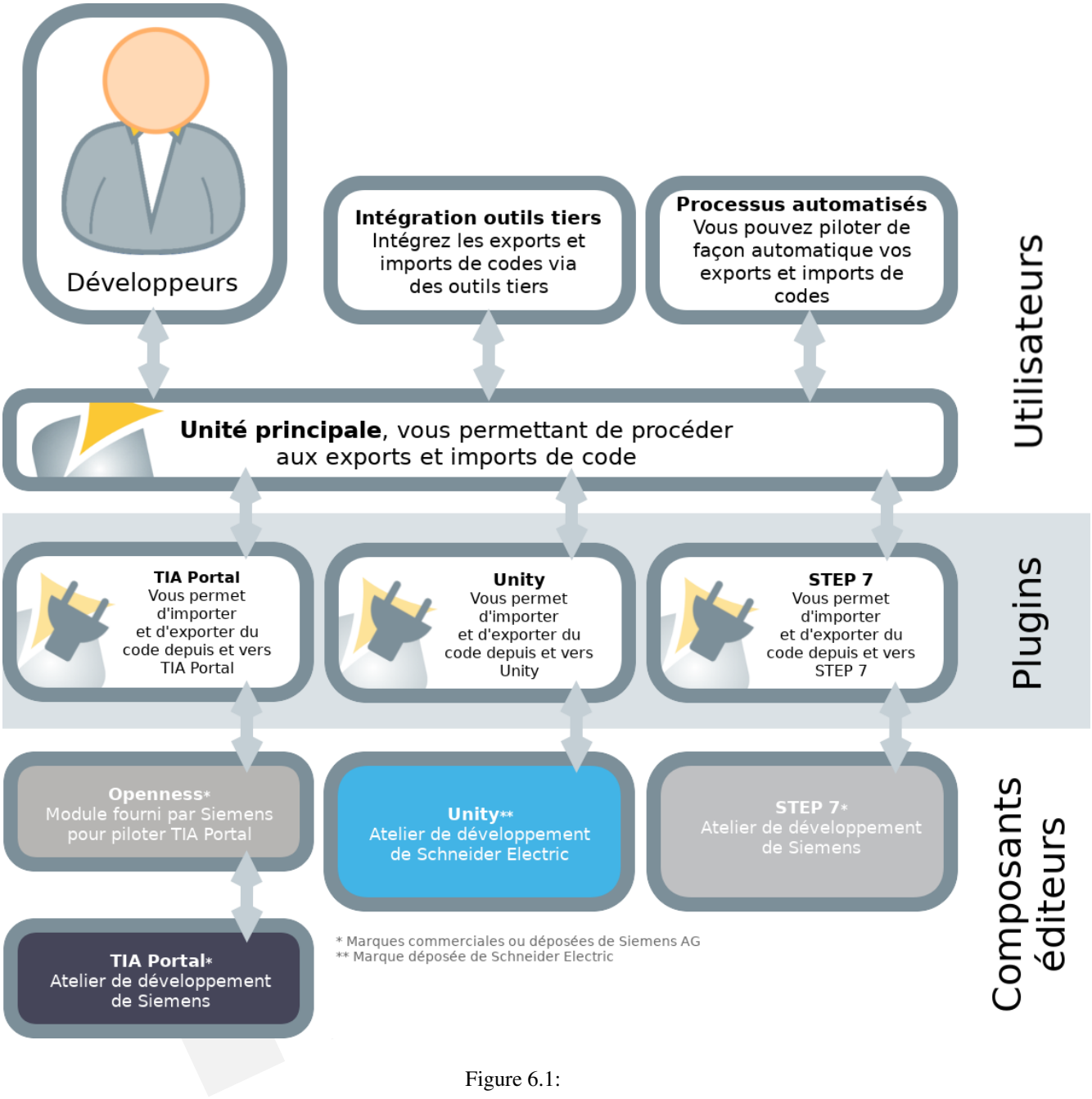


Figure 6.1:

6.2 Installation

Itris Import/Export Tool fonctionne de façon modulaire. Vous allez devoir commencer par installer l'**unité principale**, qui vous fournira les interfaces utilisateurs dont vous allez avoir besoin. Ensuite, vous devrez y adjoindre les plugins correspondant aux ateliers logiciels que vous utilisez. La majorité des produits proposés par Itris Automation ne requière que l'installation des plugins d'export, cependant, PLC Converter requerra de votre part que vous installiez en plus un plugin d'import de code.

Afin de déterminer aisément quels sont les éléments dont vous allez avoir besoin, Itris Automation propose un sélecteur en ligne disponible à l'adresse suivante : http://www.itris-automation.com/itris-importexport-tool/ .

Sélectionnez les ateliers (Unity, TIA portal, . . .) et outils (PLC Checker, PLC Converter, . . .) que vous allez utiliser avec l'outil, et le sélecteur vous indiquera les plugins que vous allez devoir télécharger et installer.

Le processus d'installation ne requière pas de formalité particulière. Par défaut l'unité principale est installée dans

le dossier "Program Files" et les plugins sont installés dans des sous-dossiers de l'unité principale. Si vous souhaitez changer ce répertoire d'installation de l'unité principale, vous devrez faire attention à choisir d'installer les plugins dans des sous-dossiers de l'unité principale.

NOTE	**Cet outil remplace l'outil TIA Portal Import/Export Tool, il reste compatible avec les anciennes unités d'import/export, et vous pouvez utiliser l'installeur de Itris Import/Export Tool pour mettre à jour TIA Portal Import/Export Tool, il faudra cependant être vigilent lors de l'installation de nouvelles unités, puisque les chemins d'installation par défaut ont changé.**

6.3 Prise en main

Après l'installation de l'unité principale et des plugins dont vous avez besoin, vous pouvez commencer à réaliser des exports et/ou imports de codes automates.

NOTE	**Certains ateliers, telles que certaines versions de TIA Portal, requièrent des étapes supplémentaires, veuillez vous reporter aux manuels dédiés aux ateliers pour obtenir davantage d'informations.**

Les deux interfaces proposées par **Itris Import/Export Tool** sont :

- L'interface utilisateur graphique : elle vous permet de réaliser très simplement toutes les opérations d'import et d'export de codes. Il s'agit de l'interface que vous utiliserez si vous n'avez pas besoin d'automatiser les exports et imports de code.
- L'interface en ligne de commande : elle vous permet de réaliser des opérations d'import et d'export de codes sans interaction avec l'utilisateur. Cela facilite l'intégration de ces opérations dans un outil tiers ou dans un système d'intégration continue. Cette interface est cependant plus difficile à utiliser.

6.3.1 Interface graphique pour les opérations du quotidien

Cette interface vous permet de réaliser des opérations d'import et d'export simplement. Elle se présente en trois parties :

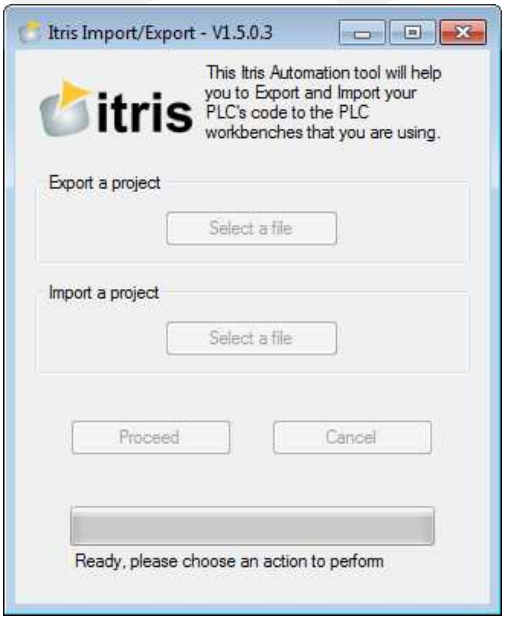


Figure 6.2:

La première partie de l'écran vous permet de demander à réaliser un **export de code**. Pour cela, il vous suffit de cliquer sur le bouton "Select a file", dans la zone "Export a project", puis de sélectionner un fichier de projet supporté par les plugins que vous avez installés.

Pour lancer l'opération d'export à proprement parler, cliquez sur "Proceed".

NOTE	**Tous les ateliers ne réagissent pas de la même façon à la réalisation d'une opération, veuillez vous reporter aux manuels dédiés aux ateliers pour obtenir davantage d'informations.**

Une fois l'export terminé, **Itris Import/Export Tool** vous indiquera où trouver le fichier exporté (il s'agit d'une archive au format ZIP), et vous proposera d'ouvrir la localisation du fichier dans l'explorateur de fichiers.

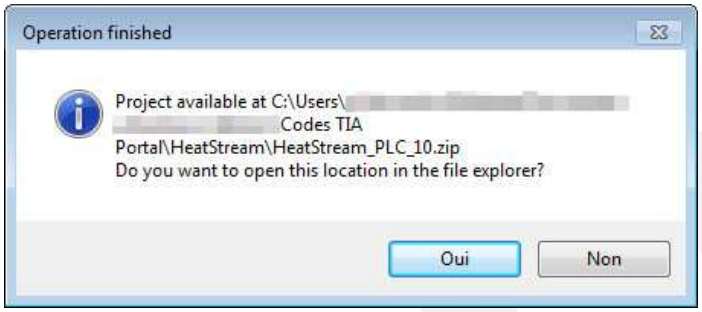


Figure 6.3:

La seconde partie de l'écran permet de réaliser un **import de code**. Pour cela, il vous suffit de cliquer sur le bouton "Select a file", dans la zone "Import a project", puis de sélectionner un fichier au format ZIP contenant un export de code. Une fois le fichier validé, il vous faudra sélectionner un fichier de projet supporté par les plugins que vous avez installés. Le code sera alors importé dans le projet que vous aurez choisi.

NOTE	**Dans l'éventualité où un bloc de code existerait à la fois dans le code à importer et dans le projet que vous avez sélectionné, le bloc contenu dans le projet sera effacé**

Pour lancer l'opération d'import à proprement parler, cliquez sur "Proceed".

Enfin, la troisième partie est la partie de validation/report d'information. Elle contient le bouton "Proceed", qui permet de valider une opération d'import ou d'export de code. Il est également possible d'annuler une demande d'import ou d'export. Quand le processus est lancé, cette partie de la fenêtre vous permet de suivre l'évolution du processus.

6.3.2 Interface en ligne de commandes pour les opérations automatisées

Cette interface est réservée aux utilisateurs avancés. Elle permet de lancer des processus sans requérir d'actions de l'utilisateur (ce dernier n'aura pas à sélectionner le projet, ni à cliquer sur "Proceed").

NOTE	**Cette interface ne peut pas mener à une configuration du système, les pré-requis de certains atelier ne peuvent donc pas être satisfaits par cette interface, il vous faudra les mettre en place via l'interface graphique standard, veuillez vous reporter aux manuels dédiés aux ateliers pour obtenir davantage d'informations.**

Pour utiliser cette interface, il vous faut l'exécuter à partir d'un terminal, ou de tout élément capable de faire passer des arguments à un programme (un raccourcis Windows par exemple). L'exécutable à utiliser se trouve à l'endroit où vous avez installé l'unité principale, et se nomme **ImportExportMainUnitCli.exe**.

Par défaut, l'exécutable analyse un à deux arguments que vous lui passez, correspondant à des chemins vers des fichiers.

- S'il y a un seul argument, alors un processus d'export de code va être lancé, et il vous faudra fournir un chemin vers un fichier de projet.

- S'il y a deux arguments, alors un processus d'import de code va être lancé, et il vous faut fournir en premier argument le chemin vers le fichier ZIP d'export, en second argument, fournir le fichier de projet vers lequel sera importé votre code.

Pour obtenir de l'aide à tout moment, il vous suffit d'appeler l'exécutable avec l'argument **-?** (ou sans aucun argument).

Quelques exemples d'utilisation :

- ImportExportMainUnitCli.exe -s project.ap13 : Exporte le projet localisé en **.\project.ap13** (TIA Portal Version 13)
- ImportExportMainUnitCli.exe -s -p=2 project.ap13 : Exporte le programme n°2 du projet localisé en **.\project.ap13** (TIA Portal Version 13)
- ImportExportMainUnitCli.exe -s codeFromConverter.zip project.ap13 : Importe silencieusement le code contenu dans **.\codeFromConverter.zip** dans le projet **.\project.ap13** (TIA Portal Version 13)

Liste des arguments :

- ? : Affichage de l'aide
- p : Si le projet contient plusieurs programmes, doit contenir le numéro du programme à choisir pour effectuer les opérations, vaut par défaut "?", signifiant que l'utilisateur devra interagir pour choisir le programme si plusieurs programmes sont détectés
- d : Synonyme de **p**
- t : Pour les exports de code, permet de choisir l'endroit où écrire le fichier ZIP, n'écrase pas un fichier qui existe déjà
- T : Comme **t**, mais écrase le fichier s'il existe déjà
- s : Affiche moins de messages lors de l'exécution
- v : Affiche plus de messages lors de l'exécution
- i : Affiche des informations sur l'installation (versions des plugins, de l'unité principale…)

6.4 Visualiser le contenu des exports

Il vous est possible de visualiser le contenu d'un export de code au format ZIP en utilisant l'utilitaire **Archive Viewer**, qui est installé en même temps que l'unité principale.

Pour cela, lancez-le, puis fournissez-lui un export de code en cliquant sur les menus "Archive…" puis "Read". Vous pourrez alors visualiser le contenu de l'export, ainsi qu'obtenir des informations sur l'état d'export des blocs de codes, et leur contenu.

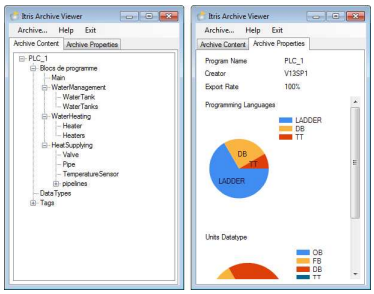


Figure 6.4:

6.5 Visualiser les plugins installés

Les deux interfaces vous permettent de visualiser la liste des plugins installés.

Lorsque vous utilisez l'**interface graphique**, cliquez droit n'importe où dans la fenêtre et sélectionnez "Plugins" :



Figure 6.5:

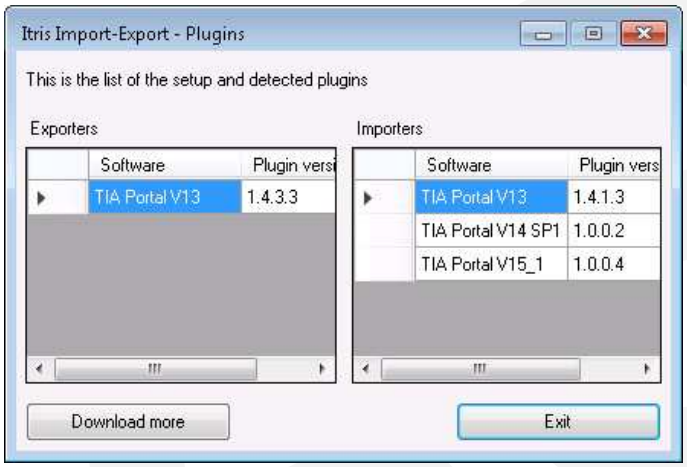


Figure 6.6:

NOTE	Cette fonctionnalité est disponible depuis la version 1.5.3.4 de l'unité principale

Lorsque vous utilisez l'**interface en ligne de commande**, vous devez utiliser l'argument "-i" pour obtenir ces informations.

Chapter 7

Itris Import/Export Tool - STEP 7

7.1 Présentation de Itris Import/Export Tool

L'outil d'import/export de code de Itris Automation vous permet de réaliser simplement l'export et l'import de vos codes automates, dans le but de les utiliser dans PLC Checker, PLC Docgen, ICS Monitoring et PLC Converter.

Pour plus d'informations, notamment relatives à l'installation et à l'utilisation de l'outil, veuillez vous reporter à la documentation spécifique à l'outil **Itris Import/Export Tool**. La présente documentation vient compléter cette dernière dans le cas où vous utiliseriez les plugins dédiés à STEP 7.

7.2 Prérequis spécifiques à STEP 7

Itris Import/Export Tool est capable d'exporter du code automate depuis STEP 7, sans que vous n'ayez d'autres opérations à faire que d'installer l'unité principale, le plugin STEP 7 et STEP 7 lui-même sur le même ordinateur.

7.3 Avertissements spécifiques à STEP 7

7.3.1 Langages de programmation

Tous les codes que vous pouvez réaliser sur STEP 7 ne sont pas susceptibles d'être exportés de la même façon à cause de limitations dans l'interface d'interconnexion avec STEP 7.

Ainsi, tous vos codes en Ladder (Contact), LIST, LOG (Logigramme) et les mnémoniques peuvent être exportés sans intervention de votre part. Si vous n'utilisez que ces langages, le présent paragraphe ne vous concerne pas.
Les codes écrits en **SCL, CFC et Grafcet** ne peuvent en revanche pas être exportés via l'interface d'interconnexion. Le plugin STEP 7 de **Itris Import/Export Tool** peut cependant les inclure dans les exports de code sous réserve que vous ne les génériez avant de procéder à l'opération d'exportation.

Pour cela, pour les blocs en **Grafcet**, il vous faudra à chaque fois que vous procéderez à une opération de génération, qui va créer une copie de votre code :

- Sélectionnez chaque bloc de code à exporter, puis ouvrez-le avec son éditeur
- Cliquez sur "Fichier", "Générer la source..."
- Donnez un nom à la source et validez, vous devrez recommencer cette opération pour chaque bloc

Pour les codes écrits en **CFC**, il vous faudra les exporter au format SCL, de la façon suivante :

- Compiler l'ensemble des sources, en sélectionnant les options suivantes :
  - Programme entier
  - Générer les sources SCL

Pour les codes en SCL, s'ils sont liés à une source valide, vous n'avez pas besoin de faire quoi que ce soit. Sinon, si la source a été supprimée, il vous faudra générer une source en STL grâce à l'éditeur STL.

Toutes ces procédures sont précisées dans le manuel de PLC Checker (voir paragraphe "Création des fichiers .ASC, .AWL, .GR7 et .SCL Step7").

7.3.2 Interface en ligne de commandes

À cause de limitations dans l'interface d'interconnexion avec STEP 7, il se peut que STEP 7 demande à l'utilisateur d'interagir lors d'un processus d'exportation de code. Ceci est problématique lorsque vous utilisez l'interface en ligne de commandes de **Itris Import/Export Tool** pour réaliser des exports sans intervention utilisateur.

7.3.3 Compatibilité avec les anciennes versions de PLC Checker, PLC Docgen et PLC Converter

Étant donné que le système d'import/export est plus récent que certaines versions des produits d'Itris Automation, le format d'export ou d'import du code utilisé par les produits d'Itris Automation et Itris Import/Export Tool sont différents et incompatibles.

Afin d'obtenir des fichiers compatibles avec les produits d'Itris Automation, il vous faudra utiliser l'utilitaire **STEP 7 Export Retro-Compatibility Tool** qui est installé sur votre ordinateur en même temps que le plugin STEP 7.

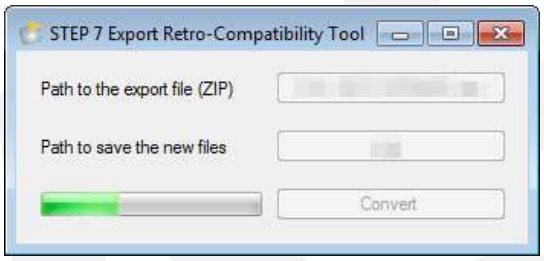


Figure 7.1:

Pour procéder, il vous suffit d'indiquer la localisation de l'export au format ZIP qui a été créé, et un dossier dans lequel écrire les fichiers à utiliser avec PLC Checker, PLC Docgen et PLC Converter, puis de cliquer sur "Convert".

A la fin de l'opération, vous obtiendrez plusieurs fichiers, que vous pouvez utiliser directement avec les produits d'Itris Automation :

- Un fichier dont l'extension est AWL, il s'agit du code CONT (Ladder), LIST et LOG (Logigramme) de votre application
- Un fichier dont l'extension est SCL, il s'agit du code SCL de votre application (s'il y en a)
- Un fichier dont l'extension est GR7, il s'agit du code Grafcet de votre application (s'il y en a)
- Un fichier dont l'extension est DIF, il s'agit de la description de l'arborescence de votre application
- Un fichier dont l'extension est ASC, il s'agit de la table de mnémoniques de votre application

Chapter 8

Itris Import/Export Tool - TIA Portal

8.1 Présentation de Itris Import/Export Tool

L'outil d'import/export de code de Itris Automation vous permet de réaliser simplement l'export et l'import de vos codes automates, dans le but de les utiliser dans PLC Checker, PLC Docgen, ICS Monitoring et PLC Converter.

Pour plus d'informations, notamment relatives à l'installation et à l'utilisation de l'outil, veuillez vous reporter à la documentation spécifique à l'outil **Itris Import/Export Tool**. La présente documentation vient compléter cette dernière dans le cas où vous utiliseriez les plugins dédiés à TIA Portal.

8.2 Liste des plugins disponibles pour TIA Portal

Chaque version de TIA Portal requière un plugin différent pour fonctionner, et l'inter-compatibilité entre plugins n'est pas assurée. Il conviendra donc d'en tenir compte lors de la réalisation de votre installation.

À ce jour, des plugins sont disponibles pour les versions suivantes de TIA Portal :

- Version 13 SP1
- Version 13 SP2 : prendre le même plugin que pour la version 13 SP1
- Version 14 : Ne peut pas cohabiter avec le plugin pour la Version 14 SP1
- Version 14 SP1 : Ne peut pas cohabiter avec le plugin pour la Version 14
- Version 15
- Version 15.1

8.3 Prérequis spécifiques à TIA Portal

Pour utiliser les plugins d'import/export avec TIA Portal, il vous faut remplir les conditions suivantes (les procédures pour les remplir sont proposées dans les paragraphes suivants) :

- Avoir TIA Portal installé sur l'ordinateur qui va effectuer les opérations,
- Avoir tous les modules complémentaires (Safety, WinCC…) utilisés par les projets à exporter installés sur l'ordinateur qui va effectuer les opérations,
- Avoir Openness,
- Avoir les fichiers d'autorisation de Siemens pour les versions 13 SP1, 13 SP2 et 14 de TIA Portal
- Traiter des projets enregistrés dans la même version que celle de TIA Portal
- Déverrouiller les blocs protégés

8.4 Installation des plugins

Les plugins pour TIA Portal s'installent de la même façon que les plugins pour les autres ateliers. Ils fonctionnent avec le logiciel "Openness", fourni par Siemens et faisant office de connexion entre TIA Portal et les plugins. Chaque version de TIA Portal dispose d'une version différente de Openness. Pour savoir comment installer Opennes en fonction de votre version de TIA Portal, veuillez vous reporter au paragraphe "Installation de Openness".

Une fois Openness installé, il va falloir effectuer les opérations décrites dans le paragraphe "Autorisation des utilisateurs" (une fois par ordinateur sur lequel vous installer au moins un plugin).

Les versions **13 SP1, 13 SP2 et 14** requièrent également de votre part de réaliser les opérations décrites dans le paragraphe "Fichiers d'autorisation".

8.4.1 Installation de Openness

Pour les versions **13 SP1, 13 SP2, 14 et 14 SP1** de TIA Portal, Openness est un programme à installer après TIA Portal. Vous le trouverez dans le dossier "Support" du DVD d'installation de TIA Portal. Si vous avez téléchargé TIA Portal et ne disposez pas d'un DVD d'installation, vous pouvez décompresser les fichiers d'installation dans un répertoire auquel vous pouvez accéder, pour y trouver le dossier "Support" (les fichiers sont par défaut extraits dans un répertoire temporaire).

*Si vous utilisez la version **13 SP1, 13 SP2 ou 14** de TIA Portal, vous devrez effectuer les opérations décrites dans le paragraphe "Fichiers d'autorisation".*

Pour les versions **15** et 15.1 de TIA Portal, l'installation de Openness se fait lors de l'installation de TIA Portal. Il vous faut alors le choisir dans l'étape vous donnant la liste des modules à installer. Si vous ne l'avez pas fait lors d'une précédente installation, vous pouvez le faire en relançant l'installation et en choisissant de modifier votre installation actuelle.

Dans tous les cas, suite à l'installation de Openness, l'étape décrite dans le paragraphe "Autorisation des utilisateurs" est à réaliser.

8.4.2 Autorisation des utilisateurs

Une fois Openness installé, il vous faut autoriser les utilisateurs du poste de travail à l'utiliser.

Pour cela, il faut vous rendre dans les paramètres de gestion de l'ordinateur de la façon suivante :

- Cliquez droit sur "Ordinateur" (ou "Ce PC" en fonction de votre version de Windows)
- Sélectionnez "Gérer"
- Dans la fenêtre qui s'ouvre, naviguez jusqu'à "Outils système" puis "Utilisateurs et groupes locaux" puis "Groupes"

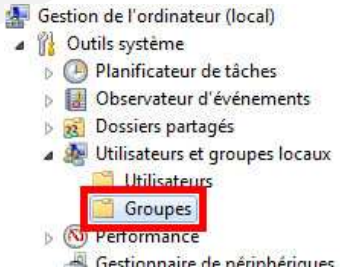


Figure 8.1:

- Vous devriez voir le groupe d'utilisateur "Siemens TIA Openness", dans la partie centrale de l'écran, double-cliquez dessus
- Si le nom de l'utilisateur de l'ordinateur que vous souhaitez autoriser à utiliser Openness apparaît pas la fenêtre qui s'ouvre, vous avez terminé cette procédure. Sinon, cliquez sur "Ajouter..." puis tapez le nom de votre utilisateur

- Cliquez sur "Vérifier les noms", le nom de votre utilisateur devrait être complété et souligné, vous pouvez cliquer sur "Ok" pour la valider, puis sur "Appliquer" pour que la modification soit prise en compte.
- Une fois l'utilisateur ajouté, redémarrez votre ordinateur.

Une fois cette procédure terminée, si vous utilisez les versions **13 SP1, 13 SP2 ou 14** de TIA Portal, complétez le paragraphe "Fichiers d'autorisation", pour les autres versions, vous êtes prêt(e) à utiliser les plugins.

NOTE	**Sur Windows 10 édition Familiale, il n'est pas possible d'ajouter les utilisateurs au groupe "Siemens TIA Openness"**

8.4.3 Fichiers d'autorisation

Si vous utilisez les versions **13 SP1, 13 SP2 ou 14** de TIA Portal, Siemens vous impose de déclarer l'utilisation de Openness auprès de ses services. Pour cela, il ne vous est pas possible d'utiliser Openness sans que vous n'installiez des fichiers qui vous sont donnés sur simple demande par Siemens.

Pour les obtenir, Siemens a mis à votre disposition un guide à l'adresse suivante :
TIA Portal Openness Guidance Usage File Enabler File .

Il n'est pas nécessaire de faire l'installation des fichiers vous-même, il vous suffit pour cela de lancer **Itris Import/-Export Tool**. Lorsque vous tenterez de faire une opération qui requière l'installation de ces fichiers, il vous proposera de les installer pour vous :

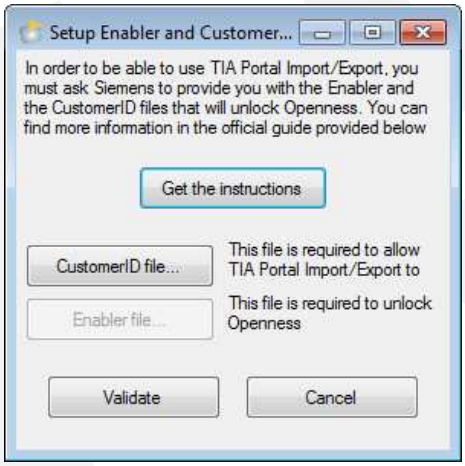


Figure 8.2:

8.5 Avertissements spécifiques à TIA Portal

8.5.1 Possibilité d'exporter les codes non compilés

Les codes non compilés ne peuvent pas être exportés par Openness. Nous vous recommandons de compiler les projets entiers lorsque vous souhaitez effectuer des opérations sur leurs programmes.

NOTE	**Les plugins d'import et d'export ne compilent pas les programmes pour vous afin de ne pas modifier votre projet, il vous revient donc de le faire.**

8.5.2 Possibilité d'exporter les codes protégés

Les plugins n'exportent pas les codes protégés afin de ne pas remettre en cause la politique de protection des codes définie par l'utilisateur.

De ce fait, il convient de supprimer la protection du code avant de procéder à l'exportation de ces codes. Cela permet aux utilisateurs de choisir quels sont les éléments auxquels Itris peut accéder.

Pour les codes Safety, la protection peut être retirée via la procédure suivante :

- Cliquer sur **Safety Administration**



Figure 8.3:

- Dans **Access protection** cliquer sur **Change**
- Taper le mot de passe dans la case correspondante
- Cliquer sur le bouton "Revoke" pour indiquer que vous ne souhaitez plus utiliser de mot de passe

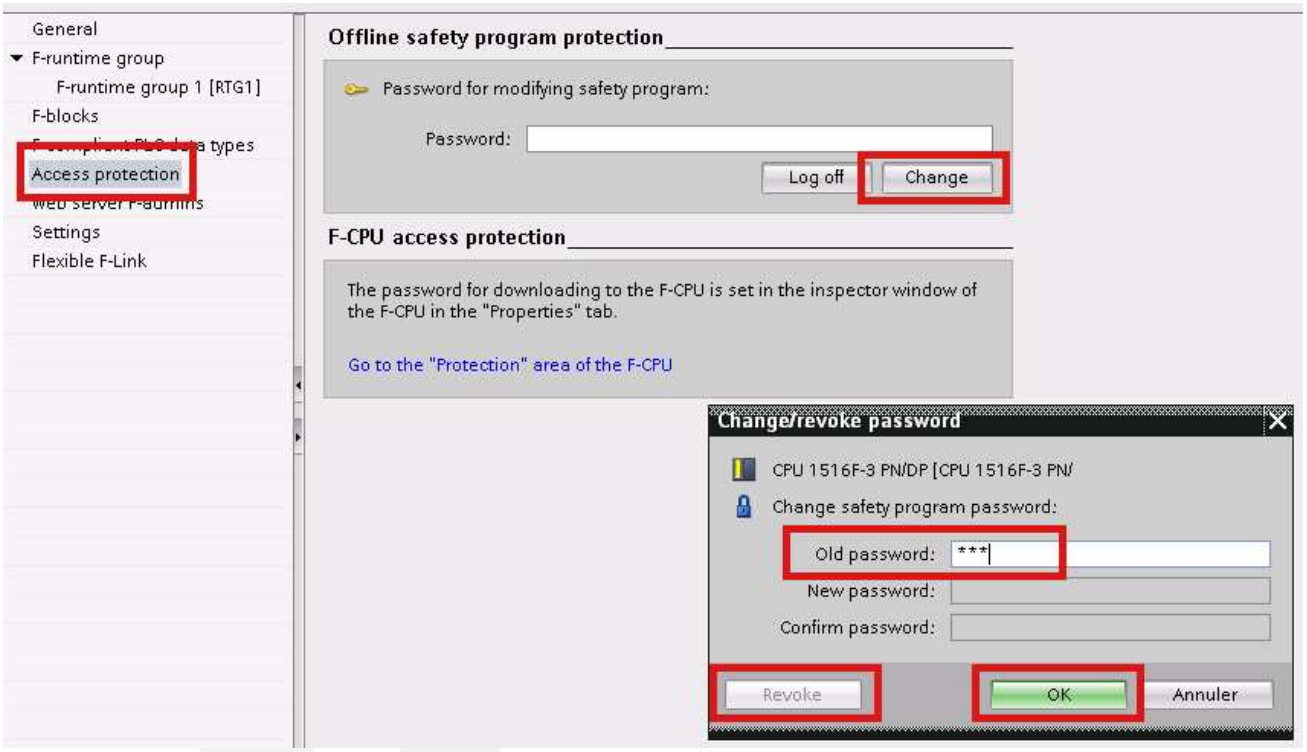


Figure 8.4:

- Valider et compiler

NOTE	**Se connecter ("Login") ne suffit pas à retirer la protection du programme**

8.5.3 Versions des projets

Les fichiers de projet sont sauvegardés dans une version indépendante de TIA Portal. Ainsi, vous pouvez sauvegarder un projet en Version 14 et l'éditer avec TIA Portal Version 14 SP1 sans vous en rendre compte.

Openness ne supporte pas ce cas de figure et doit être aligné avec la version du projet qu'il manipule. Ainsi, si vous utilisez TIA Portal Version 14 **SP1** et donc Openness Version 14 **SP1**, vous ne pouvez ouvrir que des projets sauvegardés en Version 14 avec **Itris Import/Export Tool** car votre version de Openness ne va pas le permettre.

Pour vous assurer que vos projets sont dans le bon format, ouvrez-les avec TIA Portal, puis allez dans le menu "Fichier". Si l'option "Mise à Niveau" n'est pas grisée, c'est que votre projet n'est pas dans le bon format pour Openness, il vous faudra donc le mettre à niveau via ce menu.

8.5.4 Autorisation du plugin

Lorsque vous exécutez une opération pour la première fois avec TIA Portal ce dernier vous demande si vous souhaitez autoriser le plugin à accéder à Openness via la fenêtre suivante :

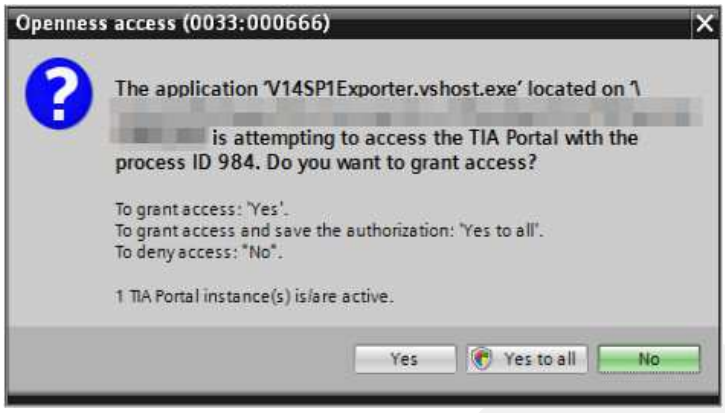


Figure 8.5:

Pour poursuivre l'exécution, il vous faut choisir l'une des deux options affirmatives. La seconde, "Yes to all", vous assurera de pouvoir utiliser **Itris Import/Export Tool** en mode console (sans intervention utilisateur) et vous permettra de ne plus avoir à autoriser le plugin dans le futur.

Chapter 9

Itris Import/Export Tool - Unity

9.1 Présentation de Itris Import/Export Tool

L'outil d'import/export de code de Itris Automation vous permet de réaliser simplement l'exportation et l'importation de vos codes automates, dans le but de les utiliser dans PLC Checker, PLC Docgen, ICS Monitoring et PLC Converter.

Pour plus d'informations, notamment relatives à l'installation et à l'utilisation de l'outil, veuillez vous reporter à la documentation spécifique à l'outil **Itris Import/Export Tool**. La présente documentation vient compléter cette dernière dans le cas où vous utiliseriez les plugins dédiés à Unity.

9.2 Prérequis spécifiques à Unity

Itris Import/Export Tool est capable d'exporter du code automate depuis Unity, sans que vous n'ayez d'autres opérations à faire que d'installer l'unité principale, le plugin Unity et Unity lui-même sur le même ordinateur.

9.3 Avertissements spécifiques à Unity

9.3.1 Compatibilité avec les anciennes versions de PLC Checker, PLC Docgen et PLC Converter

Étant donné que le système d'import/export est plus récent que certaines versions des produits d'Itris Automation, le format d'export ou d'import du code utilisé par les produits d'Itris Automation et **Itris Import/Export Tool** sont différents et incompatibles.

Dans le cas où vous voudriez utiliser le module d'export de code pour vous servir de PLC Checker et PLC Docgen, il vous faudrait ouvrir l'archive créée par **Itris Import/Export Tool** pour y trouver un fichier dont l'extension est ".xef". C'est ce fichier que vous pouvez envoyer sur la plateforme d'Itris Automation.

Chapter 10

Concepts

10.1 Règles

Cette section introduit une partie des concepts utilisés dans le fichier de règles.

10.1.1 Généralités

Un fichier de règles est constitué par une structure hiérarchique pouvant être représentée par un arbre. Les noeuds de l'arbre sont constitués par des groupes de règles (rulesGroup). Les feuilles (à l'exception des groupes de règles utilisés pour définir des filtres réutilisables) sont la description des règles à vérifier.

10.1.2 Activation des règles

Les noeuds et les feuilles du fichier de règles peuvent être activés ou désactivés en cochant ou non la boîte affichée à leur gauche. Un noeud désactivé (ainsi que toute sa descendance) ne sera pas lu et donc pas utilisé lors de l'exécution de PLC Checker. Cela permet de décomposer un fichier de règles en plusieurs parties indépendantes, et de demander telle ou telle vérification lors de l'exécution de PLC Checker.

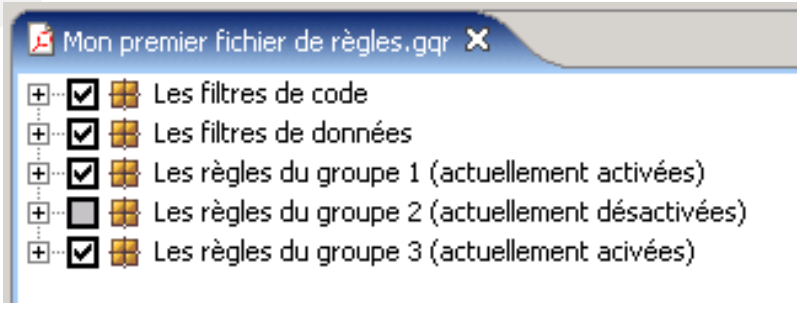


Figure 10.1: Activation de règles

10.1.3 Nom de règles

Cette propriété est nommée "**name**".

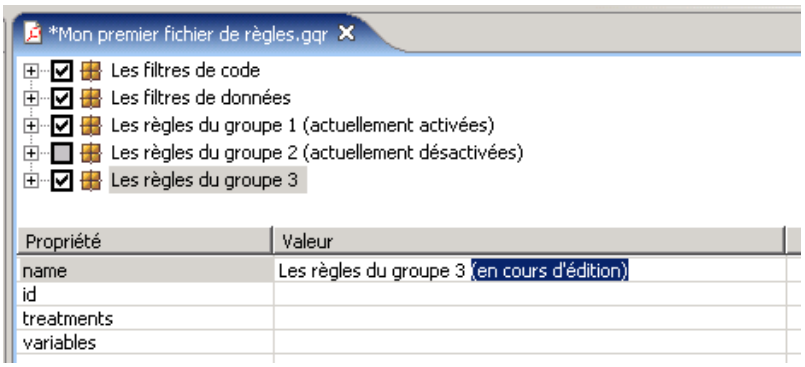


Figure 10.2: Nommage d'une règle

Chaque élément du fichier de règles a un nom. Ce nom est un commentaire indiquant ce que représente l'élément. Pour le changer, saisissez la propriété "Name".

Note : Ce nom est utilisé dans le fichier résultat dans la partie "Règles vérifiées". Il est donc vivement conseillé de donner systématiquement un nom à chaque élément du fichier de règles.

10.1.4 Identificateur de règles

Cette propriété, appelée "**id**", est utilisée lorsque vous désirez réutiliser le sous-ensembles de codes ou de données définis par cet élément (via les propriétés treatments et/ou variables) ailleurs dans le fichier de règles.

Par exemple, vous pouvez définir l'ensemble des télécommandes de votre application (un ou plusieurs intervalles de variables). Un élément (de type rulesGroup) de votre fichier de règles représentera cet ensemble. Donnez-lui comme nom (propriété name) "Définition de l'ensemble des télécommandes". Donnez-lui comme identificateur (propriété id) "télécommandes". Lorsque vous aurez besoin d'ajouter une règle effectuant une vérification sur les télécommandes, vous utiliserez #telecommandes comme filtre de données pour cette nouvelle règle.

Note : Les identificateurs NE doivent PAS contenir de caractères spéciaux (+ - # @ : . ? * ...) ou de séparateurs (espace tab...). Limitez-vous aux caractères alphanumériques (A..Z a..z 0..9) et à l'underscore (_).

10.1.5 Filtre de code

Cette propriété, appelée "**treatments**", permet de définir un sous-ensemble de codes parmi tout le code de l'application.

Voir le chapitre sur les types de filtres pour plus d'exemples.

10.1.6 Filtre d'éléments

Cette propriété, appelée "**variables**", permet de définir un sous-ensemble d'éléments parmi l'ensemble des éléments (variables, types, instructions, boites fonctionnelles, ...) de l'application.

Voir le chapitre sur les types de filtres pour plus d'exemples.

10.1.7 Validation d'une règle ou d'un ensemble de règles à l'aide d'autres règles

Cette propriété, nommée "**enabledIds**", contient une expression booléenne d'id de règles. Elle permet de valider une règle ou un ensemble de règles (rulesGroup) à partir d'un ou plusieurs noeuds.

Lorsque cette propriété est évaluée à true, la règle ou l'ensemble de règles (rulesGroup) est validé. Les ID utilisés dans l'expression booléenne sont évalués à "true" si la règle correspondante est validée et à "false" sinon.

Cette expression booléenne supporte les opérateurs OR, AND et NOT mais ne supporte pas les parenthèses. Les notations standards (ID1 OR ID2) et post-fixée (ID1 ID2 OR) sont supportées.

Les ID de règles utilisées doivent être définies en amont de leur utilisation dans la propriété "enabledIds".

Exemples : Voici quelques exemples de valeur de la proriété enabledIds d'une règle nommé regle3, avec ID1, ID2 et ID4 les valeurs des propriétés ID de règles définies précédemment et nommées respectivement regle1, regle2 et regle4 :

1. enabledIds="ID1 OR ID2" : regle3 est validée lorsque regle1 OU regle2 sont validées.

2. enabledIds="ID1 ID2 OR" : regle3 est validée lorsque regle1 OU regle2 sont validées.
3. enabledIds="ID1 AND ID2" : regle3 est validée lorsque regle1 ET regle2 sont validées.
4. enabledIds="ID1 ID2 AND" : regle3 est validée lorsque regle1 ET regle2 sont validées.
5. enabledIds="ID1 ID2 ID4 OR AND" : regle3 est validée lorsque regle1 ET (regle2 OU regle4) sont validées.
6. ~~enabledIds="ID1 AND (ID2 OR ID4)"~~ : les parenthèses ne sont pas supportées. Résultats inattendus.
7. enabledIds="" : regle3 peut être validée ou non par l'utilisateur en utilisant la boîte à cocher associée sur l'éditeur de règles (voir ci-dessus "Activation d'une règle")
8. enabledIds="true" : regle3 est toujours validée, l'utilisateur ne peut pas utiliser la boîte à cocher pour dévalider cette règle.
9. enabledIds="false" : regle3 est toujours dévalidée, l'utilisateur ne peut pas utiliser la boîte à cocher pour valider cette règle (ne faudrait-il pas tout simplement supprimer cette règle ?).

10.1.8 Validation d'une règle ou d'un ensemble de règles parmi la liste de règles

Cette opération fait appel à la propriété "**groupIds**". Cette propriété "groupIds" contient une liste d'id de règles. Elle permet de valider une et une seule règle ou un ensemble de règles (rulesGroup) parmi la liste de règles.

Exemple: groupIds="ID1 ID2 ID3". Si les noeuds ayant pour id ID1, ID2 et ID3 ont cet attribut groupIds, alors, durant l'édition du fichier de règles, le plugin "éditeur de règles" s'assure qu'un seul noeud parmi les trois est validé.

10.1.9 Identificateur pour l'export au format .csv

Cette propriété, appelée "**csvId**", permet d'ajouter 4 colonnes dans le fichier csv "Comma Separated Value" qui est généré lorsque on enregistre un fichier résultat en donnant l'extension "csv". Ces colonnes contiennent le compteur de toutes les occurrences trouvées au dessous du noeud en question, mais aussi le compteur des fatals, erreurs, warnings et info.

Cela permet par exemple d'obtenir un résumé par groupe de règles, des compteurs de messages (fatal, erreur, warning, info).

10.1.10 Visibilité du nœud

Cette propriété, appelée "**visible**" (false), permet de cacher le nœud et son sous-arbre lors de l'édition du fichier de règles par l'éditeur de règle lorsque celui-ci est protégé (propriété locked="true" du nœud racine du fichier de règles).

10.1.11 Modification automatique de la visibilité du noeud

Il est possible de rendre un nœud et son sous-arbre visible ou invisible selon une condition dépendant de l'activation ou la désactivation d'autre nœuds. Cela permet par exemple de masquer certaines règles qui n'ont pas lieu d'être pour un type d'automate donné.

Cela se fait en utilisant l'attribut **visibleId**. Les expressions possibles pour la valeur de cet attribut sont les même que pour l'attribut enabledIds.

10.1.12 Liste des propriétés éditables lorsque le fichier est protégé

La propriété, appelée "**editable**", contient la liste de propriétés qui peuvent être changées lorsque le fichier de règles est protégé. (propriété locked="true" du noeud racine du fichier de règles)

10.1.13 Sévérité des erreurs

Cette propriété, nommée "**severity**", permet de définir la manière dont les erreurs sont ajoutées dans le fichier résultat. Elle peut prendre les valeurs "fatal", "error", "warning", "info" ou "nolog".

L'utilisation de la valeur "nolog" permet de seulement compter le nombre d'occurrences d'objets (code ou données) satisfaisant à une ou plusieurs contraintes définies par une ou plusieurs règles.

10.1.14 (propriété "warningThreshold")

La propriété "**warningThreshold**" est utilisée pour définir un seuil à partir duquel monter un message de Warning. Lorsque le nombre d'occurrence d'objet (variable et code) ne répondant pas à la règle, dépasse ce seuil, un warning est remonté. La plupart du temps, la propriété "severity" vaut alors "nolog" pour ne pas remonter les messages pour chaque élément mais uniquement lors du passage du seuil.

- Si le seuil est positif, le warning apparait si la valeur est supérieure à la valeur absolue du seuil. ex: si seuil=4 et valeur=5 alors valeur > abs(seuil) et donc un warning est remonté.
- Si le seuil est négatif, le warning apparait si la valeur est inférieure à la valeur absolue du seuil. ex: si seuil=-4 et valeur=3 alors valeur < abs(seuil) et donc un warning est remonté.
- Si le seuil est compris entre -1 et 1, il est considéré comme un pourcentage et la valeur testée est donc le rapport entre le nombre d'objets ne répondant pas à la règle et le nombre d'objets testés par la règle. ex: si seuil=0.7, nbObjetTotal=10 et nbObjetFailed=8, alors nbObjetFailed/nbObjetTotal = 0.8 > seuil = 0.7 et donc un warning est remonté.
- Si le seuil est inférieur à -1 ou supérieur à 1, il n'est pas considéré comme un pourcentage et la valeur testée est donc le nombre d'objets ne répondant pas à la règle. ex: si seuil=1.1, nbObjetFailed=2 Alors nbObjetFailed > seuil et donc un warning est remonté.

10.1.15 (propriété "errorThreshold")

La propriété "**errorThreshold**" est utilisée pour définir un seuil à partir duquel monter un message d'erreur. Lorsque le nombre d'occurrences d'objet (variable et code) ne répondant pas à la règle dépasse ce seuil, une erreur est remontée. La plupart du temps, la propriété "severity" vaut alors "nolog" pour ne pas remonter les messages pour chaque élément mais uniquement lors du passage du seuil.

- Si le seuil est positif, une erreur apparaît si la valeur est supérieure à la valeur absolue du seuil. Exemple : si seuil=4 et valeur=5 alors valeur > abs(seuil) et donc une erreur est remontée.
- Si le seuil est négatif, une erreur apparaît si la valeur est inférieure à la valeur absolue du seuil. Exemple : si seuil=-4 et valeur=3 alors valeur < abs(seuil) et donc une erreur est remontée.
- Si le seuil est compris entre -1 et 1, il est considéré comme un pourcentage et la valeur testée est donc le rapport entre le nombre d'objets ne répondant pas à la règle et le nombre d'objets testés par la règle. Exemple : si seuil=0.7, nbObjetTotal=10 et nbObjetFailed=8, alors nbObjetFailed/nbObjetTotal = 0.8 > seuil = 0.7 et donc une erreur est remontée.
- Si le seuil est inférieur à -1 ou supérieur à 1, il n'est pas considéré comme un pourcentage et la valeur testée est donc le nombre d'objets ne répondant pas à la règle. Exemple : si seuil=1.1, nbObjetFailed=2, alors nbObjetFailed > seuil et donc une erreur est remontée.

10.1.16 (propriété "defaultResultId")

La propriété "**defaultResultId**" indique où l'on désire afficher l'erreur dans le fichier résultat (*.gqre) (arbre de la plateforme logicielle API[1]).

10.1.17 (propriété "errorMessage")

La propriété "**errorMessage**" donne le message d'erreur à afficher en cas d'erreur. Elle peut contenir des variables entourées des caractères et . Ces variables lors de l'émission de l'erreur sont remplacées par la valeur adéquate.

Les "variables" principales sont (respecter la casse) FUNCTION LOCATION VARIABLE et STR.

- FUNCTION contient le nom du bloc de code relatif à l'erreur.
- LOCATION contient le nom du bloc, mais aussi l'offset dans le bloc.

[1] Application Programming Interface

- VARIABLE contient le nom de la variable en erreur.
- STR est une information additionnelle. Elle est fonction de la règle vérifiée.

10.1.18 (propriété "noMatchMessage")

La propriété "**noMatchMessage**" donne le texte du message à afficher si l'ensemble sur lequel la règle s'est appliquée, s'avère vide.

10.1.19 (propriété "noMatchSeverity")

La propriété "**noMatchSeverity**" défini la sévérité (info, warning, error, fatal) du message **noMatchMessage**. Si **noMatchSeverity** n'est pas défini, alors la sévérité de la règle (voir propriété "severity") sera utilisée pour afficher le message **noMatchMessage**.

10.1.20 (propriété "isTextEditable")

L'attribut "**isTextEditable**" prend une valeur booléenne. Si sa valeur est vrai, l'action "Éditer pattern" apparait dans le menu.

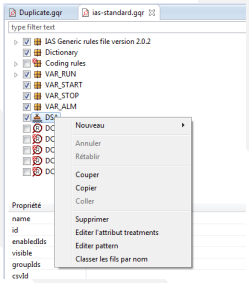


Figure 10.3:

L'action "Éditer le pattern" permet d'afficher un éditeur de texte afin de pouvoir insérer du code glips relatif à une règle (e.g. patternMatchingCheck).

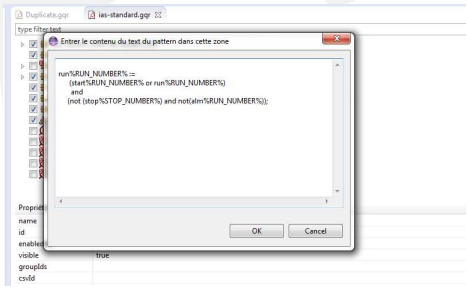


Figure 10.4: L'éditeur du texte

10.1.21 Générateur de valeur d'un attribut

Ce mécanisme permet de générer par automatiquement n'importe lequel des attributs de n'importe lequel des nœuds du fichier de règles.

Il est activé à chaque sauvegarde du fichier de règles lorsque ce dernier est édité avec le plugin d'éditeur de règle dans la plateforme Eclipse.

Pour utiliser de mécanisme il faut ajouter un attribut dit ".generator" au nœud dont on veut générer un certain attribut. Cet ajout se fait avec un éditeur texte car le plugin d'éditeur de règle n'affiche que les attributs standards.

exemple:

<element attribut1="blabla" attribut2="blabla" **attribut1.generator="code de génération"**>
je veux générer automatiquement l'attribut attribut1 de cet élément. J'ajoute donc à cet élément un attribut attribut1.generator.
</element>
Le contenu de cet attribut sera interprété à chaque sauvegarde du fichier de règles.

Langage supporté par les attributs .generator

Actuellement l'éditeur de règles supporte 2 opérateurs (UNION et liste de couples attribut/valeur).

UNION Cet opérateur est utilisé pour générer automatiquement les ensembles contenant l'union d'une liste d'ensemble.
syntaxe:
attribut.generator="UNION(FIRST_ID,LAST_ID)"
fonctionnement:
Pour l'élément ayant comme id FIRST_ID (<element id="FIRST_ID"/>) et pour tous ses frères placés après lui mais en s'arrêtant à l'élément ayant l'id LAST_ID, on réalise l'opération suivante :
Si l'élément (<element id="MON_ID" enabled="true"/>) en question est actif (enabled="true") alors on ajoute à l'attribut en question la chaîne +#MON_ID
Si LAST_ID n'est pas trouvé on arrête de concaténer lorsque on arrive sur le dernier frère de FIRST_ID.
exemple:

```
<element>
  <element id="ID1" enabled="true"/>
  <element id="ID2" enabled="true"/>
  <element id="ID3" enabled="false"/>
  <element id="ID4" enabled="true"/>
</element>
<element variables.generator="UNION(ID2,ID5)" variables="+#ID2+#ID4"/>
```

Suite de couples (;ID;valeur) Ce générateur contient une suite de couples. Le premier caractère de l'attribut .generator est utilisé comme séparateur pour tout l'attribut. En général on utilise le caractère ';'.
syntaxe:
attribut.generator="(;.+;.+)+"
Note:
le séparateur (ici ';') peut être changé et remplacé par exemple par 'l' : attribut.generator="|ID1|+#TOTO-#TITI|ID2|+#TUTU"
fonctionnement:
Pour chaque couple si l'élément ayant le bon id existe et est actif (<element id="ID" enabled="true"/>) alors la chaîne valeur est ajoutée à l'attribut généré.
exemple:

```
<element>
  <element id="ID1" enabled="true"/>
  <element id="ID2" enabled="false"/>
  <element id="ID3" enabled="true"/>
</element>
<element
   variables.generator=";ID1;+#ID1_CHILDS;ID2;+#ID2_CHILDS;ID3;+#ID3_CHILDS"
   variables="+#ID1_CHILDS+#ID3_CHILDS">
</element>
```

10.2 Filtres

Cette partie introduit le concept des filtres utilisés dans le fichier de règles.

10.2.1 Généralités

Les filtres permettent de définir des sous-ensembles d'éléments. Deux ensembles éléments sont pris en compte : l'ensemble du code de l'application et l'ensemble des éléments de l'application. Par conséquent, il y a deux types de filtres, les filtres de code et les filtres d'éléments. Les filtres de code sont définis grâce aux propriétés **treatments** des règles/groupes de règles du fichier de règles. Les filtres d'éléments sont définis grâce aux propriétés **variables** des règles/groupes de règles du fichier de règles. Pour chaque élément à vérifier, lors de l'exécution de PLC Checker, l'analyse est restreinte :

- tout d'abord, pour ce qui est du bloc de code contenant l'élément, au sous-ensemble décrit par la propriété **treatments** de la règle/du groupe de règles
- puis, pour ce qui est de l'élément lui même, au sous-ensemble décrit par la propriété **variables** de la règle/du groupe de règles

Exemple : la règle vérifiant qu'une variable n'est jamais écrite, donnera comme résultat l'ensemble des variables, parmi le sous-ensemble de variables défini par la propriété **variables**, qui ne sont pas écrites dans le sous-ensemble de code défini par la propriété **treatments**.

10.2.2 Limitations

Les propriétés décrivant des filtres **treatments** et **variables** ne doivent pas contenir d'espace (hormis dans les expressions régulières). Elles doivent obligatoirement commencer par l'opérateur + ou -.

10.2.3 Filtres de symboles

Les filtres de symboles permettent de définir des sous-ensembles en utilisant le symbole de l'objet (appelé parfois mnémonique). Les caractères génériques ? et * sont autorisés. Ils ont le sens usuel.

Exemple : DF_* (DF_TEMPHAUT et DF_TEMPBAS appartiennent au sous-ensemble) T?_* (TC_ON et TR_TEMP_EAU appartiennent au sous-ensemble).

10.2.4 Filtres de symboles utilisant les expressions régulières

Ces filtres de symboles permettent de définir des sous-ensembles en utilisant le symbole de l'objet (appelé parfois mnémonique) avec une syntaxe bien plus puissante que les filtres de symboles.

On utilise ce type de filtre en entourant avec des parenthèses "()" l'expression régulière décrivant l'ensemble des symboles que l'on souhaite définir.

Les méta-caractères :

- . : n'importe quel caractère. **Ex:** (t.t.) ⇒ toto, tati etc...
- | : alternative. **Ex:** (tataltiti) ⇒ tata ou titi
- () : groupement. **Ex:** (t(ilou)ti) ⇒ titi ou touti
- [] : classe de caractères. **Ex:** ([a-cD-F]g) ⇒ ag, bg, cg, Dg, Eg ou Fg
- ^ : permet de définir le ou les caractères interdits. **Ex:** ([^a-y]z) ⇒ az, bz, ne conviennent pas.
- \ : utilisation d'un méta-caractère ou quantificateur comme un simple caractère. **Ex:** (\. \| \(\) \[\] \\) ⇒ . | ( ) [ ] \

Les quantificateurs :

- * le ou les caractères qui précèdent peuvent apparaître **zéro** ou **plusieurs** fois. **Ex:** (.*) ⇒ n'importe quelle suite de caractères même la suite vide
- + le ou les caractères qui précèdent peuvent apparaître **une** ou **plusieurs** fois. **Ex:** ([0-9]+) ⇒ n'importe quel nombre entier entre 0 et 9999999...

- ? le ou les caractères qui précèdent peuvent apparaître **zéro** ou **une** fois. **Ex:** (test(_[a-z])?) ⇒ test, test_a, test_b, . . . test_z
- n le ou les caractères qui précèdent apparaissent exactement **n** fois. **Ex:** ([0-9]3) ⇒ 000, 001, . . . , 999
- n, le ou les caractères qui précèdent apparaissent au moins **n** fois. **Ex:** ([a-z]3,) ⇒ abc, abcd, kkkkkk, . . .
- n,m le ou les caractères qui précèdent apparaissent au moins **n** fois et au plus **m** fois. **Ex:** ([0-9]2,4) ⇒ 00, 01, . . . ,99, 000, 001, . . . , 999, 0000, . . . , 9999

Pour de plus amples informations, se reporter à l'assistant d'expressions régulières dans la fenêtre de recherche de texte de la plateforme Eclipse :

1. Ouvrir Eclipse
2. Ouvrir un fichier texte existant ou un nouveau fichier text sans titre.
3. Ouvrir la boite de recherche (Ctrl+F ou menu Edit / Find/Replace. . .)
4. Sélectionner l'option "expression régulière"
5. Sélectionner le champ de recherche, une ampoule jaune s'affiche sur la gauche du champ
6. Ouvrir l'assistant d'expression régulière (Ctrl+Space)

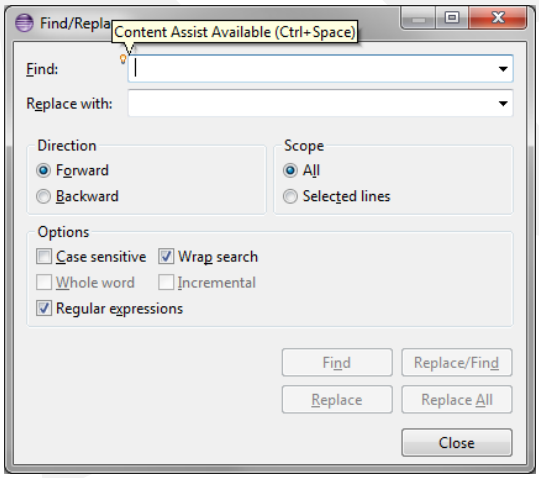


Figure 10.5: Ouvrir l'assistant d'expression régulière

7. Rechercher le contenu souhaité et le sélectionner pour afficher les exemples et explications;

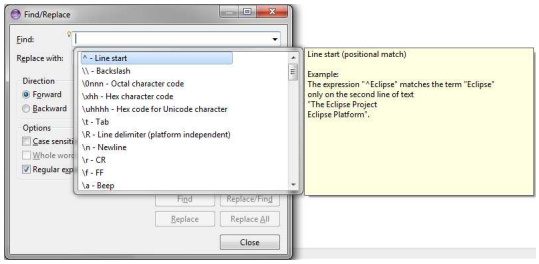


Figure 10.6: Assistant d'expression régulière

10.2.5 Filtres d'attributs

Les filtres d'attributs permettent de définir des sous-ensembles en utilisant une caractéristique de l'objet (code ou donnée). La listes des caractéristiques disponibles est fonction du type d'objet et de l'atelier de programmation (UNITY-PRO, PL7-PRO, STEP 7, RSLOGIX).

Un filtre d'attributs commence toujours par le mot clé "^" ou le mot clé "**!**".

Le mot clé "^" indique que l'objet ou un de ses ancêtres a une valeur de l'attribut qui convient.

Le mot clé "**!**" indique que l'objet lui-même a une valeur d'attribut correcte.

Après le mot clé on donne le nom de l'attribut.

Après le nom de l'attribut on place le mot clé "**=**" avant le filtre définissant l'ensemble des valeurs autorisées pour l'attribut.

Le filtre est entouré par des parenthèses "()". Si on utilise une expression régulière, cette dernière sera comme pour les filtres, entourée elle aussi de parenthèses "()".

Exemple : ^language=(+ladder) ou !comment=(+(.*réserve.*))

10.2.6 Filtres d'adresses absolues

Les filtres d'adresses absolues permettent de définir des sous-ensembles en utilisant l'adresse de l'objet.

Un filtre d'adresse absolue commence toujours par le mot clé @.

On indique ensuite le nom du plan mémoire où se trouve le ou les objets composant le sous-ensemble (ex: MW ou KD).

On place, après le nom du plan mémoire et avant l'intervalle d'adresse appartenant au sous-ensemble, le caractère ":".

Pour finir, on indique les bornes de l'intervalle. Il existe quatre syntaxes possibles pour les intervalles:

- Une adresse unique (par exemple: @MW:10)
- Un intervalle finie (par exemple: @MW:0..100)
- Un intervalle sans borne supérieure (par exemple: @MW:100..*)
- Le plan mémoire en entier (par exemple: @MW:*)

NOTE : il est aussi possible de sélectionner un élément à partir de son adresse en utilisant un filtre d'attribut pour tester l'attribut plc_address.

10.2.7 Réutilisation d'un filtre

Un élément du fichier de règles peut être utilisé pour définir un sous-ensemble. Dans ce cas on utilise un élément de type **rulesGroup**. On lui donne un identificateur (propriété **id**). Lorsque l'on veut utiliser ce sous-ensemble on utilise le jeton **#**

- **Exemple 1:**
  - Si le rulesGroup nommé "digital inputs" contient la définition des entrés TOR et a l'ID: **dig_in**
  - et Si le rulesGroup nommé "analog inputs"contient la définition des entrés ANA et a l'ID: **ana_in**
  - alors, une règle travaillant sur les entrées peut réutiliser les définitions faites dans les filtres des rulesGroup précédent en utilisant pour cette règle la propriété variables suivante : **+#dig_in+#ana_in**
- **Exemple 2:**
  - Si le rulesGroup nommé "digital inputs" contient la définition des entrés TOR et a l'ID: **dig_in**
  - et Si le rulesGroup nommé "analog inputs"contient la définition des entrés ANA et a l'ID: **ana_in**
  - et si le rulesGroup nommé "inputs", avec l'ID="**all_in**", peut regrouper toutes les entrées en utilisant la propriété variables suivante : **+#dig_in+#ana_in**
  - alors, une règle travaillant sur les entrées peut réutiliser la définition faite précédemment en utilisant la propriété variables suivante : **+#all_in**

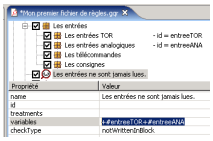


Figure 10.7: Exemple: réutilisation d'un filtre

10.2.8 Intersection de filtres

Les sous-ensembles (code et données) utilisés lors de l'analyse sont les intersections des sous-ensembles définis pour cette règle, ainsi que de tous ceux de ses ancêtres.

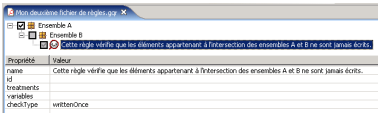


Figure 10.8: Intersection de filtres

10.2.9 Union de filtres

Vous pouvez définir un sous-ensemble comme étant l'union de deux ou plusieurs sous-ensembles en utilisant l'opérateur +.

Exemple : +DF_*+AL_*+ER_*

10.2.10 Différence de filtres

Vous pouvez définir un sous-ensemble comme étant la différence de deux sous-ensembles en utilisant l'opérateur **-**. Dans ce cas, le sous-ensemble **+A-B** est constitué par l'ensemble **A** auquel on enlève les éléments de l'ensemble **B**.

Exemple : +DF_*-DF_

10.3 Attributs Glips

Cette partie introduit les attributs Glips associés aux noeuds de l'arbre glips représentant l'application automates. Ces attributs sont souvent utilisés dans le fichier de règles pour sélectionner (filtre) ou vérifier (règle) des éléments.

- **action_type** : retourne le type (soit une fonction, une procédure, une tâche ou une réactive) d'une action (une entité qui contient du code et peut être appelée). Par contre, lorsqu'on demande action_type sur une variable, rien n'est retourné. Il faut distinguer la différence entre:
  - Fonction : retourne une valeur et elle est appelée dans une expression (ex: "a := b + myfunction(c, d); ")
  - Tâche : elle dépend d'un déclencheur (périodique, cyclique . . .) pour qu'elle s'exécute (ex: "start mytask trigger . . . ; ")
  - Procédure : elle est appelée dans une instruction (ex: "myprocedure(a, b, c, d); ")
  - Réactive : une procédure avec une évolution séquentiel.
- **application** : cet attribut indique le nom de l'application automate
- **artificial** : Retourne Vrai si le noeud a été créé par Glips et n'existe pas dans l'application originale
- **author** : retourne le nom de l'auteur de l'application ou d'un prototype (DFB, FB, ADD_ON,. . .)
- **auto_run** : pour les applications PL7 pro, il retourne la valeur (1=enabled, 0=disabled) de l'option auto_run.
- **backward** : Indique si, sur instruction GOTO, la direction est vers l'arrière. Ex:

```
        instruction0;
label1: instruction1;
        goto label2; <----------- backward = "false" or empty
        instruction3;
label2: instruction4;
        goto label2; <----------- backward = "true"
```

- **calledcount** : Retourner le nombre d'appel d'un POU donné
- **calldepthmin** : retourne la profondeur minimale d'un appel de chaque action/POU (procédure, fonction, FB, FC, Routine, Sous-routine...)
- **calldepthmax** : retourne la profondeur maximale d'appel de chaque action/POU (procédure, fonction, FB, FC, Routine, Sous-routine...)
- **callproc** : cet attribut retourne le nombre de bloc fonction dans une routine.
- **comment** : cet attribut contient le commentaire de l'élément (variable, instruction, type, prototype, bloc de code, ...) testé
- **comment.1** : contient le commentaire associé au bit numéro 1 d'un nœud (déclaration de variable...). Vous pouvez utilisez comment[ii].oo pour avoir le commentaire associé au bit oo de l'indexe ii du noeud
- **constant** : retourne la variable constante non localisée
- **dateandtime** : cet attribut contient l'heure et la date associé à une entité
- **deadaction** : retourne true si le code d'un POU est vu comme non exécuté (cela signifie que le POU est n'est pas appelé), si non il retourne false
- **evg** : retourne la valeur de la complexité essentiel(le nombre d'instructions discontinues GOTO, EXIT, RETURN) mesurée dans une action/POU (procédure, fonction, FB, FC, routine, sous-routine). Dans l'exemple ci-dessous, l'attribut evg vaut 2 pour la procédure FC1 car le code contient 2 instructions Return.

```
procedure FC1 ( b : in int ;
        c : in int ;
        a : out int) is
  my_Var : int := 0;
begin
   a := my_var;
   if b < 0 then
      return;
      if c < 0 then
         return ;
   a:=b+c;
end FC1;
```

- **extvar** : nombre de variables externes ayant une référence dans une action/POU/bloc de code
- **extvarref** : nombre de références des variables externes dans une action/POU/bloc de code
- **fanoutcount** : nombre d'actions/POU/blocs de code appelés dans une action/POU/bloc de code donné
- **format** : dépend du type de noeud :
  - pour une liste d'instructions : il retourne sa format (invisible, rung, instr_block)
  - pour une instruction d'affectation : il retourne sa format (Set, Reset, Affect, Not_Affect, Hash_affect)
  - pour une instruction conditionnelle : il retourne sa format (If_Then_Else, IfDo, Case_When)
  - pour une instruction de boucle : il retourne sa format (While_Loop, Loop_Loop, For_Loop)

- **fullkind** :Indique le type d'un noeud Glips précédé par les types de ses parents avec l'espace comme séparateur. Ex : Dans le code ci-dessous, l'attribut fullkind de l'instruction a:=b+c; sera "PACKAGE ACTION IF IF AFFECT" car il s'agit d'une instruction d'affectation contenue dans un IF qui est dans un IF qui est dans une action (=procedure ou fonction ou reactive) qui est dans un package.

```
package fc is
begin
   procedure My_FC ( b : in int ;
            c : in int ;
            a : out int) is
     my_Var : int := 0;
   begin
      a := my_var;
      if b > 0 then
         if c > 0 then
            a := b + c ;
         end if;
      end if;
   end My_FC;
end fc;
```

- **fullname** : Un attribut indique le nom complet de l'entité (du package à l'entité). c.a.d. le nom de l'entité précédé par celui de ses parents avec le point comme séparateur. Note les noms sont tous passés en minuscule. Dans l'exemple de code ci-dessus, la variable my_Var a un attribut fullname qui vaut "fc.my_fc.my_var" car la variable my_Var est dans une procédure nommée My_FC qui est dans un package nommé fc.

- **fulloriginalname** : contient le nom (mnémonique) d'une entité préfixé par les noms de tous ses parents, avec la même casse (majuscule/minuscule) que dans le fichier source. Dans l'exemple ci-dessus, l'attribut fulloriginalname pour la variable my_Var est "fc.My_FC.my_Var"

- **functionalmodule** : spécifique pour Pl7 pro et Unity pro. Il retourne le nom du module fonctionnel associé à une section/sous-routines

- **g7initstepcount** : retourne le nombre d'étapes initiales dans un Grafcet. Dans l'exemple ci-dessous, l'attribut g7initstepcount vaut 2.

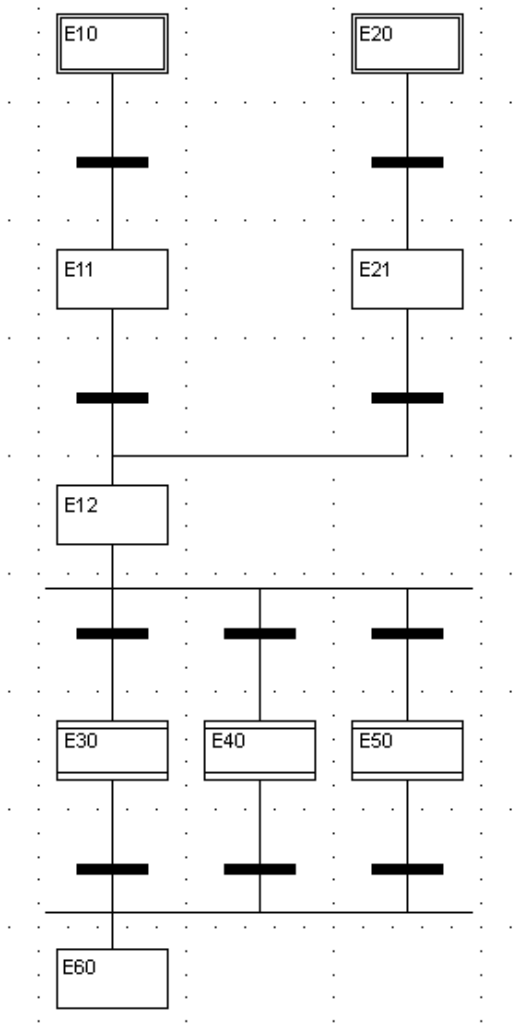


Figure 10.9: Grafcet exemple

- **g7stepcount** : retourne le nombre d'étapes (quelques soient leurs types) dans un Grafcet. Dans l'exemple ci-dessus, l'attribut g7stepcount vaut 9.
- **hasexternalreadref** : retourne true si l'entité a des références en lecture en dehors de son étendue (=où elle est déclarée). Cela signifie que si elle a au moins un indicateur qui pointe sur l'entité et qui est en mode lecture et qui est situé en dehors du périmètre où l'entité a été déclarée. Ex: retourne true si une variable locale d'un bloc est lue en dehors du bloc.
- **hasexternalref** : retourne true si l'entité a des références en dehors de son étendue (=où elle est déclarée). Cela signifie que si elle a au moins un indicateur qui pointe sur l'entité et qui est situé en dehors du périmètre où l'entité a été déclarée. Ex: retourne true si une variable locale d'un bloc est utilisée en dehors du bloc.
- **hasexternalwriteref** : retourne true si l'entité a des références en écriture en dehors de son étendue (=où elle est déclarée). Cela signifie que si elle a au moins un indicateur qui pointe sur l'entité et qui est en mode écriture et qui est situé en dehors de son périmètre où l'entité a été déclarée. Ex: retourne true si une variable locale d'un bloc est écrite en dehors du bloc.
- **hasreadref** : Vrai si le mnémonique de l'entité est lu quelque part. Note : si l'entité est référencée par son adresse alors le mnémonique n'est pas vu comme référencé en lecture. Dans l'exemple suivant l'attribut hasreadref vaut true pour toto car le mnémonique toto est utilisé en lecture. Il vaut false pour titi car le mnémonique titi n'est utilisé qu'en écriture, puisque la lecture de la variable se fait par adresse et pas par mnémonique.

```
toto : bool at %M10;
titi : bool at %M20;
...
titi := %M20 OR toto;
```

- **hasref** : Vrai si le mnémonique de l'entité est lu ou écrit quelque part. Dans l'exemple ci-dessus l'attribut vaut true pour les deux variables toto et titi car les 2 mnémoniques sont utilisés en lecture et/ou en écriture.
- **haswriteref** : Vrai si le mnémonique de l'entité est écrit quelque part. Dans l'exemple ci-dessus l'attribut vaut true pour titi car le mnémonique titi est utilisé en écriture. Il vaut false pour toto car le mnémonique toto n'est utilisé qu'en lecture.
- **hs_distinctoperator** : Métrique d'Halstead : nombre d'opérateurs distincts
- **hs_distinctoperand** : Métrique d'Halstead : nombre d'opérandes distincts
- **hs_totaloperator** : Métrique d'Halstead :nombre total d'opérateurs
- **hs_totaloperand** : Métrique d'Halstead : nombre total d'opérandes
- **hs_vocabulary** : Métrique d'Halstead : vocabulaire
- **hs_programlength** : Métrique d'Halstead : longueur d'un programme
- **hs_calcprogramlength** : Métrique d'Halstead : longueur calculée d'un programme
- **hs_volume** : Métrique d'Halstead : volume
- **hs_difficulty** : Métrique d'Halstead : difficulté
- **hs_effort** : Métrique d'Halstead : effort
- **hs_timerequired** : Métrique d'Halstead : temps requis pour l'implémentation
- **hs_numberdeliverybug** : Métrique d'Halstead : nombre de bugs livrés
- **initvalue** : Renvoie la valeur initiale de la variable
- **instancecount** : Retourne le nombre d'instances d'un type donné
- **interval** : retourne pour PL7 Pro la période d'un tâche (ex: "T#5 MS"). Pour une tâche cyclique l'attribut interval vaut "T#0 MS"
- **intrarunggoto** : dans l'instruction GOTO, cet attribut retourne si la destination de l'étiquette est dans le même rung/réseau que l'instruction GOTO. EX :

```
NETWORK
        goto label1; <----------- intrarunggoto= "true" ; backward = "false" or empty
        instruction1;
label1: instruction2;
        goto label2; <----------- intrarunggoto= "false" ; backward = "false" or empty
END NETWORK
NETWORK
        instruction3;
label2: instruction4;
END NETWORK
```

- **language** : cet attribut contient le nom du langage de programmation (selon la norme IEC61131 : ladder, litteral, grafcet, fbd ou list) de l'élément testé (instruction, réseau, bloc de code, . . .)
- **library** : cet attribut indique si l'élément testé appartient à une bibliothèque (library="true")
- **locked** : True if POU or FB is locked

- **version** : cet attribut peut être disponible sur différents types de noeuds en fonction de type API[2] (application, POU, FB, programme)
- **longcomment** : cet attribut est spécifique au Step7 puisque les blocs (OB, FC, FB, DB) peuvent avoir deux commentaires :
  - un premier commentaire sur une seule ligne dans la table des mnémoniques (fichier *.asc), ce commentaire est associé à l'attribut "comment".
  - un deuxième commentaire sous forme d'une longue description qui peut être mise pour le bloc en question dans le fichier source (*.SCL, *.AWL), ce commentaire est associé à l'attribut "longcomment".
- **loopvar** : retourne yes si une variable est utilisée comme un index d'une boucle mais retourne rien.
- **macro** : attribut utilisé pour distinguer un macro graphe (quand cet attribut contient "yes") des autres graphes SFC. Il est utilisé quand vous voulez vérifier le nombre d'étapes initiales (qui doit être 1) pour tous les SFCs sauf les macros (qui doit être 0). Dans l'exemple ci-dessous, l'attribut macro vaut "no" pour le graphe1 car il s'agit d'un grafcet ordinaire. Il vaut "yes" pour le graphe2 car il s'agit d'un macro graphe.

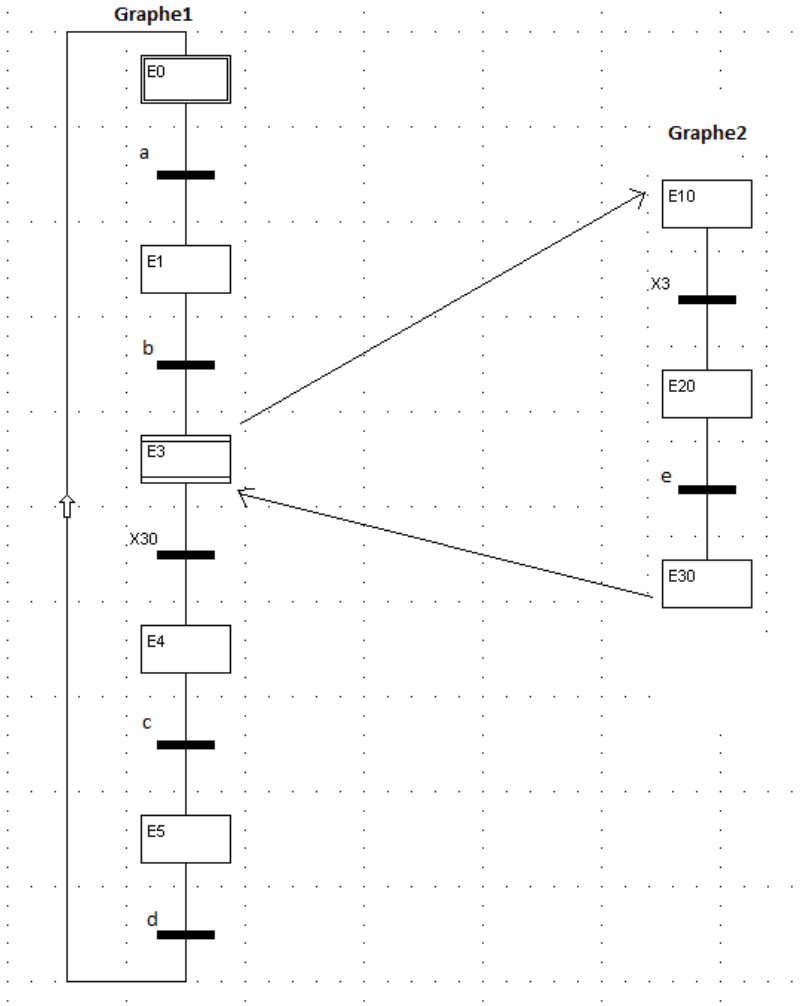


Figure 10.10: macro graphe

- **mnemonic** : dans une version ancienne de PLC Checker, cet attribut est utilisé pour indiquer si la variable a un mnémonique (maintenant il doit être l'attribut "no_mnemonic"="true" or "false"

[2] Application Programming Interface

- **mode** : Pour step7 et unity, il indique le mode de la variable (in, out, in_out, temp, stat pour step7 et IN, in_out, OUT, public, private pour unity). Dans l'exemple ci-dessous, l'attribut mode vaut "in" pour les variables var1 et var2 et vaut in_out pour la variable out1.

```
procedure FB_1
   (out1 : in out int;
    var1 : in int;
    var2 : in int) is
    begin
       out1:= var1 + var2 + out1;
    end FB_1;
```

- **multiinstance** : Dans Step 7, il retourne si un FB a plusieurs instances ("multiinstance"="yes")
- **mw_init_zero** : Dans PL7 Pro, il indique si la mémoire MW sera initialisée à 0 en démarrant l'automate.
- **nbofbranches** : retourne le nombre de branches par divergence dans un Grafcet. Dans l'exemple ci-dessous, l'attribut nbofbranches vaut 4 pour la première divergence et vaut 11 pour la deuxième.

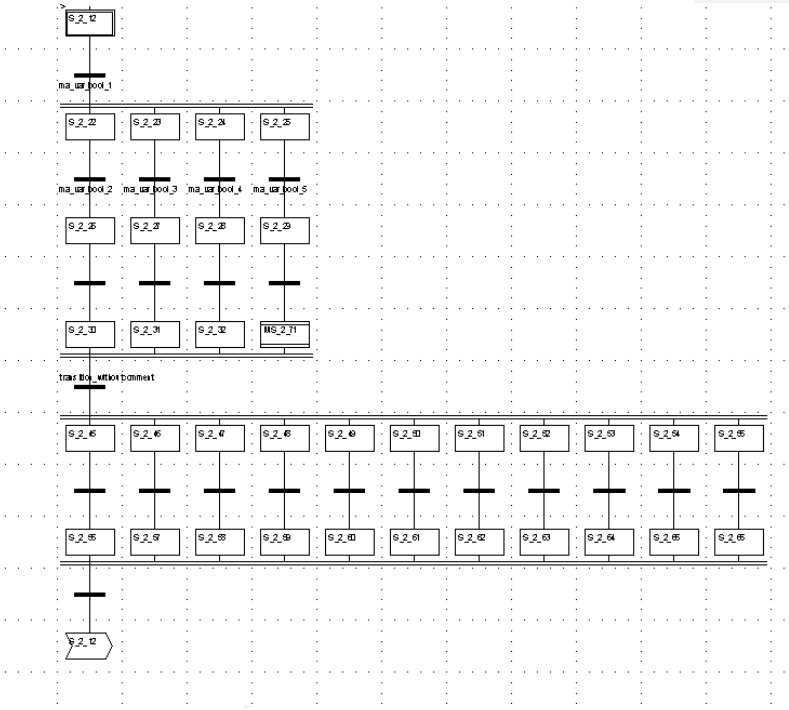


Figure 10.11:

- **networkcount** : retourne le nombre de réseaux/rungs dans un bloc de code. Note : le concept de réseau n'existe pas pour certains langages (comme le littéral) et/ou certains ateliers (comme UnityPro).
- **no_mnemonic** : Sur Step7, il indique si les variables n'avaient pas des mnémoniques
- notanattribute : cet attribut n'existe pas. Il est utilisé des fois dans le fichier de règles standard IAS juste pour créer une fausse règle qui est toujours à false et remonte toujours un message pour chaque élément testé.
- **originalname** : retourne le nom/symbole/mnémonique d'une entité (variable, UDT, fonction . . .) avec la même casse que le fichier source (lettres majuscules/minuscules).
- **parent_entity_type** : retourne le type des parents d'une entité.

- **plc_address** : Adresse topologique, par exemple %MW12 sur unity

- **plc_family** : Un attribut contient la famille "API[1]" de l'application (unity, step7, rockwell, pl7)

- **reference** : dépends du type de noeud :

  - pour une entité (variable, type, prototype, bloc de code, . . .) : retourne son originalname (ex : "ma_Var"),
  - pour une référence (désignateur, sélecteur) : retourne les originalnames des entités référencées (ex : "MaFonction.ma_Var"),
  - pour un system address (accès direct à une partie de la mémoire) : retourne le system address (ex : "@DB[102].0,0:16")
  - pour une valeur immédiate : retourne la valeur (ex : "12", "false", "42.2", "toto")
  - pour un intervalle : retourne l'intervalle (ex : "10..29")
  - pour les autres noeuds : retourne le type et l'id du noeud (ex : "N_INSTR_IF: 12345"). Note : l'id peut varier au moindre changement, il est donc déconseillé de l'utiliser pour un test (filtre ou règle).

- **rename** : L'entité renomme une autre entité (comme "alias" dans Rockwell PLC)

- **run_stop** : dans PL7 Pro, il retourne l'adresse (ex: "t#100 MS") de l'entrée physique qui contrôle le mode opératoire de l'automate (marche/arrêt).

- **rung** : retourne "true" pour une liste d'instructions rung/réseau

- **signature** : retourne une chaîne de caractères caractérisant une entité Glips.

- **sloc** : "Source Line Of Code", retourne le nombre d'instructions pour chaque bloc de code.

- **state** : retourne l'état d'une étape Grafcet ("Name_Initial" pour une étape initiale du grafcet, "Name_In" pour une étape d'entrée de la macro, "Name_out" pour une étape de sortie de la macro, "No_Name" pour une étape normale du grafcet). Dans l'exemple ci-dessous, l'attribut state vaut "Name_Initial" pour l'étape E0 et "No_Name" pour les autres étapes.

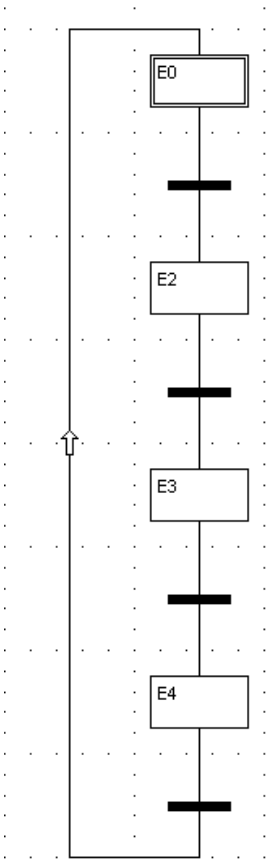


Figure 10.12: Grafcet exemple

- **temporary** : cet attribut est utilisé pour identifier les entités créées par l'analyseur Glips (le but est d'éviter de tester ces entités avec le PLC Checker)
- **title** : pour Step 7 et RSLogix, il retourne le titre des rung/réseaux
- **tmax** : pour PL7 Pro, il retourne le retard (ex: "t#100 MS") avant qu'un chien de garde est soulevé pour une tâche de durée assez longue.
- **transition** : pour unity, il retourne l'état de la transition du Grafcet.
- **type** : Indique le nom du type de la variable. Dans l'exemple ci-dessous, il vaut "bool" pour la variable var1 et "int" pour la variable var2.

```
 var1 : bool at %M100;
var2 : int at %M101;
...
if var1 then
   var2:=10;
end if;
```

- **typeadress** : retourne l'adresse du type de la variable (exemple on Step7: "FB 12", "UDT 100")
- **typeclass** : Indique la classe du type de la variable
- **typecodechecksumstring** : dans unity pro, la signature du code des DFBs

- **typekind** : Un attribut indique le type de la variable (tableau,enregistrement. . .)
- **typesignaturechecksumstring** : dans unity pro, la signature du model d'un DFB
- **unconstraint** : vrai si l'entité n'est pas contrainte. Par exemple un pointeur peut être traduit par un tableau non-contraint qui couvre l'ensemble du plan mémoire
- **var_mode** : Indique le mode de la variable (lire, écrire, entrée, sortie TOR, ANA, variable de communication)
- **varlocation** : retourne la localisation d'une variable
- **vg** : retourne la valeur de la complexité cyclotomique mesurée dans une action/POU (procédure, fonction, FC, FB, routines, sous-routines), il correspond au nombre de branches (ex: pour une instruction conditionnelle "IF THEN ELSE", le nombre des branches est 2 : la branche THEN et la branche ELSE)

10.4 Fichier résultat

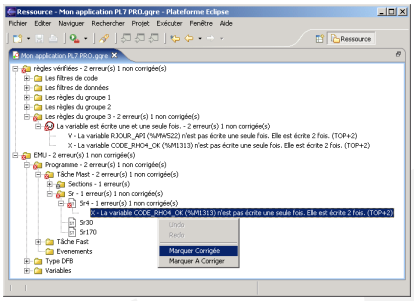


Figure 10.13: Exemple de fichier résultat

Le résultat de l'analyse de l'application est mis dans un fichier ayant l'extension **.gqre** (glips qualité résultat). Ce fichier présente les erreurs selon deux classements:

1. par règle non respectée (voir arbre règles vérifiées). Seule la partie activée lors de l'exécution PLC Checker est affichée.
2. par rapport au programme vérifié. Chaque erreur est attachée à l'élément de l'atelier PL7 PRO que l'on devra vraisemblablement éditer afin de corriger l'erreur.

Le bouton droit de la souris permet de marquer une erreur comme étant corrigée (menu Marquer Corrigée). Le menu **Marquer A Corriger** permettant de revenir en arrière.

Une erreur non corrigée est précédée du caractère X. Une erreur marquée "corrigée" est précédée du caractère **V**.

10.4.1 Impression du fichier résultat

Pour imprimer le fichier de résultat ou inclure son contenu dans un document, vous devez tout d'abord ouvrir le fichier de résultat **.gqre** et l'enregistrer au format **.html**.

Ouvrir le fichier **.gqre**.

Le menu **Fichier** > **Sauvegarder sous. . .** fait apparaître la boîte de dialogue **Sauvegarder sous. . .**

Dans la zone **Nom du fichier**, changez l'extension du fichier de **.gqre** en **.html** (ATTENTION : respectez la casse), et cliquez sur **OK**. Le fichier **.html** est maintenant dans la fenêtre Navigateur.

Ouvrez ensuite le fichier **.html** avec l'éditeur système (menu contextuel **Ouvrir avec** > **Editeur système**) et utilisez sa fonction d'impression (menu **Fichier** > **Imprimer. . .**) ou son presse-papier (menu **Edition** > **Copier**)

Note : Les erreurs corrigées apparaissent barrées.

Chapter 11

Les règles

1. Groupe de règles (rulesGroup)
2. Vérification des lectures (readCheck)
3. Vérification des écritures (writeCheck)
4. Vérification des accès (accessCheck)
5. Vérification des symboles (symbolCheck)
6. Vérification des attributs (attributeCheck)
7. Vérification des blocs de code (blockCheck)
8. Vérification d'équation (equationCheck)
9. Vérification de mémoire (memoryCheck)
10. Vérification de copier/coller (copyPastCheck)
11. Vérification des structures dans le fichier source (FEFStructureCheck)
12. Vérification des attributs dans le fichier source (FEFAttributeCheck)
13. Vérification des labels (labelCheck)
14. Vérification de correspondance de modèle (patternMatchingCheck)
15. Duplication automatique de règles (duplicate)

11.1 Groupe de règles (rulesGroup)



Figure 11.1: rulesGroup

11.1.1 Utilisation

Cet élément a trois utilisations possibles:

1. Structuration du fichier de règles. Cet élément est uniquement là afin de regrouper des règles. Nous pouvons avoir un élément regroupant tous les filtres relatifs au code, un autre regroupant toutes les règles relatives à la lisibilité.

2. Partie d'un filtre réalisé grâce à l'intersection de ses ancêtres et de ses descendants.
3. Définition d'un filtre.

Cet élément sert à définir un filtre (code ou donnée) qui pourra être réutilisé à l'aide de son identificateur (propriété id).

11.1.2 Propriétés

Nom (name)

Voir Concepts - Règles - Nom de règles

Identificateur (id)

Voir Concepts - Règles - Identificateur de règles

Filtre de code (treatments)

Voir Concepts - Règles - Filtre de code

Filtre de données (variables)

Voir Concepts - Règles - Filtre de données

11.2 Vérification des lectures (readCheck)



Figure 11.2: readCheck

11.2.1 Utilisation

Cet élément permet de vérifier que les lectures effectuées dans tout ou partie du code (filtre de code) par tout ou partie des variables (filtre de données) ne sont ni trop nombreuses (propriété maxRead) ni trop peu nombreuses (propriété minRead).

11.2.2 Propriétés

Nom (name)

Voir Concepts - Règles - Nom de règles

Identificateur (id)

Voir Concepts - Règles - Identificateur de règles

Filtre de code (treatments)

Voir Concepts - Règles - Filtre de code

Filtre de données (variables)

Voir Concepts - Règles - Filtre de données

Nombre de lectures minimales (minRead)

Cette règle est le plus souvent utilisée avec un minRead égal à 1.

Dans ce cas, les variables du filtre de données qui ne sont pas lues dans le filtre de code sont ajoutées au fichier résultat.

Cette utilisation de la règle permet de vérifier la modularité de l'application.

Note : Si la propriété est laissée vide, alors aucune erreur n'est générée.

Nombre de lectures maximales (maxRead)

Cette règle est le plus souvent utilisée avec un maxRead égal à 0.

Dans ce cas, les variables du filtre de données qui sont lues dans le filtre de code sont ajoutées au fichier résultat.

Cette utilisation de la règle permet de vérifier la modularité de l'application.

Note : Si la propriété est laissée vide, alors aucune erreur n'est générée

Type de vérification (checkType)

Par défaut (defaultCheck), cette règle vérifie les nombres min et max de lecture définis ci-dessus (voir minRead et maxRead).

Mais il est aussi possible (readOnceVariableWrittenInOtherTaskCheck) de vérifier que les variables écrites par une autre tâche sont lues 0 ou 1 fois (mais pas plusieurs fois) dans la tâche vérifiée. Dans ce cas les paramètres minRead, maxRead ne sont pas utilisés.

11.3 Vérification des écritures (writeCheck)



Figure 11.3: writeCheck

11.3.1 Utilisation

Cet élément permet de vérifier que tout ou partie (filtre de code) des écritures de tout ou partie (filtre de données) des variables sont réalisées suivant le mode décrit par la propriété checkType.

11.3.2 Propriétés

Nom (name)

Voir Concepts - Règles - Nom de règles

Identificateur (id)

Voir Concepts - Règles - Identificateur de règles

Filtre de code (treatments)

Voir Concepts - Règles - Filtre de code

Filtre de données (variables)

Voir Concepts - Règles - Filtre de données

Type de vérification (checkType)

Écrite depuis une seule tâche (writtenFromOneTaskOnly) Cette règle vérifie que toutes les occurrences des écritures de chaque variable, appartenant au filtre de données (propriétés variables) trouvé dans le code compris dans le filtre de code (propriété treatments), ne peuvent être exécutées que depuis une seule tâche.

Note : Un programme PL7 PRO peut comporter une tâche MAST, une tâche FAST, et une tâche par évènement.

Cette règle est utile pour identifier les variables critiques à cause d'accès concurrents.

Écrite une et une seule fois (writtenOnce) Cette règle vérifie que toutes les variables appartenant au filtre de données (propriétés variables) ne sont écrites qu'une seule fois dans le code compris dans le filtre de code (propriété treatments).

Cette règle est utile afin de vérifier par exemple que les sorties sont l'objet d'une équation unique.

Set unique (setOnce) Cette règle vérifie que toutes les variables appartenant au filtre de données (propriétés variables) ne sont pas écrites par autre chose qu'un SET et ne sont pas écrites plusieurs fois, dans le code compris dans le filtre de code (propriété treatments).

Note : Seul le SET et le ":= true;" sont autorisés. De plus, la variable peut ne pas être écrite.

Reset unique (resetOnce) Cette règle vérifie que toutes les variables appartenant au filtre de données (propriétés variables) ne sont pas écrites par autre chose qu'un RESET et ne sont pas écrites plusieurs fois, dans le code compris dans le filtre de code (propriété treatments).

Note : Seul le RESET et le ":= false;" sont autorisés. De plus, la variable peut ne pas être écrite.

Écrite depuis un seul bloc de code (writtenFromOneBlocOnly) Cette règle vérifie que toutes les occurrences des écritures de chaque variable appartenant au filtre de données (propriété variables) trouvées dans le code compris dans le filtre de code (propriété treatments) se trouvent dans un seul bloc de code.

Exemple : Les blocs de code dans l'atelier PL7 PRO sont les sections, les SR et les évènements.

Cette règle est utile pour vérifier la modularité de l'application.

Écrite depuis une seule instruction (writtenFromOneInstrOnly) Cette règle vérifie que toutes les occurrences des écritures de chaque variable appartenant au filtre de données (propriété variables) trouvées dans le code compris dans le filtre de code (propriété treatments) se trouvent dans une seule instruction complexe (IF, CASE, FOR, WHILE). **Une variable écrite depuis différent blocs de code est exclue de la vérification.** Un message est émis si deux écritures de la variables se font dans des instructions différentes dont le seul parent est le bloc de code.

Exemple:

```
1 IF cond THEN
2   var := value;
3 END IF;
4 ....
5 IF othercond THEN
6   var := othervalue;
7 END IF;
```

Sur ce code la règle writtenFromOneInstrOnly va émettre un message car les deux écritures de la variable var se font dans deux instructions différentes rattachées à un même bloc de code.

```
1 IF cond THEN
2   var := value;
3 ELSIF othercond THEN
4   var := othervalue;
5 ELSE
6   var := defaultvalue;
7 END IF;
```

Sur ce code la règle writtenFromOneInstrOnly n'émettra aucun message car les différentes écritures de la variable var se font dans une seule instruction : un IF avec plusieurs cas.

Cette règle est utile pour vérifier la modularité de l'application.

Écrite avant d'être lue à chaque tour de cycle (writtenBeforeRead) Cette règle vérifie que toutes les variables appartenant au filtre de données (propriété variables) dans le code compris dans le filtre de code (propriété treatments) sont systématiquement écrites avant d'être lues.

Cette règle permet de vérifier la réactivité de l'application.

Écrite (writtenInBlock) Cette règle vérifie que toutes les variables appartenant au filtre de données (propriété variables) sont écrites au moins une fois dans le code compris dans le filtre de code (propriété treatments).

Cette règle permet de vérifier que toutes les variables de l'application sont utilisées.

Pas écrite (notWrittenInBlock) Cette règle vérifie que toutes les variables appartenant au filtre de données (propriété variables) ne sont pas écrites dans le code compris dans le filtre de code (propriété treatments).

Cette règle permet de vérifier la modularité de l'application.

Écrite un nombre de fois compris entre un min et un max (writtenCount) Cette règle vérifie que toutes les variables appartenant au filtre de données (propriété variables) sont écrites un nombre de fois supérieur à un Minimum et inférieur à un maximum dans le code compris dans le filtre de code (propriété treatments). Si le nuombre d'écriture n'est pas compris entre le min et le max, un message d'erreur est généré.

Les propriétés suivantes permettent de sélectionner le nombre d'occurences d'écritures à vérifier par cette règle :

- minWrite (Nombre minimum d'écritures)
- maxWrite (Nombre maximum d'écritures)
- minSet (Nombre minimum de Set)
- maxSet (Nombre maximum de Set)
- minReset (Nombre minimum de Reset)
- maxReset (Nombre maximum de Reset)

Note : Si une propriété est laissée vide, alors aucune erreur n'est générée.

Cette règle permet de vérifier la modularité de l'application.

Exemple : Vérifier que chaque variable :

- est écrite par un SET une et une seule fois
- ET est écrite par un RESET une et une seule fois
- ET n'est jamais écrite d'un autre façon (affectation).

Pour cela, les propriétés sont renseignées comme suit :

- minSet = 1
- maxSet = 1
- minReset = 1
- maxReset = 1
- minWrite = 2 (correspond à la somme du nombre de set et de reset et d'affectation).
- maxWrite = 2 (correspond à la somme du nombre de set et de reset et d'affectation).

11.4 Vérification des accès (accessCheck)



Figure 11.4: acessCheck

11.4.1 Utilisation

Cette règle permet de s'assurer que les variables de l'application ont un accès ou pas (en écriture, en lecture ou en exécution). L'accès peut être partiel ou Total.

11.4.2 Propriétés

Nom (name)

Voir Concepts - Règles - Nom de règles

Identificateur (id)

Voir Concepts - Règles - Identificateur de règles

Filtre de code (treatments)

Voir Concepts - Règles - Filtre de code

Filtre de données (variables)

Voir Concepts - Règles - Filtre de données

Type de vérification (checkType)

Cette propriété permet d'indiquer le type de vérification qu'on va effectuer.

1. partial : Une vérification partielle (vérification sur au moins un bit de la variable)
2. total : Une vérification totale (vérification sur la variable entière)

Type d'accès (accessType)

Cette propriété permet d'indiquer le type d'accès. Ce type d'accès peut être en lecture (read), en écriture (write) ou en exécution (exec). Pour tester plusieurs type de vérification en même temps, il suffit de séparer les types d'accès par une virgule dans le fichier de règles. Exemple : pour tester qu'une variable est lue et écrite : accessType ="read,write"

Le nombre d'accès minimum (minAccess)

C'est un "entier" qui présente le nombre minimum d'accès à une variable. Exemple : un minimum d'accès = 1 avec un accesType = read et un checkType = total, va vérifier que la variable est lue au moins une fois. Si le champs est vide, la règle ne fait pas la vérification par rapport à un minAccess.

Le nombre d'accès maximum (maxAccess)

C'est un "entier" qui présente le nombre maximum d'accès à une variable. Exemple : un maximum d'accès = 1 avec un accesType = write et un checkType = total, va vérifier que la variable est écrite au maximum une fois dans l'application. Si le champs est vide, la règle ne fait pas la vérification par rapport à un maxAccess.

11.5 Vérification des symboles (symbolCheck)



Figure 11.5: symbolCheck

11.5.1 Utilisation

Cette règle permet de s'assurer que les objets (code et données) de l'application ont tous un symbole, que ce symbole ne sera jamais tronqué, qu'il ne contient pas d'information sur son adresse physique et qu'un ensemble d'objets est bien dans une zone mémoire donnée.

11.5.2 Propriétés

Nom (name)

Voir Concepts - Règles - Nom de règles

Identificateur (id)

Voir Concepts - Règles - Identificateur de règles

Filtre de code (treatments)

Voir Concepts - Règles - Filtre de code

Filtre de données (variables)

Voir Concepts - Règles - Filtre de données

Longueur minimale des symboles (minLength)

Cette propriété permet d'indiquer pour tous les éléments de l'application (filtres de code et de données) la longueur minimale que doit avoir le symbole associé.

Longueur maximale des symboles (maxLength)

Cette propriété permet d'indiquer pour tous les éléments de l'application (filtres de code et de données) la longueur maximale que doit avoir le symbole.

Le symbole ne comporte pas d'information relative à son adresse mémoire (notAbsoluteCheck)

Ce booléen permet d'isoler parmi le code et les données (filtre de code et de données) les symboles qui ont dans leur nom une référence soit à leur adresse absolue, soit à leur type.

Exemples :

- W_10_VITESSE_TAPIS à l'adresse %MW10 génère une erreur (référence à 10).
- MW_VITESSE_TAPIS à l'adresse %MW10 génère une erreur (référence à MW).
- AMBIANCE à l'adresse %MB10 génère une erreur (référence à MB[1]).
- AMBIANCE à l'adresse %M10 ne génère pas d'erreur (pas de test de référence au type représenté par un seul caractère)
- PRESSION_POMPE_101 à l'adresse %M10 ne génère pas d'erreur (10 est comparé à 101)

[1] Megabyte

Les variables appartiennent à une zone mémoire (memFilter)

Ce test permet de vérifier que toutes les variables, appartenant au sous-ensemble défini par la propriété variables, appartiennent au sous-ensemble défini par la propriété memFilter.

Cette règle est utilisée pour vérifier par exemple que les télécommandes sont toutes dans la zone attendue.

11.6 Vérification des attributs (attributeCheck)



Figure 11.6: attributeCheck

11.6.1 Utilisation

Cette règle permet de vérifier la valeur d'un attribut donné (propriété attribut) d'un ensemble d'éléments de l'application (voir sélection des types d'éléments ci-dessous).

La liste des attributs disponibles dépend du type d'élément et de l'atelier de programmation des API[2] (Unity-Pro, PL7-Pro, Step7, RSLogix5000).

11.6.2 Propriétés

Nom (name)

Voir Concepts - Règles - Nom de règles

Identificateur (id)

Voir Concepts - Règles - Identificateur de règles

Filtre de code (treatments)

Voir Concepts - Règles - Filtre de code

Filtre de données (variables)

Voir Concepts - Règles - Filtre de données

Longueur minimale de l'attribut (minLength)

Cette propriété permet d'indiquer pour tous les éléments de l'application (filtres de code et de données et types d'éléments vérifiés) auxquels la vérification s'applique, la longueur minimale que doit avoir l'attribut associé afin qu'une erreur ne soit pas générée.

Longueur maximale de l'attribut (maxLength)

Cette propriété permet d'indiquer pour tous les éléments de l'application (filtres de code et de données et types d'éléments vérifiés) auxquels la vérification s'applique, la longueur maximale que peut avoir l'attribut associé afin qu'une erreur ne soit pas générée.

Nom de l'attribut (attribute)

Cette propriété défini le nom de l'attribut dont la valeur sera vérifiée. Une liste non-exhaustive d'attributs utilisables est disponible ici :Attributs Glips

[2] Application Programming Interface

Expression (expression)

Cette propriété défini l'expression que la valeur de l'attribut doit satisfaire pour qu'une erreur ne soit pas générée.

Sélection des types d'éléments

Les propriétés suivantes permettent de sélectionner, lorsqu'elles valent "True", les types d'éléments à vérifier par cette règle :

- Instruction
- typeDeclaration
- typeDesignation
- variableDeclaration
- variableDesignation
- prototypeDeclaration
- prototypeDesignation
- grafcetTransitionIncluded
- program
- grafcetStepIncluded
- label
- procedureDeclaration
- procedureDesignation
- instructionList
- rung
- operator

11.7 Vérification des blocs de code (blockCheck)



Figure 11.7: blockCheck

11.7.1 Utilisation

Cet ensemble de règles (selon le type de vérification fixé par la propriété checkType) permet de travailler au niveau des blocs de code. Pour PL7 PRO les deux types de bloc de code sont les sections et les SR.

11.7.2 Propriétés

Nom (name)

Voir Concepts - Règles - Nom de règles

Identificateur (id)

Voir Concepts - Règles - Identificateur de règles

Filtre de code (treatments)

Voir Concepts - Règles - Filtre de code

Filtre de données (variables)

Voir Concepts - Règles - Filtre de données

Type de vérification (checkType)

Le bloc contient du code (containsCode) Ce type de vérification s'assure que tous les blocs de code définis par la propriété treatments contiennent au moins une instruction.

Le bloc de code est présent(present) Ce type de vérification s'assure qu'au moins un bloc de code défini par la propriété treatments est présent dans l'application. Ceci permet de s'assurer que l'application suit un schéma de conception donné.

Le bloc de code n'est pas présent (notPresent) Ce type de vérification s'assure qu'aucun bloc de code défini par la propriété treatments n'est présent dans l'application. Ceci permet de s'assurer par exemple que le code temporaire (mise au point, simulation, validation) a bien été enlevé.

Le bloc contient du code jamais exécuté (containsDeadCode) Ce type de vérification s'assure que toutes les instructions de tous les blocs de code définis par la propriété treatments peuvent être exécutées. L'ensemble des lignes non accessibles est ajouté au fichier résultat. Cela permet de trouver des problèmes du type (le code en question est inutile). Cela permet aussi en maintenance de ne perdre son temps sur de fausses informations.

11.8 Vérification d'équation (equationCheck)



Figure 11.8: equationCheck

11.8.1 Utilisation

Cette règle fonctionne sur les références en écritures des variables. Elle permet de vérifier si au moins une référence en lecture d'une variable X (voir filtre var) est utilisée dans chaque instruction qui écrit la variable Y (voir filtre variables).

Par exemple, il est possible de vérifier que toutes les écritures des sorties (S_*) dépendent d'une variable d'autorisation (aut_*). ie: s_* = f(aut_*,...)

11.8.2 Propriétés

Nom (name)

Voir Concepts - Règles - Nom de règles

Identificateur (id)

Voir Concepts - Règles - Identificateur de règles

Filtre de code (treatments)

Voir Concepts - Règles - Filtre de code

Filtre de données (variables)

Voir Concepts - Règles - Filtre de données
Cela permet de sélectionner les variables dont les références en écriture seront vérifiées.

Filtre de variable utilisée (var)

Ce filtre (var) permet de sélectionner les variables dont les références en lectures seront recherchées dans les références en écritures des variables définies par le précédent filtre (variables).

Type de vérification (checkType)

Cet attribut permet de sélectionner le type de vérification que la règle doit effectuer.

Structure d'équation (checkType non défini) Si le checkType n'est pas défini, alors la vérification par défaut est faite à savoir : vérification de la structure d'une équation : présence d'une classe de variable (définie par l'attribut var) lorsqu'une variable particulière (définie par l'attribut variables) est élaborée.

Recherche des valeurs immédiates (noImmediateValueCheck) Si checkType="noImmediateValueCheck", alors la règle remonte des erreurs lorsqu'elle trouve des valeurs immédiates (ex: 12, 56.48, "bonjour", ...). Note : les valeurs immédiates présentent dans les initialisations de variable ou dans les indices de bits de mots sont ignorées lors de la vérification. De même, les valeurs immédiates très fréquentes (TRUE, FALSE, 1, 0, -1) ne sont pas remontées en erreurs.

Recherche stricte des valeurs immédiates (noImmediateValueStrictCheck) Si checkType="noImmediateValueStrictCheck", alors la règle remonte des erreurs lorsqu'elle trouve des valeurs immédiates (ex: 12, 56.48, "bonjour", ...), y compris les valeurs TRUE, FALSE, -1, 0 et 1. Note : les valeurs immédiates présentent dans les initialisations de variable ou dans les indices de bits de mots sont ignorées lors de la vérification.

Recherche des tests d'égalité entre flottants (noFloatEqualityCheck) Si checkType="noFloatEqualityCheck", alors la règle remonte des erreurs lorsqu'elle trouve des opérateurs de test d'égalité (ou de différence) sur un flottant (ex: if my_float = 12,34 then ...)

Recherche des tests d'égalité entre entiers (noIntegerEqualityCheck) Si checkType="noIntegerEqualityCheck", alors la règle remonte des erreurs lorsqu'elle trouve des opérateurs de test d'égalité (ou de différence) sur un entier (ex: if my_int = 12 then ...)

Recherche des comparaison entre pointeurs (noPointerComparisonCheck)

- Si checkType="noPointerComparisonCheck", alors la règle remonte des erreurs lorsqu'elle trouve des opérateurs de comparaison (=, /=, <, <=, >, >=) sur un pointeur (ex: if my_pointer > another_pointer then ...)

Vérification des parenthèses (parenthesisCheck) ??? **TODO** ???

11.9 Vérification de mémoire (memoryCheck)



Figure 11.9: memoryCheck

11.9.1 Utilisation

Cette règle fonctionne sur les emplacements des éléments appartenant à un plan mémoire. Elle permet de vérifier si des déclarations de variables se chevauchent (overlapStrict), ou si des variables écrites se chevauchent (overlapLazzy), ou si un plan mémoire (défini par le filtre memFilter) contient au moins un pourcentage donné d'emplacements libres (voir paramètre free).

11.9.2 Propriétés

Nom (name)

Voir Concepts - Règles - Nom de règles

Identificateur (id)

Voir Concepts - Règles - Identificateur de règles

Filtre de code (treatments)

Voir Concepts - Règles - Filtre de code

Filtre de données (variables)

Voir Concepts - Règles - Filtre de données

Filtre de plan mémoire (memFilter)

Ce filtre (memFilter) permet de sélectionner le(s) plan(s) mémoire testé(s).

Paramètre d'emplacement libre (free)

Ce paramètre défini le taux minimum d'emplacement libre désiré dans chaque plan mémoire sélectionné (voir memFilter). ex: free="10%" si l'on désire qu'il y ai au moins 10% d'emplacement libre pour chaque plan mémoire.

Type de vérification (checkType)

Chevauchement de déclaration (overlapStrict) Ce type de vérification s'assure qu'aucune déclaration d'élément (généralement des variables) ne chevauche une autre. Les adresses, les offset et les tailles des éléments sont pris en compte pour cette vérification.

Chevauchement de déclaration en écriture (overlapLazzy) Ce type de vérification s'assure qu'aucune déclaration de variable écrite ne chevauche une autre variable écrite. Les adresses, les offset et les tailles des éléments sont pris en compte pour cette vérification. Les déclarations de variables qui se chevauchent sont autorisées si une seule variable est écrite. Cela permet d'autoriser l'écriture d'un mot d'état et la lecture des bits de ce mot d'état, même si le mot et ses bits de mots se chevauchent.

Taux d'utilisation du plan mémoire (available) Ce type de vérification s'assure qu'au moins un certain pourcentage du plan mémoire est libre (paramètre free).

11.10 Vérification de copier/coller (copyPastCheck)

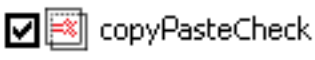


Figure 11.10: copypastecheck

11.10.1 Utilisation

Cette règle mesure un taux de copier/coller sur l'ensemble de l'application.

11.10.2 Propriétés

Nom (name)

Voir Concepts - Règles - Nom de règles

Identificateur (id)

Voir Concepts - Règles - Identificateur de règles

Filtre de code (treatments)

Voir Concepts - Règles - Filtre de code

Filtre de données (variables)

Voir Concepts - Règles - Filtre de données

Taille du copier/coller (tokenLength)

Ce paramètre permet d'ajuster le nombre d'éléments dans les ensembles à tester pour la détection de copie. Un nombre trop petit risque de détecter beaucoup de copier/coller. Un nombre trop grand risque de ne pas en détecter. Une taille de 100 éléments est généralement utilisée. Exemple : dans l'ensemble "Total := a + rest ;" il y a 6 éléments qui sont : "Total", ":=", "a", "+", "rest" et ";"

Type de vérification (checkType)

Détection de copie stricte (cpd_symbols) Ce type de vérification détecte une copie si les structures ET les mnémoniques des deux ensembles sont identiques. Exemple : "a := b + c ;" est une copie de "a := b + c ;".

Détection de copie de structure (cpd_no_symbols) Ce type de vérification détecte une copie si les structures des deux ensembles sont identiques. Exemple : "e := f + g ;" est une copie de "a := b + c ;".

11.11 Vérification des structures dans le fichier source (FEFStructureCheck)

11.11.1 Utilisation

[Règle obsolète] Anciennement utilisée dans les premières applications PL7pro pour vérifier la structure des sections directement dans le ficher d'export FEF, sans transition dans l'arbre GLIPS.

11.11.2 Propriétés

Nom (name)

Voir Concepts - Règles - Nom de règles

Identificateur (id)

Voir Concepts - Règles - Identificateur de règles

Filtre de code (treatments)

Voir Concepts - Règles - Filtre de code

Filtre de données (variables)

Voir Concepts - Règles - Filtre de données

TODO

11.12 Vérification des attributs dans le fichier source (FEFAttributeCheck)



Figure 11.11: fefAttributeCheck

11.12.1 Utilisation

Seulement pour PL7pro ou application UnityPro: Cette règle permet de rechercher, directement dans le fichier source d'export et sans passer par l'arbre GLIPS, pour un attribut donné (checkedAttributeName) d'un type d'objet donné (fefObjectName) et pour vérifier la valeur de cet attribut (voir checkedAttributevalue et checkType properties). Ceci permet notamment de tester un attribut qui n'a pas encore été traduit dans l'arbre GLIPS.

11.12.2 Propriétés

Nom (name)

Voir Concepts - Règles - Nom de règles

Identificateur (id)

Voir Concepts - Règles - Identificateur de règles

Filtre de code (treatments)

Voir Concepts - Règles - Filtre de code

Filtre de données (variables)

Voir Concepts - Règles - Filtre de données

Sélection de type d'objet (fefObjectType)

Cette propriété définie le type d'objet sur lequel l'attribut sera cherché (ex: "languageExtension" qui est un noeud xml dans un fichier d'export XEF)

Nom de l'attribut à vérifier (checkedAttributeName)

Cette propriété défini le nom de l'attribut à chercher sur chaque objet dont le type est défini par la propriété fefObject-Type. (ex: "multiToken" qui est un attribut de l'object "languageExtension", voir ci-dessus).

Valeur de l'attribut à vérifier (checkedAttributeValue)

Cette propriété définie la valeur à comparer avec celle de l'attribut trouvé (voir checkedAttributeName), si il y en a une.

Type de vérification

Cette propriété définie si la valeur de l'attribut doit être égale (Equal) ou différente (notEqual) à celle de la valeur définie dans la règle (checkedAttributeValue).

Nom d'attribut pour filtrage (filterAttributeName)

Cette propriété définie le nom de l'attribut à vérifier sur chaque objet dont le type est celui défini par la propriété fefObjectType, pour voir si l'objet doit être vérifié ou non. Si le type d'objet est peu précis, il est conseillé de filtrer un sous-ensemble.

Valeur d'attribut pour filtrage (filterAttributeValue)

Cette propriété définie la valeur à comparer à celle trouvée par le filtrage par nom (filterAttributName), si il y en a un.

Type de filtrage (filterType)

Cette propriété définie si la valeur de l'attribut doit être égale (Equal) ou différente (notEqual) à celle de la valeur définie dans la règle (filterAttributeValue).

11.13 Vérification des labels (labelCheck)

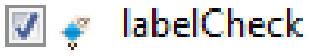


Figure 11.12: labelCheck

11.13.1 Utilisation

Cette règle permet de faire différentes sortes de vérifications (voir propriété checkType) sur les labels d'un ensemble de blocs de code.
Les vérifications sur la croissance, la continuité, le bornage des labels concernent les labels se terminant par un nombre (ex: L251).

11.13.2 Propriétés

Nom (name)

Voir Concepts - Règles - Nom de règles

Identificateur (id)

Voir Concepts - Règles - Identificateur de règles

Filtre de code (treatments)

Voir Concepts - Règles - Filtre de code

Filtre de données (variables)

Voir Concepts - Règles - Filtre de données

Type de vérification (checkType)

Le label est absent (absence) Ce type de vérification s'assure que tous les labels, dont le nom est décrit par la propriété variables, sont absents de tous les blocs de code définis par la propriété treatments.

Le label est présent (presence) Ce type de vérification s'assure que tous les labels, dont le nom est décrit par la propriété variables, sont présents dans au moins un des blocs de code définis par la propriété treatments.

Les labels sont continus (continuity) Ce type de vérification s'assure que tous les labels sont continus (différence avec précédent = 0 ou 1) dans les blocs de code définis par la propriété treatments.

Les labels sont croissants (increasing) Ce type de vérification s'assure que tous les labels sont croissants (différence avec précédent >= 1) dans les blocs de code définis par la propriété treatments.

Les labels sont bornés (boundary) Ce type de vérification s'assure que tous les labels sont compris (bornes exclues) entre une borne min (propriété minAddr) et une borne max (propriété maxAddr) dans les blocs de code définis par la propriété treatments.

Les labels sont bornés (boundaryIncluded) Ce type de vérification s'assure que tous les labels sont compris (bornes inclues) entre une borne min (propriété minAddr) et une borne max (propriété maxAddr) dans les blocs de code définis par la propriété treatments.

Les labels sont uniques (unicity) Ce type de vérification s'assure que tous les labels sont uniques, dans les blocs de code définis par la propriété treatments.

11.14 Vérification de correspondance de modèle (patternMatchingCheck)

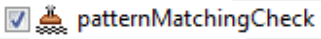


Figure 11.13: patternMatchingCheck

11.14.1 Utilisation

Cette règle permet de vérifier la présence ou l'absence de modèle de code dans un ensemble de blocs de code.
Il est possible de décrire dans le modèle, une ou plusieurs instructions dans un champ texte en fils de la règle. Ce modèle peut être éditer si l'attribut isTextEditable est positionné à true (voir propriete_istexteditable).

11.14.2 Propriétés

Nom (name)

Voir Concepts - Règles - Nom de règles

Identificateur (id)

Voir Concepts - Règles - Identificateur de règles

Filtre de code (treatments)

Voir Concepts - Règles - Filtre de code

Filtre de données (variables)

Voir Concepts - Règles - Filtre de données

Type de vérification (checkType)

Présence d'instructions (instructionPresence) Cette règle vérifie qu'au moins une occurence du modèle de code est présente dans le code compris dans le filtre de code (propriété treatments).

Absence d'instructions (instructionAbsence) Cette règle vérifie qu'aucune occurence du modèle de code n'est présente dans le code compris dans le filtre de code (propriété treatments).

Modèle de code

Ce champ de texte, en fils XML de la règle, contient, en langage glips (langage structuré), le code décrivant le modèle à vérifier. Ce modèle peut contenir des jokers pour laisser de la latitude sur la forme du code testé.

Joker pour les expressions Le joker pour les expressions est : $. . . $

- entre les deux, on trouve les filtres PLC Checker (filtre de texte, filtre d'attribut, filtre d'adresse, filtre d'attribut)
- Si c'est vide ça accepte tout

Par exemple :

- $$ ⇔ N'importe quelle expression
- $+toto$ ⇔ N'importe quelle entité/désignateur pointant sur une entité nommée toto.
- $+@M:800$ ⇔ La variable qui se trouve dans le plan mémoire M à l'adresse 800.
- $+#VARS_RESERVE$ ⇔ n'import quel élément satisfaisant au filtre variables du rulesGroup dont l'id est VARS_RESERVE.
- $+(test.+)-!comment=(-(demo.*))$ ⇔ N'importe quelle entité/désignateur pointant sur une entité nommée test* et dont le commentaire commence par demo.

Joker pour les instructions L'identifiant Joker pour une instruction est : [ ]

- [ ] contient la description de ce qu'on veut
- contient la description de combien de fois on le veut : voir les expressions régulières pour la syntaxe :
  - n n fois exactement
  - n, au moins n fois
  - n,m entre n et m fois

Par exemple :

- [$+out_*$:= $$;]3 ⇔ 3 affectations consécutives de variables nommées out_*.
- [$$]5,9 ⇔ entre 5 et 9 instructions.

11.15 Duplication automatique de règles (duplicate)



Figure 11.14: duplicate

Ce mécanisme utilise l'ajout d'un nœud <duplicate/> dans le fichier de règles pour dupliquer automatiquement des règles, des groupes de règles ou des chaines de caractères.

11.15.1 Lancement de la duplication

Les nœuds duplicate sont interprétés par le PLC Checker lors de l'analyse (et pas par l'éditeur de règle de la plateforme Eclipse avant de lancer l'analyse). Le fichier de règle (*.gqr) ne sera donc pas modifié s'il contient un ou plusieurs nœuds <duplicate/>.

11.15.2 Définitions

Un nœud duplicate sert à définir une liste de valeurs à utiliser (ex: A, B, C). La taille de cette liste définit le nombre de duplication (ex : ici 3). Une valeur a utilisée peut-être constituée de plusieurs champs (ex: (A,titi), (B,toto), (C,tata)).

Les différentes façons de créer ces listes et nombre de duplication sont listés dans les cas de duplication.

Un nœud duplicate doit être défini avant d'être utilisé.

Un même nœud duplicate peut être utilisé pour faire différentes choses (voir utilisations des duplications).

L'utilisation d'un nœud duplicate nécessite un PATTERN qui sera remplacé selon les valeurs à utiliser définies dans ce nœud duplicate.

11.15.3 Syntaxe

Le nœud duplicate comporte un certain nombre d'attributs (id, filter, catch, initial, increment, count, alphanumeric, values, filepath, conditions).

L'attribut id contient un identifiant (par convention une chaîne du type [A-Z][A-Z_0-9]*). ex : id="ID_VALUE"

Les autres attributs servent selon les cas de duplication.

Pour chaque utilisation d'un nœud duplicate prédéfini, il faut :

1. que le nœud dans lequel il doit être utilisé contienne un attribut dynamic="true"
2. définir le PATTERN à remplacer. ex : %ID_VALUE%
3. et l'emplacement du PATTERN dans les attributs de ce nœud. ex: monAttribut="...%ID_VALUE%..."

La syntaxe du PATTERN défini son type d'utilisation (duplication de règle, concaténation de chaines) et son contenu. Dans le cas où une valeur contient plusieurs champs disponibles, le numéro du champ à utiliser est indiqué dans le PATTERN après l'id du nœud duplicate. (ex: %ID_VALUE.2% pour le 2ème champ des différentes valeurs définies par le nœud duplicate dont l'id est "ID_VALUE").

11.15.4 Fonctionnement

Cas de duplication

Un cas de duplication défini la liste de valeurs que prendra le PATTERN à remplacer et donc le nombre de duplication à effectuer. ex: un cas de duplication avec les 3 valeurs A, B et C.
Il existe plusieurs cas de duplication :

- SOIT selon la détection d'éléments dans le programme : le nombre de duplication et les différentes valeurs à remplacer seront alors définis par
  - l'expression régulière (ex: "+[(moteur|vanne)([0-9]+)]") contenue dans le filtre (attribut **filter**),
  - le groupe (ex: "N" pour le N-ème groupe) de caractères, ou la combinaison ("N,M" ou "NxM") de groupes de caractères identifié(e) dans l'expression régulière (attribut **catch**) et
  - les différentes valeurs de ces groupes de caractères trouvées dans les différents éléments présents dans le programme et répondant au filtre.
- SOIT selon une liste prédéfinie :
  - Les valeurs sont listées dans l'attribut **value** et séparées par un espace (ex: value="val1 val2 val3 ... valn")
- SOIT selon une incrémentation : Les valeurs sont calculées
  - en commençant par une valeur entière initiale (attribut **initial**) (ex: initial="10")
  - et, pour chaque nouvelle valeur, en ajoutant un incrément (attribut **increment**) à la valeur précédente, (ex increment="5")
  - jusqu'à obtenir le nombre désiré de valeurs (attribut **count**) (ex: count="6")
  - et en comptant :

    * en base décimale (attribut **alphanumeric**="false") (ex: initial="10" increment="5" count="6" alphanumeric="false", génèrera les valeurs 10, 15, 20, 25, 30, 35)
    * ou en base hexadécimale (attribut **alphanumeric**="true") (ex: initial="10" increment="5" count="6" alphanumeric="true", génèrera les valeurs 10, 15, 1A, 1F, 24, 29)

- SOIT selon le contenu d'un fichier externe :
  - le fichier à utiliser est identifié par l'attribut **filepath** (ex: file="mon_fichier.csv")
  - le nombre de duplication correspond au nombre de lignes satisfaisant aux conditions définies dans l'attribut **conditions**
    * Les conditions sont exprimées en précisant le numéro de la colonne (ex: 1, 2, 3, ...) et le filtre de texte (simple ou expression régulière) à appliquer pour savoir si la ligne est sélectionnée ou pas
    * Il est possible de combiner plusieurs conditions avec les opérateurs AND OR et NOT et une notation polonaise inversée
    * exemple 1 : conditions="(4,+X) (5,-Y*) AND", ne sélectionnera que les lignes dont la colonne 4 contient la chaine de caractères X (ou x car pas de sensibilité à la case) ET dont la colonne 5 contient une chaine de caractères NE commençant PAS par Y (ou y car pas de sensibilité à la case)
    * exemple 2 : conditions="(4,+(F(B|C)_.+)) (5,-(STD[0-9]+_.+)) OR NOT", NE sélectionnera PAS les lignes dont la colonne 4 contient une chaine de caractères commençant par FB_ ou FC_ OU dont la colonne 5 contient une chaine de caractères NE commençant PAS par STDxx_ (avec xx correspondant a un ou plusieurs chiffres)
  - les différentes valeurs à remplacer dépendent alors du contenu des lignes sélectionnées. Pour le remplacement, il est possible de n'utiliser qu'une colonne (ex: PATTERN.3 pour la 3ème colonne) et pas la ligne entière (selon le type de fichier).

Choix du cas

Pour un nœud duplicate donné, le choix entre un cas de duplication ou un autre dépend des attributs définis dans ce nœud duplicate.
La priorité est donnée :

1. à la duplication selon la détection d'éléments dans le programme (si l'attribut **filter** est présent)
2. sinon à celle selon une liste prédéfinie (si l'attribut **value** est présent)
3. sinon à celle selon une incrémentation (si l'attribut **increment** est présent)
4. sinon à celle selon le contenu d'un fichier externe (si l'attribut **filepath** est présent)
5. sinon aucune duplication n'est effectuée.

Utilisation des duplications

Un même cas de duplication (nombre + valeurs) peut être utilisé pour :

1. dupliquer des nœuds XML de l'arbre du fichier de règles. Donc pour dupliquer des règles (*Check) ou des groupes de règles (rulesGroup). Ainsi pour chaque valeur de la liste de duplication, la règle à dupliquer sera dupliquée et chaque PATTERN relatif à cette duplication sera remplacé par l'un des champs de la valeur à utiliser. voir les exemples de [[#duplication_de_noeud_xml|duplication de noeuds].
2. ET/OU remplacer une chaine de caractères par une concaténation des différentes valeurs du cas de duplication. voir les exemples de [[#concatenation_de_chaine|concaténation de chaines].

La distinction entre ces utilisations se fait par le format du PATTERN :

1. %ID% ou %ID.12% : pour les duplication de noeud
2. %ID$pref$suf% ou %ID.12$pref$suf% pour les concaténation de chaines

PRINCIPE DE FONCTIONNEMENT

On visite les nœuds duplicate dans l'ordre d'apparition dans le fichier de règles. Pour chaque nœud duplicate trouvé et dont l'id est PATTERN, on revisite les nœuds dans l'ordre d'apparition dans le fichier de règles (les pères sont visités avant les fils).

1. **CAS DE LA DUPLICATION DE NOEUD :**
   Si ce nœud a un attribut nommé dynamic="true" et qu'un des attributs de ce nœud contient le pattern %PATTERN%

alors, pour chaque valeur de la liste de valeurs, ce nœud et ses fils sont clonés et chaque chaîne %PATTERN% de ce sous-arbre clonée est remplacée par la valeur courante.
Le nombre de duplications du sous-arbre sera fonction de la longueur de la liste de valeurs du nœud duplicate concerné.

1. **CAS DE LA CONCATENATION DE CHAINES :**
   Si ce nœud a un attribut nommé dynamic="true" et qu'un des attributs de ce nœud contient le pattern %PATTERNprsu%

alors, ce pattern sera remplacé par la concaténation de toutes les valeurs de la liste de valeurs, chacune étant préfixée par le préfix "pr" et suffixée par le suffix "su".
Le nombre de valeurs concaténées sera fonction de la longueur de la liste de valeurs du nœud duplicate concerné.

Dans le cas d'une duplication, selon la détection d'éléments dans le programme, pour laquelle on utilise une combinaison ("1,3" ou "1×3") de groupes de caractères, et pas un unique groupe de caractères donné ("1" ou "3"), la liste de valeur est en fait une liste d'ensembles de valeurs. En effet, à une duplication correspond un ensemble composé de la valeur de chaque groupe de caractères. Dans ce cas, il faut choisir de quel groupe on utilisera la valeur pour remplacer le pattern. Cela se fait en rajoutant après le pattern, le rang du groupe dans l'ensemble de valeur (ex: %PATTERN.2% pour utiliser la 2ème valeur dans l'ensemble de valeur de cette duplication). De manière générale, la chaîne %PATTERN.x% est remplacée par le contenu du x-ème champ d'un ensemble de substitution.

Ainsi pour utilisée la 3ème colonne d'un fichier CSV, on utilisera un duplicate de type file="..." et un pattern %ID.3% (pour un duplication de noeud) ou %ID.3$$% (pour une concaténation de chaines sans préfixe ni suffixe)

Dans tous les cas, %PATTERN% et %PATTERN.1% sont remplacés par la même valeur.

11.15.5 Exemples

Duplication de noeud xml

- Duplication incrémentale:

```
<duplicate id="PATTERN1" initial="15" increment="2" count="8"
alphanumeric="false"/>
<element dynamic="true" variables="+moteur%PATTERN1%" />
```

donnera après clonage :

```
<duplicate id="PATTERN1" initial="1" increment="2" count="8"
alphanumeric="false"/>
<element dynamic="true" variables="+moteur15" />
<element dynamic="true" variables="+moteur17" />
<element dynamic="true" variables="+moteur19" />
<element dynamic="true" variables="+moteur21" />
<element dynamic="true" variables="+moteur23" />
<element dynamic="true" variables="+moteur25" />
<element dynamic="true" variables="+moteur27" />
<element dynamic="true" variables="+moteur29" />
```

- Duplication incrémentale:

```
<duplicate id="PATTERN1" initial="15" increment="2" count="8"
alphanumeric="true"/>
<element dynamic="true" variables="+moteur%PATTERN1%" />
```

donnera après clonage :

```
<duplicate id="PATTERN1" initial="1" increment="2" count="8"
alphanumeric="true"/>
<element dynamic="true" variables="+moteur15" />
<element dynamic="true" variables="+moteur17" />
<element dynamic="true" variables="+moteur19" />
<element dynamic="true" variables="+moteur1B" />
<element dynamic="true" variables="+moteur1D" />
<element dynamic="true" variables="+moteur1F" />
<element dynamic="true" variables="+moteur21" />
<element dynamic="true" variables="+moteur23" />
```

- Duplication incrémentale:

```
<duplicate id="PATTERN1" initial="A000" increment="F" count="4"
alphanumeric="true"/>
<element dynamic="true" variables="+moteur%PATTERN1%" />
```

donnera après clonage :

```
<duplicate id="PATTERN1" initial="A000" increment="F" count="4"
alphanumeric="true"/>
<element dynamic="true" variables="+moteurA000" />
<element dynamic="true" variables="+moteurA00F" />
<element dynamic="true" variables="+moteurA01E" />
<element dynamic="true" variables="+moteurA02D" />
```

- Duplication selon une liste

```
<duplicate id="PATTERN2" values="FC_OPEN FC_READ FC_CLOSE"/>
<element dynamic="true" treatments="+%PATTERN2%" />
```

donnera après clonage :

```
<duplicate id="PATTERN2" values="FC_OPEN FC_READ FC_CLOSE"/>
<element dynamic="true" treatments="+FC_OPEN" />
<element dynamic="true" treatments="+FC_READ" />
<element dynamic="true" treatments="+FC_CLOSE" />
```

- Duplication selon détection d'éléments dans le programme, **SANS** combinaison de groupes de caractères :

```
<rulesGroup variables="+motor*">
  <duplicate id="PATTERN3" filter="+(moteur([0-9]+))" catch="1"/>
</rulesroup>
<element dynamic="true" variables="+moteur%PATTERN3%" name="test mot. %PATTERN3%"/>
```

sera cloné de la façon suivante, si les éléments moteur2, moteur34 et moteurGT existent dans le programme :

```
<rulesGroup variables="+motor*">
  <duplicate id="PATTERN3" filter="+(moteur([0-9]+))" catch="1"/>
</rulesroup>
<element dynamic="true" variables="+moteur2" name="test mot. 2"/>
<element dynamic="true" variables="+moteur34" name="test mot. 34"/>
```

notons que le clonage n'a pas eu lieu pour moteurGT car il ne correspond pas au filtre moteur([0-9]+) de l'élément duplicate.

- Duplications selon détection d'éléments dans le programme, **AVEC** différentes combinaison de groupes de caractères : Nous avons un filtre défini nommé IG avec ses duplicate :

```
<rulesGroup id="IG" variables="+(i*_g*)" ...>
  <duplicate id="IGN" filter="+(i(.*)_g(.*))" catch="1,2" />
  <duplicate id="IN" filter="+(i(.*)_g(.*))" catch="1" />
  <duplicate id="GN" filter="+(i(.*)_g(.*))" catch="2" />
  <duplicate id="IGXN" filter="+(i(.*)_g(.*))" catch="1x2" />
<element dynamic="true" variables="+i%PATTERN3%" name="test mot. %PATTERN3%"/>
```

Supposons que l'application contienne les variables suivantes : i1_g4, i1_g3, i2_g4. Alors chacun des quatre premiers duplicate va utiliser la même liste d'entités (en l'occurrence les variables i1_g4, i1_g3, i2_g4 mentionnées au-dessus). En revanche les listes des ensembles de substitution pour les duplications seront différentes :

1. pour le duplicate IGN : ("1","4"), ("1","3"), ("2","4") - il y aura donc 3 duplications et pour lesquelles il est possible d'utiliser %IGN.1% et %IGN.2% qui seront remplacés par "1" et "4" puis par "1" et "3" puis par "2" et "4" :

   ```
   <element dynamic="true" variables="+i%IGN.1%_g%IGN.2%" name="test ilot %IGN% - grp %IGN.2%"/>
   ```

   sera remplacé par

   ```
   <element dynamic="true" variables="+i1_g4" name="test ilot 1 - grp 4"/>
   <element dynamic="true" variables="+i1_g3" name="test ilot 1 - grp 3"/>
   <element dynamic="true" variables="+i2_g4" name="test ilot 2 - grp 4"/>
   ```

2. IN : "1", "2" - il y aura donc 2 duplications et pour lesquelles il est possible d'utiliser %IN% (ou %IN.1%) qui sera remplacé par "1" puis par "2" :

   ```
   <element dynamic="true" variables="+i%IN.1%_g(.*)" name="test ilot %IN%"/>
   ```

   sera remplacé par

   ```
   <element dynamic="true" variables="+i1_g(.*)" name="test ilot 1"/>
   <element dynamic="true" variables="+i2_g(.*)" name="test ilot 2"/>
   ```

3. GN : "3", "4" - il y aura donc 2 duplications et pour lesquelles il est possible d'utiliser %GN% (ou %GN.1%) qui sera remplacé par "3" puis par "4" :

   ```
   <element dynamic="true" variables="+i(.*)_g%GN%" name="test grp %GN.1%"/>
   ```

   sera remplacé par

   ```
   <element dynamic="true" variables="+i(.*)_g3" name="test grp 3"/>
   <element dynamic="true" variables="+i(.*)_g4" name="test grp 4"/>
   ```

4. IGXN : ("1","3"), ("1","4"),("2","3"),("2","4") - il y aura donc 4 duplications et pour lesquelles il est possible d'utiliser %IGXN.1% et %IGXN.2% qui seront remplacés par "1" et "3" puis par "1" et "4" puis par "2" et "3" puis par "2" et "4" :

   ```
   <element dynamic="true" variables="+i%IGXN.1%_g%IGXN.2%" name="test ilot %IGXN% - grp %IGXN.2%"/>
   ```

   sera remplacé par

   ```
   <element dynamic="true" variables="+i1_g3" name="test ilot 1 - grp 3"/>
   <element dynamic="true" variables="+i1_g4" name="test ilot 1 - grp 4"/>
   <element dynamic="true" variables="+i2_g3" name="test ilot 2 - grp 3"/>
   <element dynamic="true" variables="+i2_g4" name="test ilot 2 - grp 4"/>
   ```

Concaténation de chaine

- Concatenation de valeurs
  1. pour les valeurs (A, B, C),
     (a) le PATTERN %ID$$% sera remplacé par ABC
     (b) le PATTERN %ID.1$$% sera remplacé par ABC
  2. pour les valeurs avec plusieurs champs ((A,titi), (B,toto), (C,tata)),
     (a) le PATTERN %ID$$% sera remplacé par ABC
     (b) le PATTERN %ID.1$$% sera remplacé par ABC
     (c) le PATTERN %ID.2$$% sera remplacé par tititototata
- Concaténation de valeurs avec préfix
  1. pour les valeurs (A, B, C),
     (a) le PATTERN %ID$Pref$% sera remplacé par PrefAPrefBPrefC
     (b) le PATTERN %ID.1$Pref$% sera remplacé par PrefAPrefBPrefC
  2. pour les valeurs avec plusieurs champs ((A,titi), (B,toto), (C,tata)),
     (a) le PATTERN %ID$Pref$% sera remplacé par PrefAPrefBPrefC
     (b) le PATTERN %ID.1$Pref$% sera remplacé par PrefAPrefBPrefC
     (c) le PATTERN %ID.2$Pref$% sera remplacé par PreftitiPreftotoPreftata
- Concaténation de valeurs avec suffix
  1. pour les valeurs (A, B, C),
     (a) le PATTERN %ID$$Suf% sera remplacé par ASufBSufCSuf
     (b) le PATTERN %ID.1$$Suf% sera remplacé par ASufBSufCSuf
  2. pour les valeurs avec plusieurs champs ((A,titi), (B,toto), (C,tata)),
     (a) le PATTERN %ID$$Suf% sera remplacé par ASufBSufCSuf
     (b) le PATTERN %ID.1$$Suf% sera remplacé par ASufBSufCSuf
     (c) le PATTERN %ID.2$$Suf% sera remplacé par titiSuftotoSuftataSuf
- Concaténation de valeurs avec préfix et suffix
  1. pour les valeurs (A, B, C),
     (a) le PATTERN %ID$Pref$Suf% sera remplacé par PrefASufPrefBSufPrefCSuf
     (b) le PATTERN %ID.1$Pref$Suf% sera remplacé par PrefASufPrefBSufPrefCSuf
  2. pour les valeurs avec plusieurs champs ((A,titi), (B,toto), (C,tata)),
     (a) le PATTERN %ID$Pref$Suf% sera remplacé par PrefASufPrefBSufPrefCSuf
     (b) le PATTERN %ID.1$Pref$Suf% sera remplacé par PrefASufPrefBSufPrefCSuf
     (c) le PATTERN %ID.2$Pref$Suf% sera remplacé par PreftitiSufPreftotoSufPreftataSuf

Chapter 12

Création et édition du fichiers de règles

12.1 Création d'un fichier de règles

Pour créer un fichier de règle, procédez comme suit :

1. Dans la vue Navigateur, cliquez à l'aide du bouton droit de la souris sur le projet ou le dossier dans lequel vous souhaitez créer le nouveau fichier de règles.

2. Dans le menu en incrustation, sélectionnez **Nouveau** > **Fichier**.

3. Indiquez le nom du fichier, y compris le suffixe de fichier (par exemple, Mon fichier de règles.gqr). Le suffixe du fichier doit être impérativement **gqr** (glips quality rules) et respecter la casse.

4. Cliquez sur Fin.

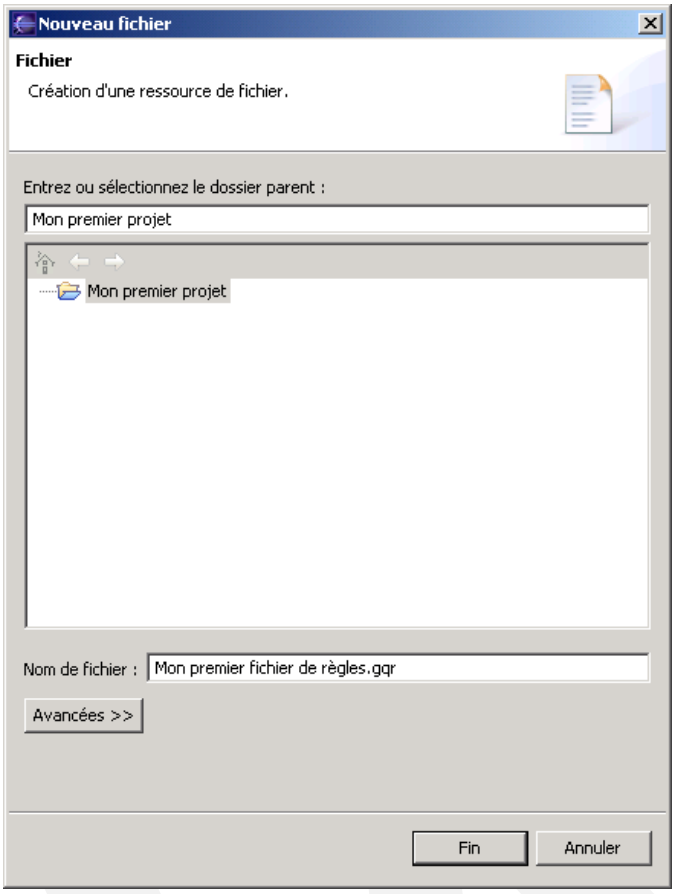


Figure 12.1: Création d'un nouveau fichier de règles

Vous obtenez un fichier de règles complet :

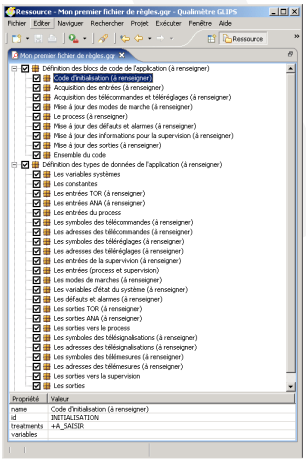


Figure 12.2: Le nouveau fichier de règles

Le fichier s'ouvre dans l'éditeur de règles pour le qualimètre GLIPS. Vous devez renseigner le fichier afin d'indiquer à PLC Checker comment s'appellent les différents types de blocs de code (initialisation, acquisition des entrées, émission des sorties etc...), mais aussi des types de variables (entrées, sorties, alarmes, etc...). Si nécessaire, vous devrez aussi indiquer où se trouve physiquement les tables pour la supervision.

IMPORTANT Avant de modifier les propriétés treatments (blocs de code) ou variables (types de variables), veuillez impérativement consulter la partie Concepts - Filtres afin de prendre note du formalisme utilisé lors de l'écriture de ces propriétés.

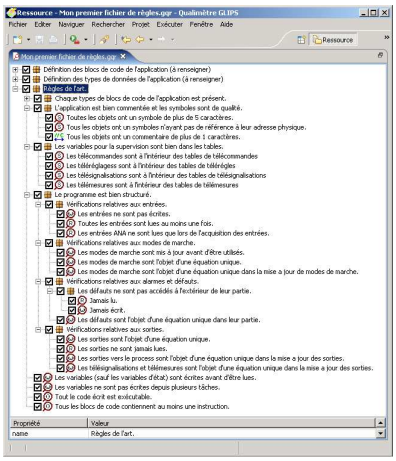


Figure 12.3: Les règles de l'art incluses par défaut dans chaque nouveau fichier de règles

12.2 Ajout d'une règle

Toutes les opérations relatives aux règles sont accessibles à partir du menu contextuel qui s'affiche lorsque l'on clique dans l'éditeur de règle avec le bouton droit de la souris.

L'ajout d'une règle est réalisé à l'aide du menu contextuel **Nouveau**.

Le type de la règle à ajouter est choisi à l'aide du dernier menu affiché (rulesGroup writeCheck symbolCheck commentCheck blockCheck readCheck).

- **Nouveau** > **Frère** > Avant: ajouter une règle juste au-dessus de la règle sélectionnée.
- **Nouveau** > **Frère** > Après: ajouter une règle juste au-dessous de la règle sélectionnée.

Si la règle sélectionnée est un groupe de règles, vous pouvez aussi lui ajouter un fils.

- **Nouveau** > **Fils** > Premier: ajouter un fils en tête de la liste des fils de la règle sélectionnée.
- **Nouveau** > **Fils** > Dernier: ajouter un fils en queue de la liste des fils de la règle sélectionnée.

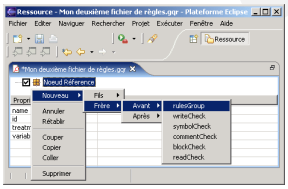


Figure 12.4: Création d'une nouvelle règle

12.3 Annulation des modifications

A l'aide du menu contextuel **Annuler**, vous pouvez revenir en arrière et annuler les modifications. L'opération inverse (refaire les modifications) se réalise à l'aide du menu contextuel **Rétablir**.

Note : Lorsque vous êtes revenu à l'état sauvegardé sur le disque, le menu contextuel **Annuler** est désactivé.

12.4 Copier et coller des règles

Lorsqu'un ensemble de règles contiguës et ayant le même père est sélectionné, vous pouvez, à l'aide du menu contextuel **Copier**, les mettre dans le presse-papier (contenant un ensemble de règles).

Le menu contextuel **Couper** a le comportement habituel, à savoir une suppression de la sélection après écriture dans le presse-papier. Vous pouvez, à l'aide du menu contextuel **Coller**, ajouter ensuite une ou plusieurs fois dans un ou plusieurs fichiers de règles la ou les règles contenues dans le presse-papier, extraites du fichier de règles. Les règles sont ajoutées au-dessous de la règle sélectionnée.

Note : Si la règle sélectionnée est du type "groupe de règles", et si ce groupe de règles n'a pas de fils, alors le presse-papier est ajouté en tant que fils de la règle sélectionnée.

12.5 Suppression de règles

Le menu contextuel **Supprimer** permet d'enlever la ou les règles sélectionnées. Le contenu de la sélection n'est pas mis dans le presse-papier.

12.6 Opérations relatives aux propriétés

A l'aide du menu contextuel qui s'affiche lorsque le curseur de saisie se trouve sur une propriété, vous pouvez réaliser des opérations **Copier**, **Couper**, **Coller** de, ou vers, le presse-papier textuel du système.

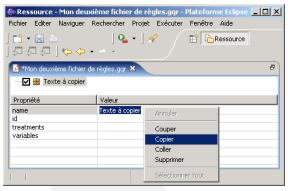


Figure 12.5: Copier-Coller

12.7 Impression du fichier de règles

Pour imprimer le fichier de règles ou inclure son contenu dans un document, vous devez tout d'abord enregistrer le fichier au format **html**.

Le menu **Fichier** > **Sauvegarder sous...** fait apparaître la boîte de dialogue **Sauvegarder sous...**

Dans la zone "Nom du fichier", changez l'extension du fichier de **gqr** en **html** (respectez la casse), et cliquez sur **OK**. Le fichier **html** est maintenant dans la fenêtre Navigateur.

Ouvrez ensuite le fichier **html** avec l'éditeur système (menu contextuel **Ouvrir avec** > **Editeur système**) et utilisez sa fonction d'impression (menu **Fichier** > **Imprimer...**) ou son presse-papier (menu **Edition** > **Copier**)

Chapter 13

Contrat de Licence Utilisateur Final pour logiciel PLC Checker

Le contrat de licence se trouve à l'adresse suivante : http://www.automationsquare.com/fr/licenses

IAS