

PLC Checker User Guide

Itris Automation Square- January 2020

www.itris-automation.com

©Itris Automation

2008-2019

Contents

1 Preamble **4**

2 Introduction **6**

3 Eclipse platform tutorial **7**
3.1 Installation . . . 7
3.1.1 Usage . . . 7
3.1.2 Hardware and software configuration needed . . . 7
3.1.3 Platform installation . . . 8
3.2 Software update . . . 11
3.3 Language Translation . . . 16
3.3.1 New language translation package installation . . . 16
3.3.2 Force Language Usage . . . 16
3.4 Launching the platform . . . 17
3.4.1 Workspace . . . 17
3.4.2 Identification . . . 17
3.4.3 Work bench . . . 18
3.5 Import/Export files into/from the project . . . 20
3.5.1 Drag'N Drop . . . 20
3.5.2 Importing files . . . 20
3.6 Comparing files . . . 23
3.7 Local history . . . 24
3.7.1 Compare with a older version . . . 24
3.7.2 Replacing by an older version . . . 25

4 PLC Checker project creation and usage **26**
4.1 PLC Checker project creation . . . 27
4.1.1 Creation with wizard of a default PLC Checker application on Itris Automation Square website (Default Plc Checker p
4.1.2 Creation with wizard of a PLC Checker application on the Itris Automation Square website (Plc Checker project to be
4.1.3 Access to an existing PLC Checker program on the Itris Automation Square website . . . 36
4.1.4 New PLC Checker project creation on Eclipse platform . . . 38
4.1.5 Creation of copy of existing PLC Checker project on Eclipse platform . . . 40
4.2 Run PLC Checker . . . 41
4.2.1 Run PLC Checker from an Eclipse platform : . . . 41
4.2.2 Run PLC Checker from the website: . . . 44
4.3 View PLC Checker results . . . 46
4.3.1 View results on Eclipse platform . . . 46
4.3.2 View results on the website . . . 48

5 PLC export files generation **50**
5.1 FEF export file generation from PL7-Pro . . . 50
5.2 L5K export file generation from RSLogix5000 . . . 50
5.3 ASC, AWL, GR7 and SCL Export files generation from Step7 . . . 51
5.3.1 Create ASC file . . . 52

5.3.2 Create AWL file . . . 53
5.3.3 Create GR7 file . . . 56
5.3.4 Create CFG file . . . 57
5.3.5 Create SCL source for CFC codes . . . 57
5.3.6 Safety applications . . . 59
5.3.7 Export files . . . 61
5.4 XEF or ZEF export file generation from Unity . . . 62
5.5 SMC file retrieval from OMRON Sysmac Studio . . . 62
5.6 XML export file generation from Codesys V3 . . . 62
5.7 EXP export file generation from Codesys V2 . . . 63
5.8 XML export file generation from SoMachine . . . 64
5.9 XML export file generation from Beckhoff Twincat . . . 66
5.10 EXP export file generation from Beckhoff TwinCAT V2 . . . 67
5.11 PC5 export file generation from RSLogix 5 . . . 68

6 Itris Import/Export Tool 70
6.1 Introduction . . . 70
6.2 Installation . . . 71
6.3 Usage . . . 72
6.3.1 Graphical UI for the daily usage . . . 72
6.3.2 Command line interface to automate the processes . . . 73
6.4 Visualise The Export Content . . . 74
6.5 Get the list of installed plugins . . . 74

7 Itris Import/Export Tool - STEP 7 76
7.1 Introduction of Itris Import/Export Tool . . . 76
7.2 STEP 7 Plugin Pre-requisite . . . 76
7.3 Specific warnings about STEP 7 . . . 76
7.3.1 Programming languages . . . 76
7.3.2 Command line interfaces . . . 77
7.3.3 Compatibility with the old versions of PLC Checker, PLC DocGen and PLC Converter . . . 77

8 Itris Import/Export Tool - TIA Portal 78
8.1 Introduction of Itris Import/Export Tool . . . 78
8.2 Available plugins for TIA Portal . . . 78
8.3 Specific Pre-requisite for TIA Portal . . . 78
8.4 Plugins Setup . . . 79
8.4.1 Openness installation . . . 79
8.4.2 Users authorisation . . . 79
8.4.3 Authorization files . . . 80
8.5 TIA Portal specific information . . . 80
8.5.1 Non compiled codes exportation . . . 80
8.5.2 Protected code exportation . . . 80
8.5.3 Project versions . . . 81
8.5.4 Plugin authorisation . . . 82
8.6 Introduction of Itris Import/Export Tool . . . 82
8.7 Specific Pre-requisites for Unity . . . 82
8.8 Specific Information for Unity . . . 82
8.8.1 Compatibility with the old versions of PLC Checker, PLC DocGen et PLC Converter . . . 82
8.9 XML export file generation from Phoenix Contact Multiprog . . . 83

9 Concepts **85**
9.1 Rules . . . 85
9.1.1 Generality . . . 85
9.1.2 Rules activation . . . 85
9.1.3 Rules name . . . 85
9.1.4 Rules Identity . . . 86
9.1.5 Code filter . . . 86
9.1.6 Data filter . . . 86
9.1.7 To enable a rule using other rules . . . 86
9.1.8 To enable one rule or rules group in a list . . . 87
9.1.9 Identifier for .csv format export . . . 87
9.1.10 Node visibility . . . 87
9.1.11 Automatic Node visibility using other rules . . . 87
9.1.12 List of editable properties for protected files . . . 88
9.1.13 Errors severity . . . 88
9.1.14 "warningThreshold" property . . . 88
9.1.15 "errorThreshold" property . . . 89
9.1.16 "defaultResultId" property . . . 89
9.1.17 "errorMessage" property . . . 89
9.1.18 "noMatchMessage" property . . . 89
9.1.19 "noMatchSeverity" property . . . 89
9.1.20 "isTextEditable" property . . . 89
9.1.21 Generator of attribute value . . . 90
9.2 Filters . . . 92
9.2.1 Generality . . . 92
9.2.2 Limitations . . . 92
9.2.3 Symbols filters . . . 92
9.2.4 Symbols filters using regular expressions . . . 93
9.2.5 Attributes filters . . . 94
9.2.6 Absolutes addresses filters . . . 95
9.2.7 Filters usage . . . 95
9.2.8 Filters intersection . . . 96
9.2.9 Filters union . . . 96
9.2.10 Filters difference . . . 96
9.3 Glips Attributes . . . 96
9.4 Result file . . . 104
9.4.1 Printing results file . . . 104

10 The rules **105**
10.1 Rules group (rulesGroup) . . . 105
10.1.1 Usage . . . 105
10.1.2 Properties . . . 106
10.2 Read Check (readCheck) . . . 106
10.2.1 Usage . . . 106
10.2.2 Properties . . . 106
10.3 Write Check (writeCheck) . . . 107
10.3.1 Usage . . . 107
10.3.2 Properties . . . 107
10.4 Symbol check (symbolCheck) . . . 109
10.4.1 Usage . . . 109
10.4.2 Properties . . . 109
10.5 Attribute check (attributeCheck) . . . 110
10.5.1 Usage . . . 110
10.5.2 Properties . . . 110
10.6 Code blocks check (blockCheck) . . . 112

10.6.1 Usage . . . 112
10.6.2 Properties . . . 112
10.7 Equation check (equationCheck) . . . 112
10.7.1 Usage . . . 113
10.7.2 Properties . . . 113
10.8 Memory check (memoryCheck) . . . 114
10.8.1 Usage . . . 114
10.8.2 Properties . . . 114
10.9 Copy/Past percentage check (copyPastCheck) . . . 115
10.9.1 Usage . . . 115
10.9.2 Properties . . . 115
10.10 Check FEF program structure in source file (FEFStructureCheck) . . . 115
10.10.1 Usage . . . 115
10.10.2 Properties . . . 115
10.11 Check attributes value in export source file (FEFAttributeCheck) . . . 116
10.11.1 Usage . . . 116
10.11.2 Properties . . . 116
10.12 Label check (labelCheck) . . . 117
10.12.1 Usage . . . 117
10.12.2 Properties . . . 117
10.13 Pattern matching check (patternMatchingCheck) . . . 118
10.13.1 Usage . . . 118
10.13.2 Properties . . . 118
10.14 Automatic duplication of rules (duplicate) . . . 119
10.14.1 Execution of the duplication . . . 119
10.14.2 Definitions . . . 120
10.14.3 Syntax . . . 120
10.14.4 Behavior . . . 120
10.14.5 Examples . . . 122

11 Rules file creation and edition **126**
11.1 Rules file creation . . . 126
11.2 Add a rule . . . 128
11.3 Cancel modifications . . . 129
11.4 Copy and past rules . . . 129
11.5 Delete rules . . . 129
11.6 Operations on rules properties . . . 129
11.7 Printing rules file . . . 130

12 Final license contract for PLC Checker **131**

Chapter 1

Preamble

This document is the user guide of the PLC Checker software of Itris Automation Square.

After a short introduction, the Eclipse platform dedicated to PLC Checker is presented (installation, updates, usage,...).

The rules files creation and edition and the PLC export files generation are explained in the two following chapters.

Next chapter describes the PLC Checker project creation, the way to launch the PLC Checker and how to show PLC Checker results.

An introduction to the filter and rules concepts and a description of the result file are given in the following chapter.

The last two chapters show the rules references and the link to the PLC Checker license contract.

1. Introduction
2. Eclipse platform tutorial
   (a) Installation
   (b) Update
   (c) New language installation
   (d) Launch
   (e) File import in the projects
   (f) Files comparison
   (g) Local history
3. PLC Checker project creation and usage
   (a) PLC Checker project creation
      i. Creation on IAS with wizard (Default PLC Checker project).
      ii. Creation on IAS with wizard (PLC Checker project to be defined).
      iii. Creation of copy of existing PLC Checker program on IAS.
      iv. Creation of New PLC Checker project on Eclipse platform
      v. Creation of copy of Existing PLC Checker project on Eclipse platform
   (b) Run PLC Checker
      i. Run PLC Checker on Eclipse platform
      ii. Run PLC Checker on IAS web site
   (c) View Results
      i. View results on Eclipse
      ii. View results on IAS web site
4. PLC Export file generation
   (a) FEF file generation from PL7Pro

(b) L5K file generation from RSLogix5000
(c) ASC, AWL, GR7 and SCL files generation from Step7
(d) XEF file generation from Unity
(e) SMC file retrieval from OMRON Sysmac Studio
(f) EXPORT file generation from Codesys v3
(g) EXP file generation from Codesys v2
(h) XML file generation from SoMachine
(i) XML file generation from Beckhoff Twincat V3
(j) EXP file generation from Beckhoff Twincat V2
(k) PC5 file generation from RSLogix 5
(l) Itris Import/Export Tool (automated code exportation)
i. STEP 7
ii. TIA Portal version 13 SP1, 13 SP2, 14, 14 SP1 and 15
iii. Unity
(m) Phoenix Contact Multiprog

5. Concepts
(a) Rules
(b) Filters
(c) Glips Attributes
(d) Results file
6. Rules references
7. Rules file creation and edition
(a) Rules file creation
(b) Add a rule
(c) Cancel modifications
(d) Copy/Past
(e) Delete a rule
(f) Rule property
(g) Printing rules
8. License contract

IAS

Chapter 2

Introduction

PLC Checker analyzes a PLC code to check if it respects programing and conception rules defined in a rules file. After checking, a results file is generated. The rules violations are logged in the results file.
This PLC Checker version supports PLC programs of Schneider-Electric PL7-PRO, Siemens Symatic Step 7, Schneider-Electric UnityPro, Rockwell Automation RSLogix5000 workbenches and many more.
Step to follow to use PLC Checker in SaaS (Software as a Service), are described in this user guide.

The PLC Checker obtains the "Grand Prix de l'Innovation 2004", organized by AUTOMATION-OPTIMATION EUROPE in the "meilleur logiciel d'automatisme et de gestion industrielle" category.

IAS

Chapter 3

Eclipse platform tutorial

The aim of this tutorial is to install and to get started with the Eclipse platform.

You will find more details about the installation and the necessary plugins, the definition of the workspace, workplan and project concepts and finally the most significant functions will be explained in more details.

3.1 Installation

This section contains the installation procedure for the Eclipse platform.

3.1.1 Usage

Warning, the use of Itris Automation Square products is licensed and payed for as SaaS mode (Software as a Service) on a server (cloud or private) and is conditioned by a commercial validation. To obtain a license, please contact the commercial@automationsquare.com.

Creating a user account on the site Itris Automation Square is mandatory to use Itris Automation Square products.

In order to use Itris Automation Square products, the user has to identify himself by entering his login and password defined during his registration on IAS website.

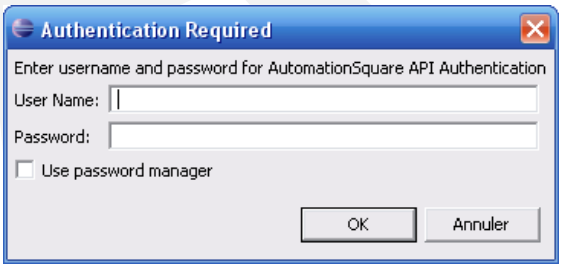


Figure 3.1: login

3.1.2 Hardware and software configuration needed

Windows OS

The Itris Automation Square workshop is integrated in the Eclipse 4.2.x (The PLC Checker plugin is pre-installed. See installation program link in Installation chapter below).

This platform and the workshop specific code need the Sun java virtual machine JRE 5.0.

This software configuration require a PC with following minimal characteristics:

- Microsoft Windows 2000/XP Professional/Windows 7/Windows 8,
- Minimal Processor frequency of 1GHz,

- 200 MB[1] of free disk space,

- 512 MB[1] of RAM,

Other OS

For Linux and Mac operating systems, the installation of the Eclipse platform is not pre-packaged with PLC Checker plugin pre-installed. The minimal configuration needed are described on Eclipse website. See installation program link in Installation chapter below.

3.1.3 Platform installation

Windows OS

The installation program is available here, and is named InstallPlcChecker.exe. Launch the InstallPlcChecker.exe in order to install the platform.

The installation program creates an entry in the windows start menu.
Detailed procedure:

- Choose installer language



Figure 3.2: Choose language

- On the welcome message, click on **Next**

[1] Megabyte



Figure 3.3: Welcome in installation program

- In order to install the PLC Checker platform, you have to **agree the license**.

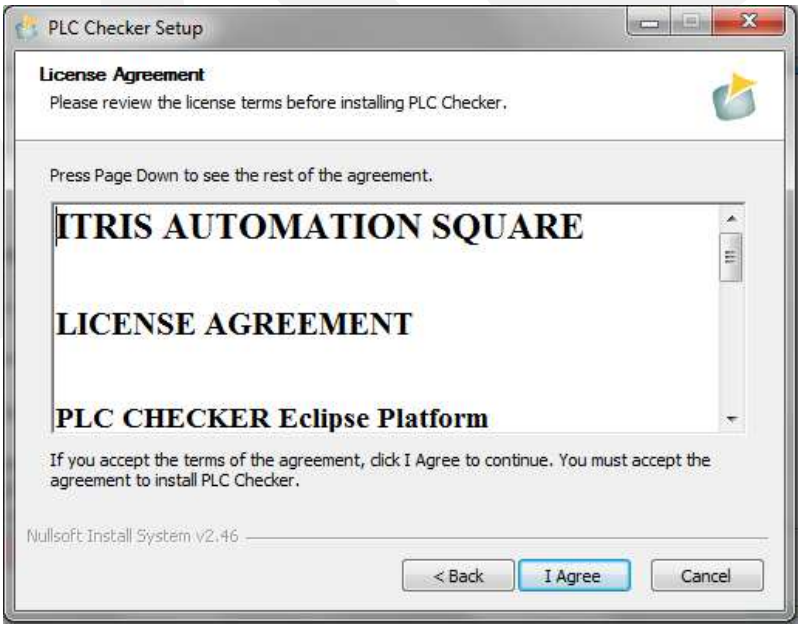


Figure 3.4: License

- Choose the installation directory (by default C:\IAS), then click on **Install**

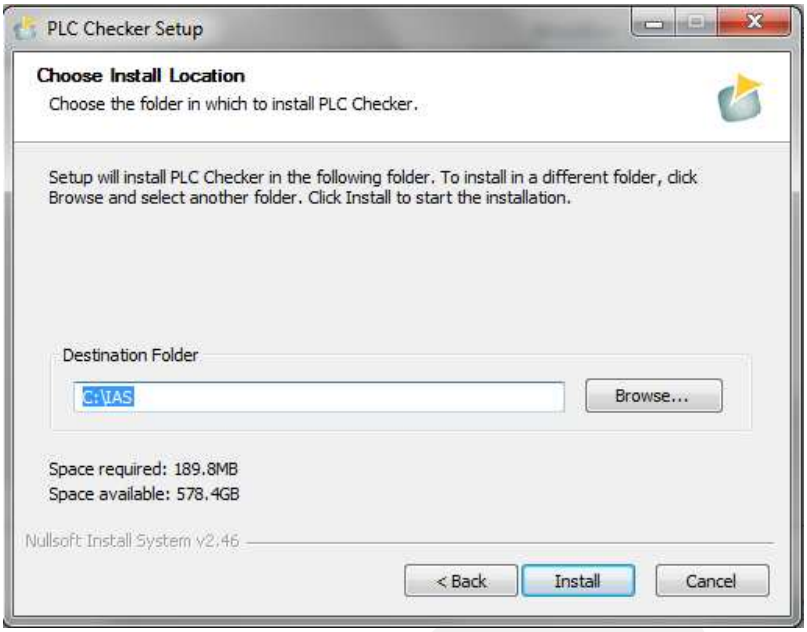


Figure 3.5: Choose the installation directory

- After installation, click on **Finish** to close the installation program and to run the PLC Checker.

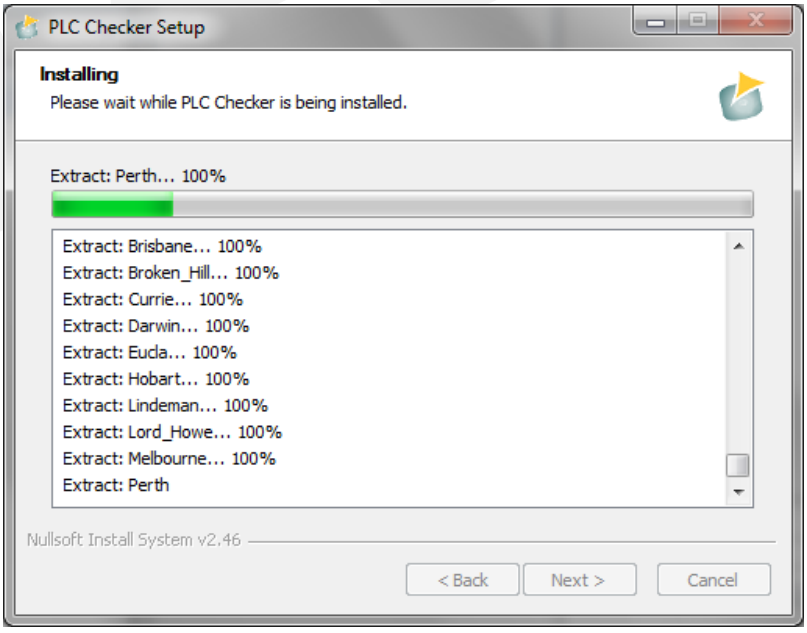


Figure 3.6: Installation

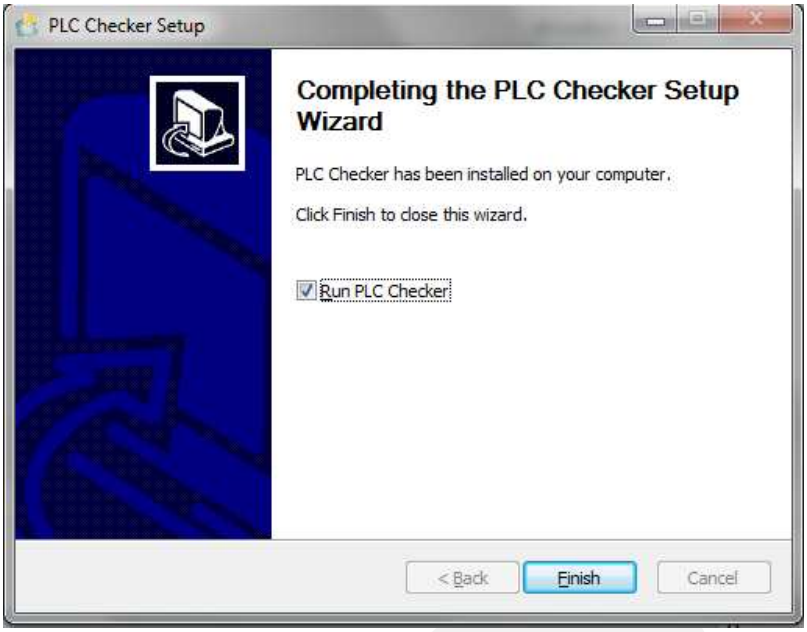


Figure 3.7: Close installation program

Others OS

The installation procedure for Itris Automation Square workshop on Linux or Mac Operating System contains two steps (first Eclips installation, then Itris plugins installation):

1. Install the Eclipse platform in version 4.2.x from the eclipse website by following the procedure corresponding to your OS[2].

2. Install the Itris Automation Square PLC Checker plugins:

   (a) Open the newly installed Eclipse platform

   (b) Ask for new software installation, using menu: Help / Install New Software...

   (c) Select the Itris Automation Square plugin repository by typing http://www.automationsquare.com/update-4.2 in the "Work with" field

   (d) Select the "GLIPS platform" and "PLC Checker" plugin categories

   (e) Install the selected plugin categories and restart the Eclipse platform

3.2 Software update

The search of available updates is done automatically by the platform when it starts.

[2]Operating System

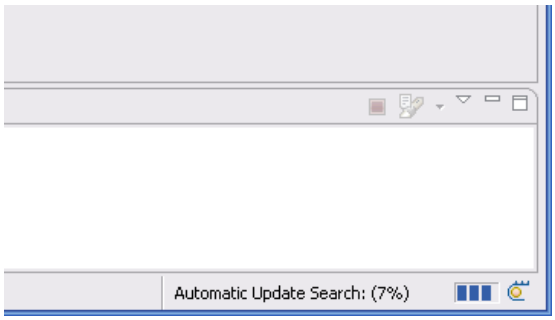


Figure 3.8: Automatic search for update

- When plugins updates are found, the platform ask you to view and install them. Click on **Yes**.

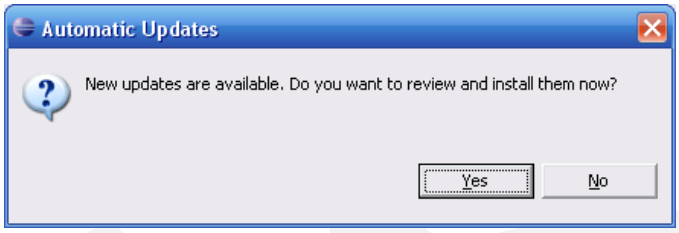


Figure 3.9: Ask for installation

- Available updates can be seen. Let all updates selected and click on **Next**.

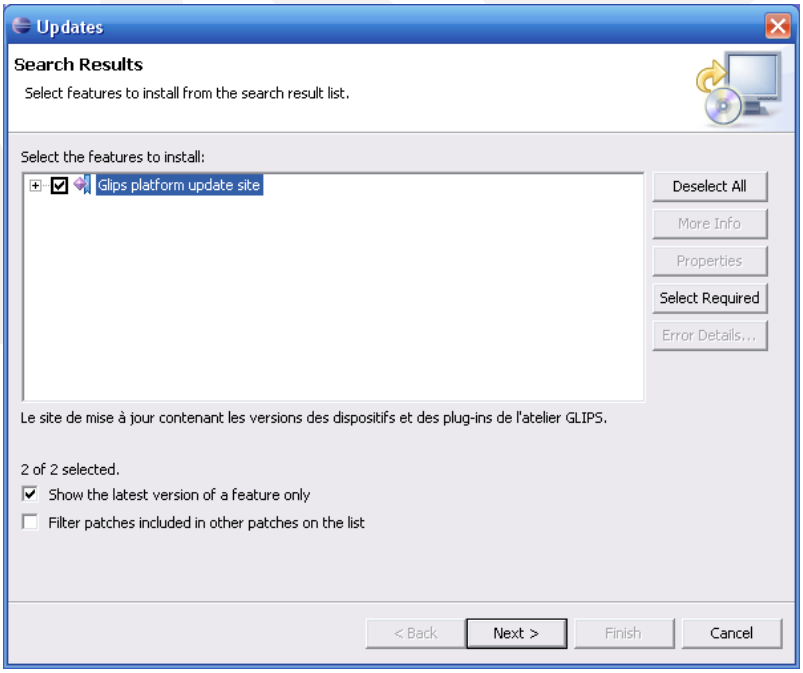


Figure 3.10: Updates selection

- **Accept license** contract, then click on **Next**.

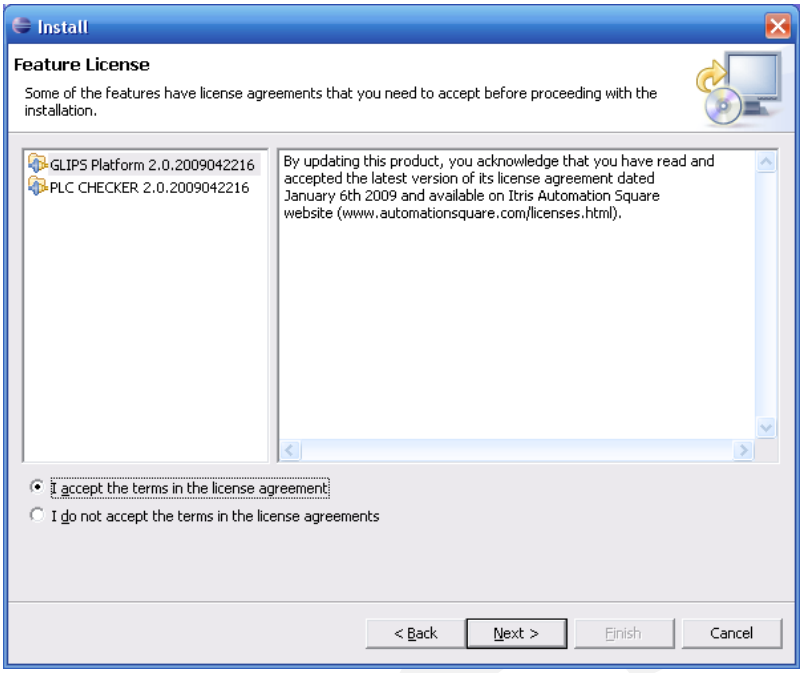


Figure 3.11: License

- The overview of updates to be installed is presented. Click on **Finish** to launch updates download.

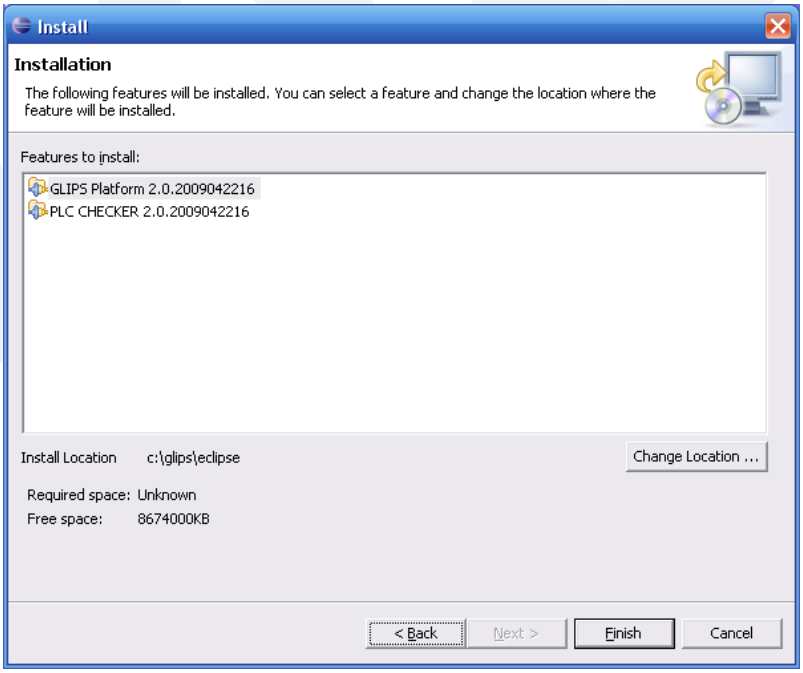


Figure 3.12: Download overview

- After download, you are asked to check the update emitter, because of unsigned updates. Click on **Install all**.

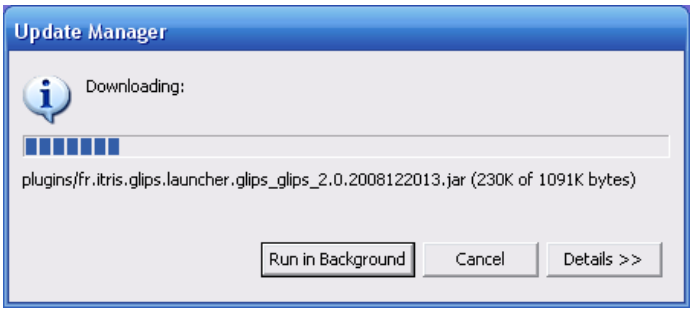


Figure 3.13: Downloading updates

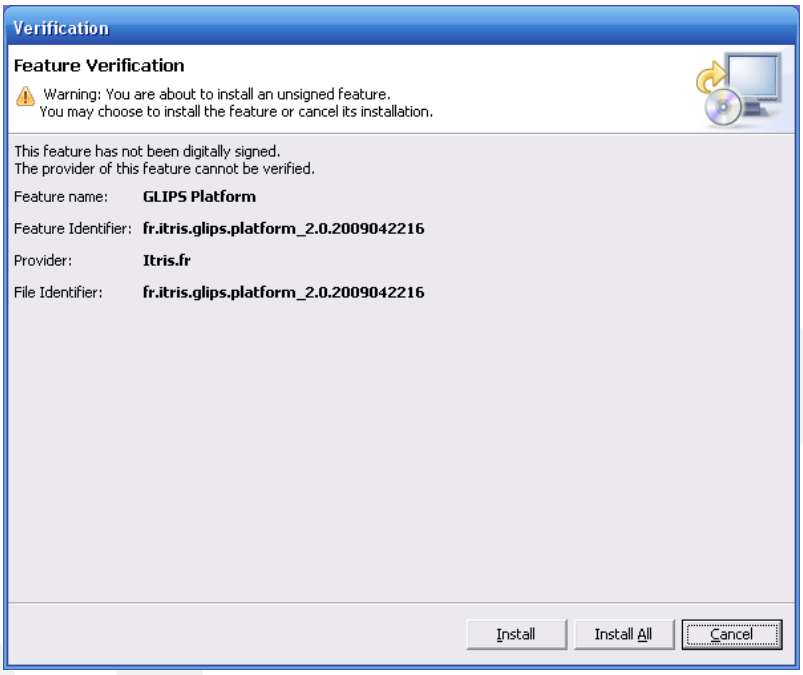


Figure 3.14: Check updates emitter

- After installation, restart the platform as asked to. Click on **Yes**.

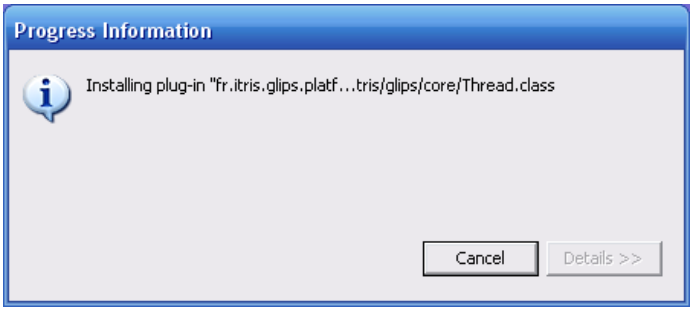


Figure 3.15: Updates installation

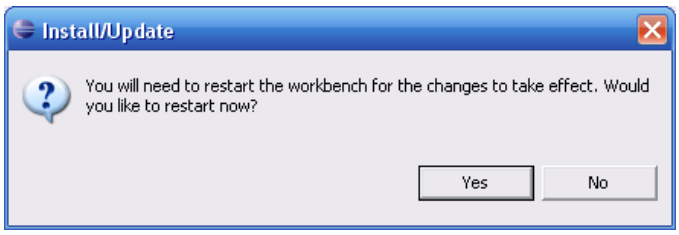


Figure 3.16: Asking to restart

IAS

3.3 Language Translation

The Eclipse platform start by default using the language of Operating System language.

But if this language translation package is not installed in the Eclipse platform, then English version is used by default.

In order to install a new language translation package in the Eclipse platform, follow the next chapter.

In order to force the usage of a language that is not your operating system language, please follow the chapter [Force Language Usage].

3.3.1 New language translation package installation

here is an example of a link for the installation of a language package in eclipse :

http://archive.eclipse.org/technology/babel/update-site/R0.11.1/juno

1. Open the eclipse platform
2. Use the menu help/install new software. . .
3. Copy the previous link and past it in the field named "work with:"
4. Search for the language package (ex: "Babel Language Pack in Spanish") and open it
5. Select the three following packages:
   (a) eclipse
   (b) rt.equinox
   (c) modeling emf
6. Click on Next
7. Click on Next
8. Select "I accept . . . "
9. Click on finish
10. Restart the Eclipse platform

3.3.2 Force Language Usage

In order to force the usage of a different language than the one of your Operating System, please follow the next steps:

1. Close the Eclipse platform
2. Edit the property of the shortcut on the Eclipse platform.
3. At the end of the Target field of this shortcut, add the option " -nl xx" where xx is the abbreviation of the desired language (ex: to force Spanish language usage the target must be something like "C:/IAS/eclipse/eclipse.exe -nl sp ")
4. save the modification of the properties and use this shortcut to relaunch the Eclipse platform.

3.4 Launching the platform

To launch the platform, click on the icon from the Windows desktop or choose the Glips platform item from the Start Menu.

At the end of the installation, the platform will start automatically.

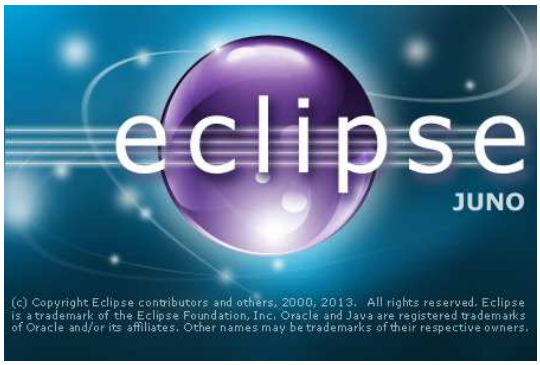


Figure 3.17: Eclipse Logo

3.4.1 Workspace

When launching the platform, the following dialog box pops up :

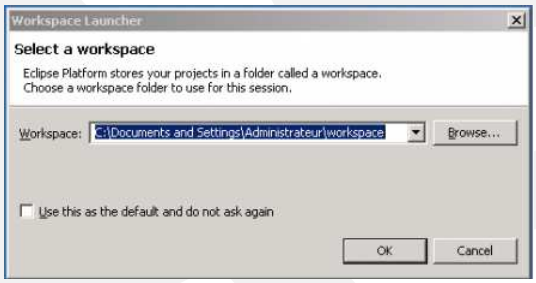


Figure 3.18: Workspace selection

The workspace is a hard disk drive directory reserved for Eclipse needs. In this specified folder, Eclipse builds up another folder called .metadata containing all the data it needs to work. Note that it is important :

1. To split the Eclipse workspace from your current work folders
2. Not to use the Windows explorer to generate files in the workspace (for this purpose use the import/export function or file drag'n drop function)
3. To have only one workspace

The **.metadata** contains the Eclipse project list.

Once the folder is chosen, you won't get the start question again on the next start of the platform when the option in the dialog box is ticked.

3.4.2 Identification

When you are asked for your identification enter your login and password that you obtained by registering on our website.

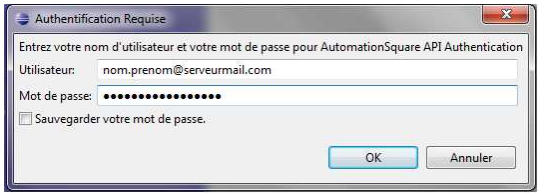


Figure 3.19: login

3.4.3 Work bench

When first starting the platform, the following welcome screen is displayed. You may follow the Eclipse tutorials. You also may quit this screen by clicking on "go to the work bench" or closing the inner window.

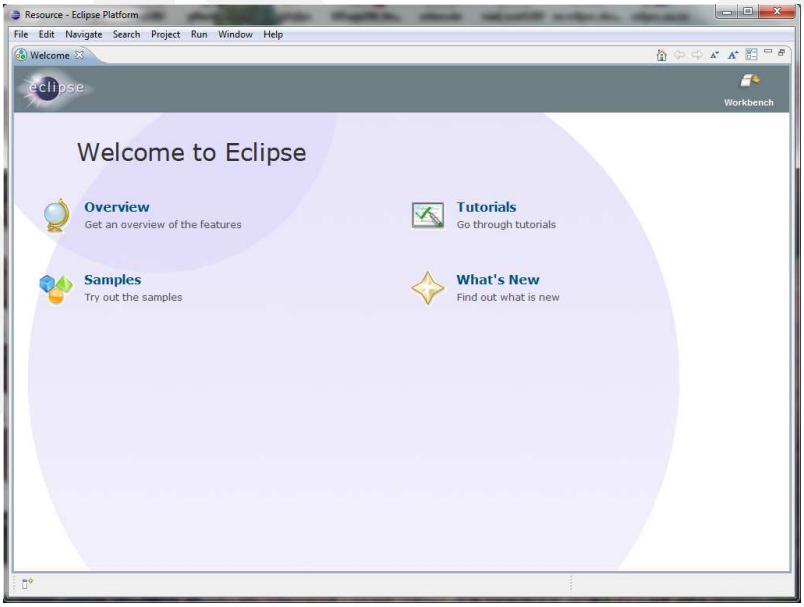


Figure 3.20: Eclipse welcome page

The Eclipse work bench includes several windows. The default layout is as shown on the following picture :

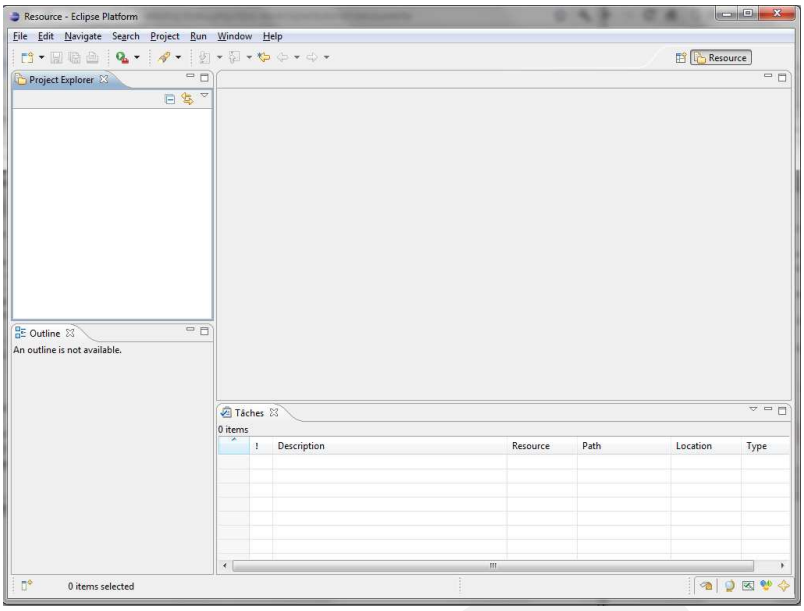


Figure 3.21: Eclipse default layout

On the left part is the explorer. You may view and browse your working space.

On the right part, the big grey window displays the used editors when opening files of the work space.

At the bottom left, a structured overview of the current editor is displayed. It is empty in the present case.

See the ‘to do’ window on the bottom right of the page. You may use this part to plan your tasks. Search for "Tasks view" in the Eclipse help for more details.

Warning: Search for updates is done automatically each time you start the platform. If the interface does not seem to answer, check if it is not updating.

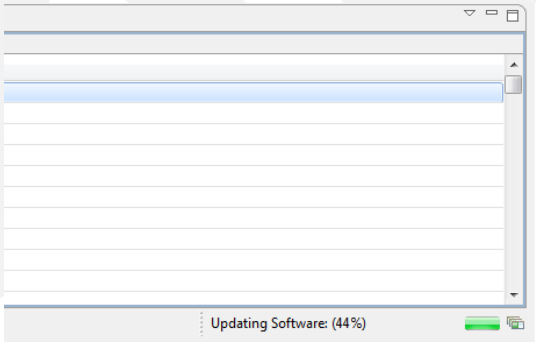


Figure 3.22: updating

3.5 Import/Export files into/from the project

File import must be done only using the Eclipse feature. There is no way using the Microsoft file browser. Importing or exporting data can be completed in different ways in the workspace. The drag'n drop and use of import features are described hereunder.

3.5.1 Drag'N Drop

The following snapshots show the different steps to import files into the workspace using the drag'n drop way. A file browser must be open and you only need to drag the desired files and drop them to the workspace. This operation does not erase the files from their original location.

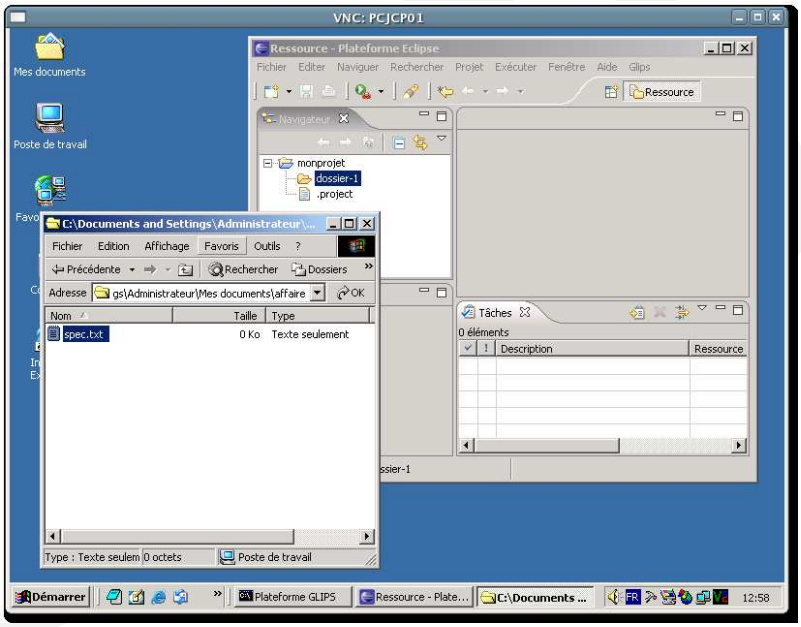


Figure 3.23: Drag'n Drop files

3.5.2 Importing files

The import feature using the dialog boxes is more complex. It is more suitable for importing only few files among others of a folder; selecting them becomes easier.

In the following example, you wish to import files into myfolder. myfolder must be selected, right-click and choose 'Import'

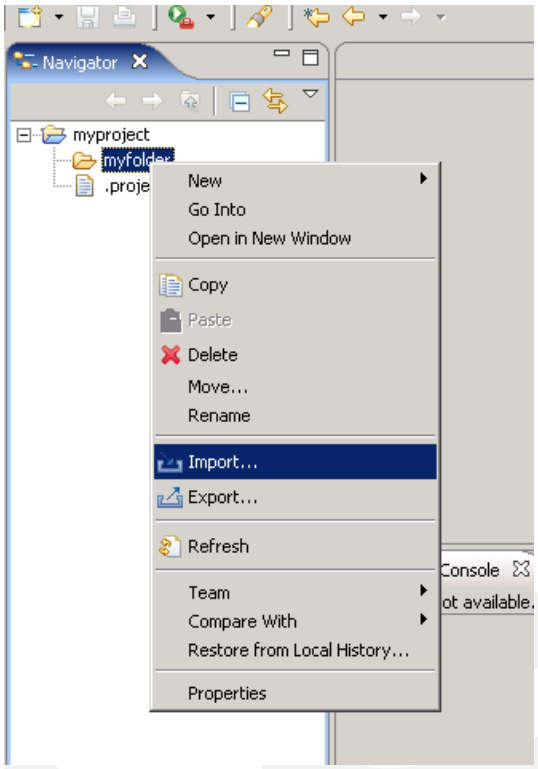


Figure 3.24: Contextual menu - Import

Among all the alternatives, importing files from the file system is the most interesting.

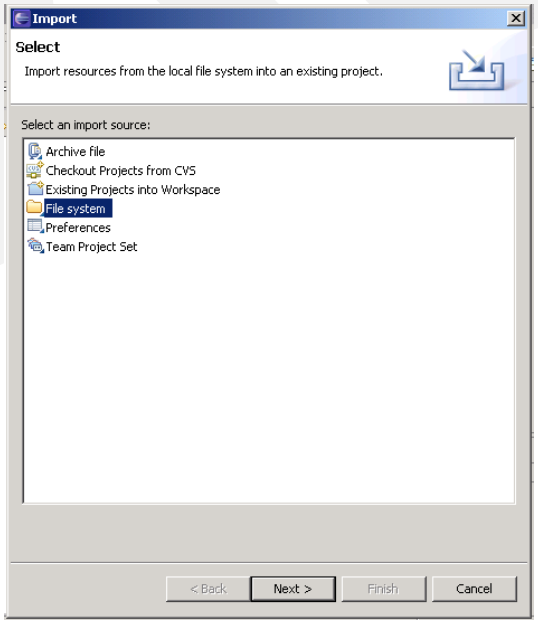


Figure 3.25: Select file system source for import

You must now select the hard disk directory from which the files will be imported.

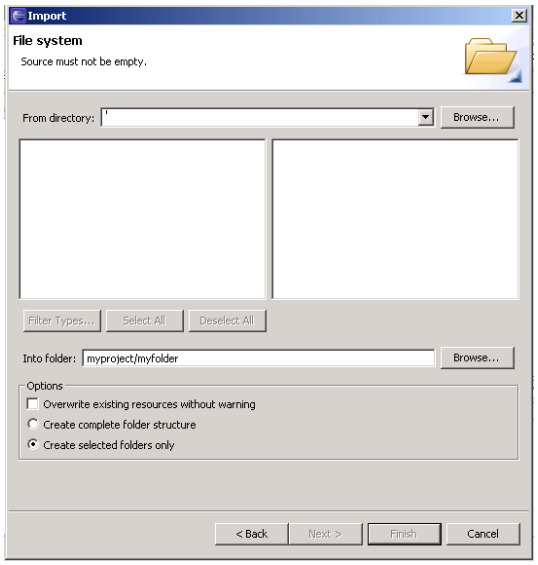


Figure 3.26: Select directory for import

Click on 'Browse'. A folder selection box pops up.

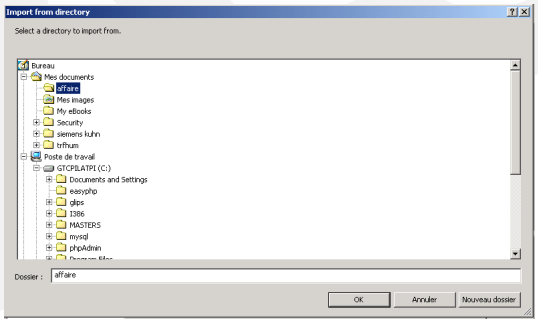


Figure 3.27: Browse directory to select one

Once the source folder is chosen, confirming displays the list of importable files. Select now the files to be imported, in our case we choose 'spec[3].txt'.

[3] specification

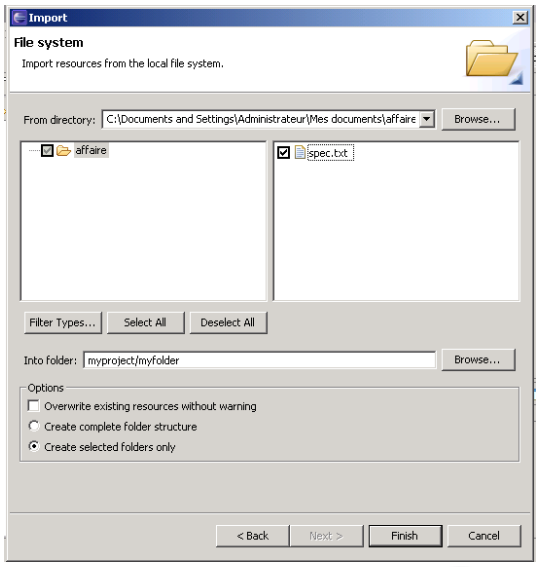


Figure 3.28: In the directory, select the file(s) to be imported

After confirming the desired file is imported into the workspace.

3.6 Comparing files

Comparing files in the Eclipse environment is a very easy operation.

Select first the both files to be compared.

The pop-up menu, obtained by a right click, allows you to perform a crossed comparison between selected files (menu: compare / with each other).

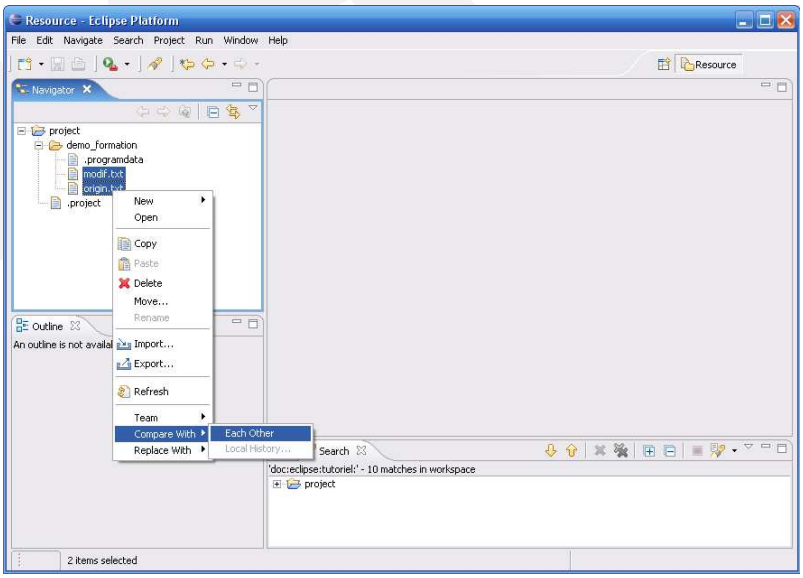


Figure 3.29: Launch files comparison

Selecting the menu input allows to open the difference editor :

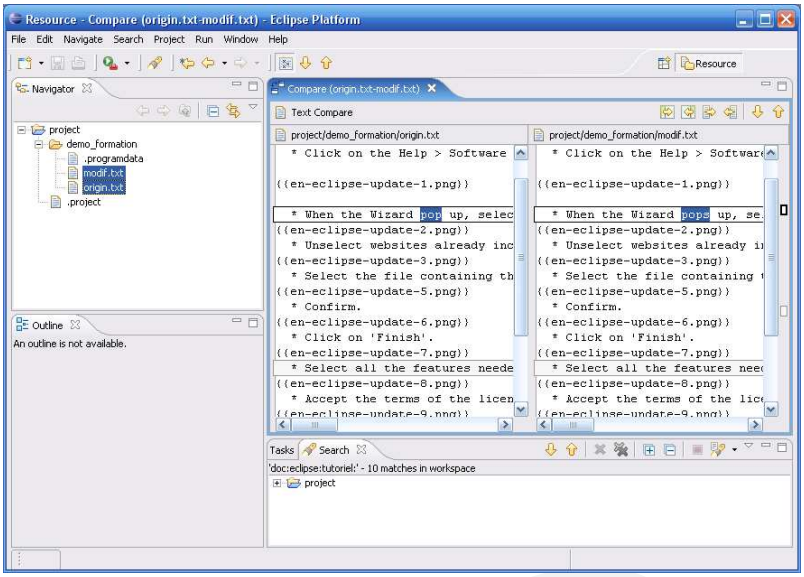


Figure 3.30: Look for differences

You can shift between differences by clicking the small white squares on the right side of the window or by using up and down arrows on the top right side.

To exit the comparison mode, only close the main window.

3.7 Local history

Eclipse provides a local history feature for all text files (not only *.txt) that belong to a project.

Traces of iterative modifications of text files are kept.

At Anytime, the user is able to check the differences between the current version and the former ones and get back to a previous version quickly and easily.

3.7.1 Compare with a older version

Only select one file in the Eclipse environment. Select ’Compare > Local history’ with the pop-up menu.

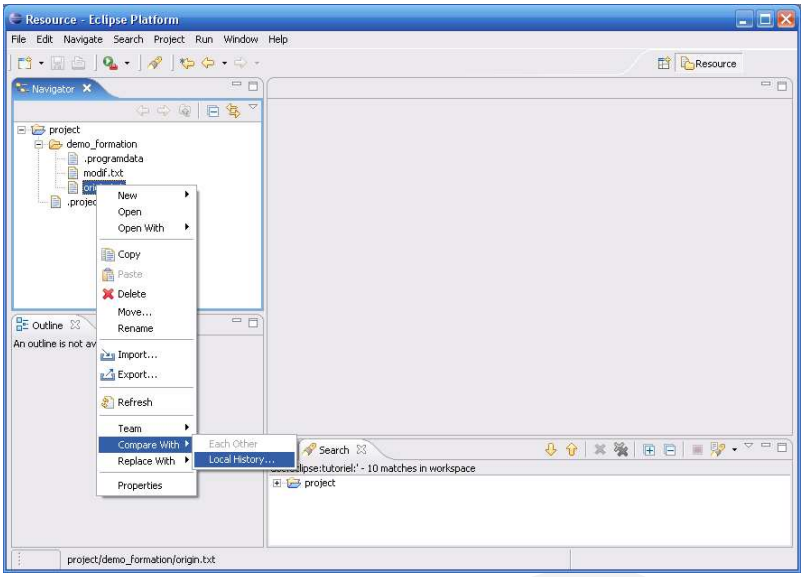


Figure 3.31: Launch local history comparison

The windows that pops up looks pretty much like the one to compare files at the top which is the local history of the file.

Successively selecting versions gives you the differences between the current version and the selected one.

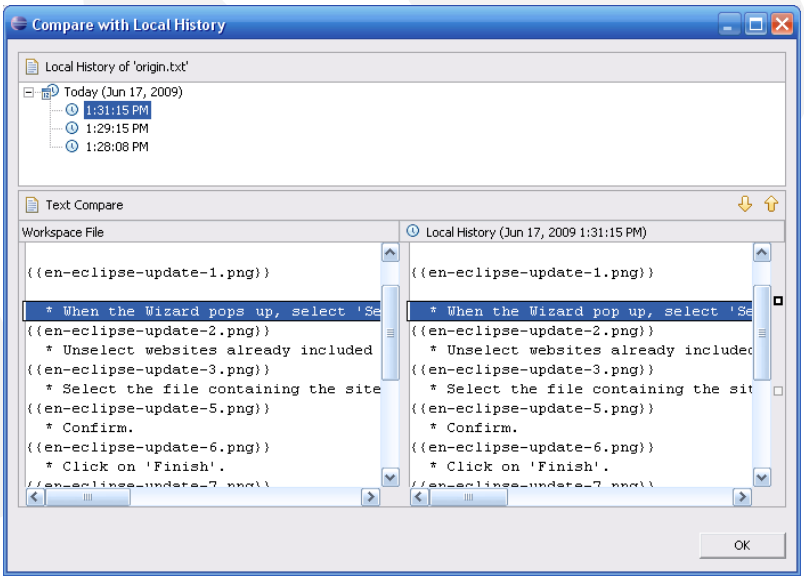


Figure 3.32: Local history comparison

3.7.2 Replacing by an older version

In the pop-up menu, choosing 'Replace by local history', replacing the current version by an older version becomes possible, at the same time viewing the differences.

Chapter 4

PLC Checker project creation and usage

This chapter describes the procedure for project creation, PLC Checker usage and results visualization.
The synthetic procedure is the following:

- **Preparations**:
  1. Create an user account on Itris Automation Square website
  2. Obtain PLC Checker rights from Itris Automation Square commercial service: commercial@itris-automation.com
  3. Obtain a rules file from your client or from your quality service or from the Itris Automation Square support@itris-automation.com or create and edit your own rules file (*.gqr).
  4. Generate your PLC application Export file(s) from your PLC workbench.
- **Create** project:
  1. Create a PLC Checker project on the Eclipse platform
     (a) Name and describe your project on the Eclipse platform
     (b) Associate a right to your project on the Eclipse platform
  2. Create a PLC Checker project on the Itris Automation Square website
     (a) Name and describe your project on the website (step 1)
     (b) Associate a right to your project on the website (step 4)
- **Execute** PLC Checker:
  - Method 1 : from the Eclipse platform :
    1. Create an PLC Checker project in your workspace of the Eclipse platform, that gets the contents and the characteristics of the PLC Checker project previously created on the website.
    2. Import PLC application export file(s) in this project folder on your Eclipse workspace.
    3. Import the rules file in this project folder on your Eclipse workspace.
    4. Execute PLC Checker from your Eclipse workspace.
  - Method 2 : from the website:
    1. Upload PLC application export file(s) to the server from the website (step 2).
    2. Upload rule file to the server from the website (step 3).
    3. Execute PLC Checker on the server from the website (step 5).
- **View** results on the Eclipse platform:
  1. If you have chosen method 2, you need to create a PLC Checker project in your workspace of the Eclipse platform, that gets the contents and the characteristics of the PLC Checker project previously created on the website (see method 1).
  2. View results file (*.gqre) on Eclipse platform.

4.1 PLC Checker project creation

4.1.1 Creation with wizard of a default PLC Checker application on Itris Automation Square website (Default Plc Checker project)

To create a Plc Checker application with the wizard, please click on "+New Plc Checker" button or click on "Tools" then "PLC Checker" in the menu on the left of the page. The application is created by default in the default project, you can move it to another project after finishing the creation.

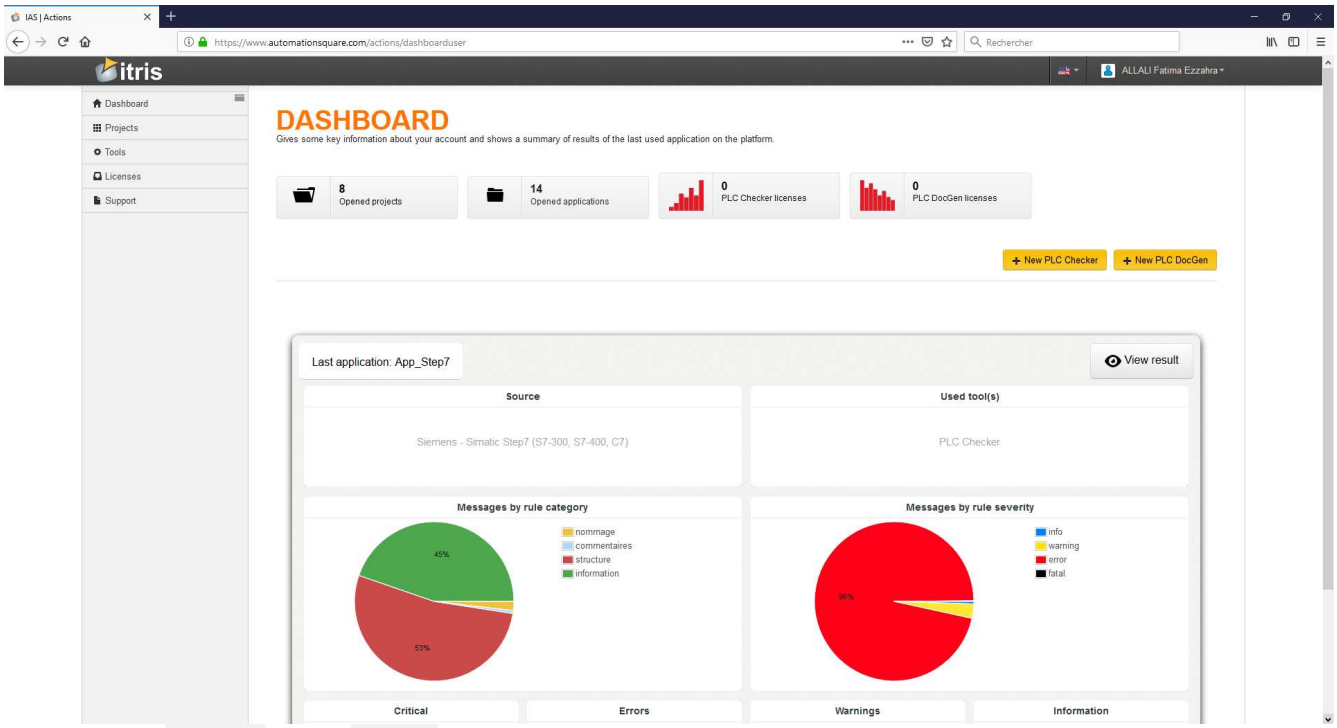


Figure 4.1: Dashboard

- Name the project from website (step1):

1. Give a name of the application and a name of the customer.

2. Some complementary informations can be specified for the application.

3. Finish the entry by clicking on "Forward" button.

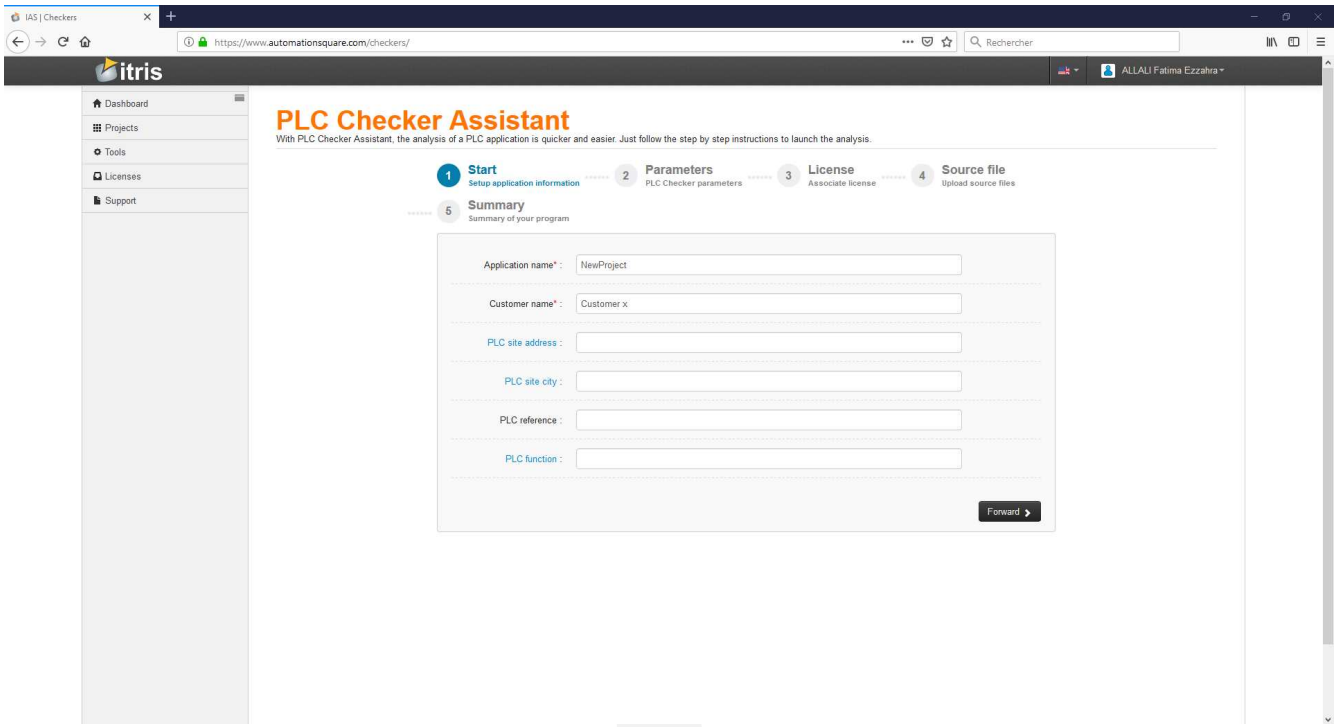


Figure 4.2: Project information

- Choice of the PLC type and the rules file (step 2).

Please select the Plc type from the list.

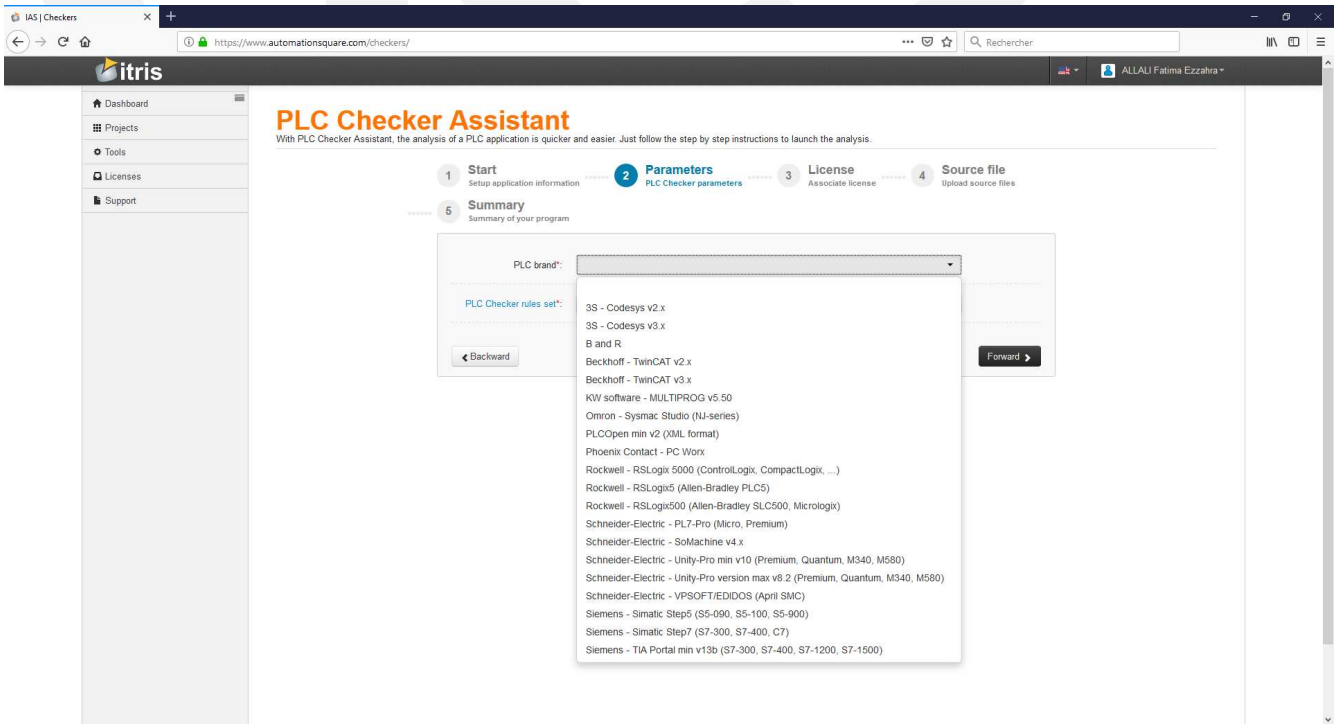


Figure 4.3: Choose the Plc type

Please select the rules file to use or choose "New upload in step 4 to add a new rules file in step 4.

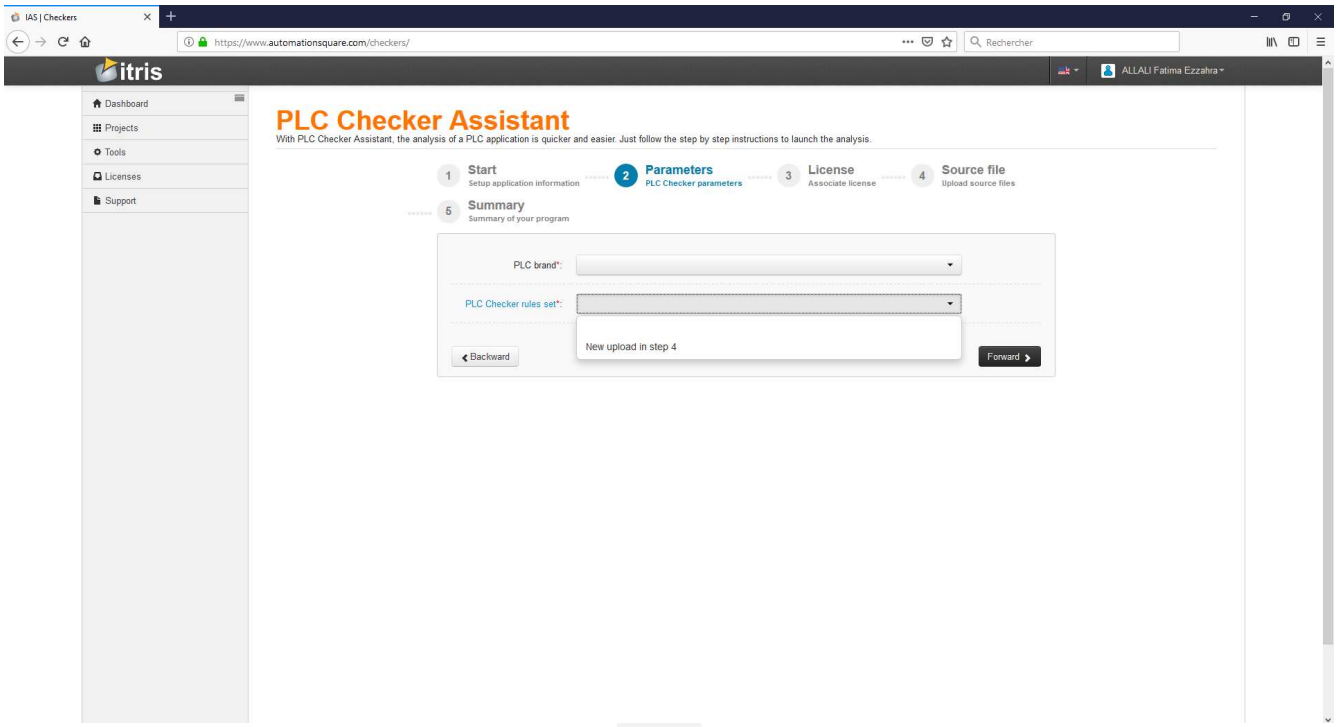


Figure 4.4: Choose the rules file

- Associate a right to the project from the website (step 3).

If you do not have any available rights for this action, you can still continue creating the application by pressing "Forward".

If you have rights available for this action, please select the license to associate with this application from the drop-down menu and press "Forward".

If you have a free available license, please enter its identifier in the "License #id" box and click on the "Next" button.

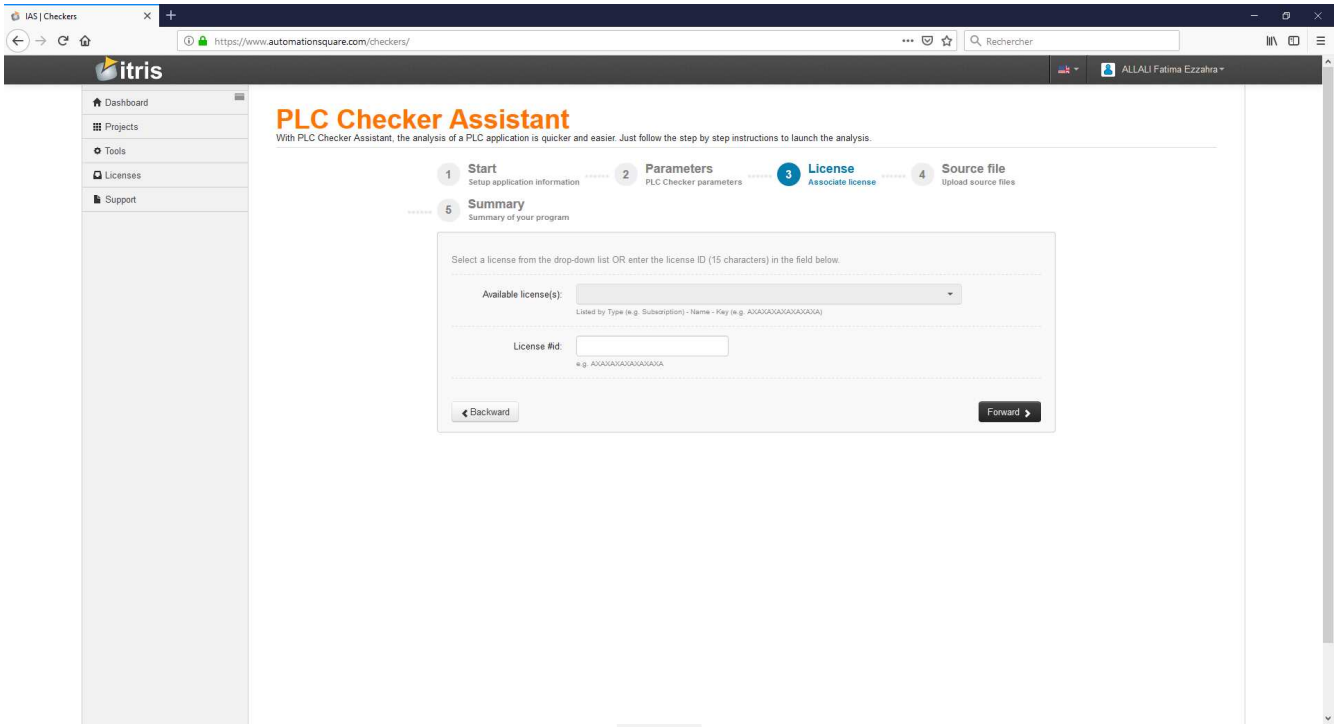


Figure 4.5: Association of license

- Load rule files and source files (step 4).

Please click on the "SELECT FILES . . . " button to import the rules file and the source files of the application and click "Forward" to validate. Depending on the type of PLC chosen, you have a list of authorized files. You can continue without adding files by clicking "Forward".

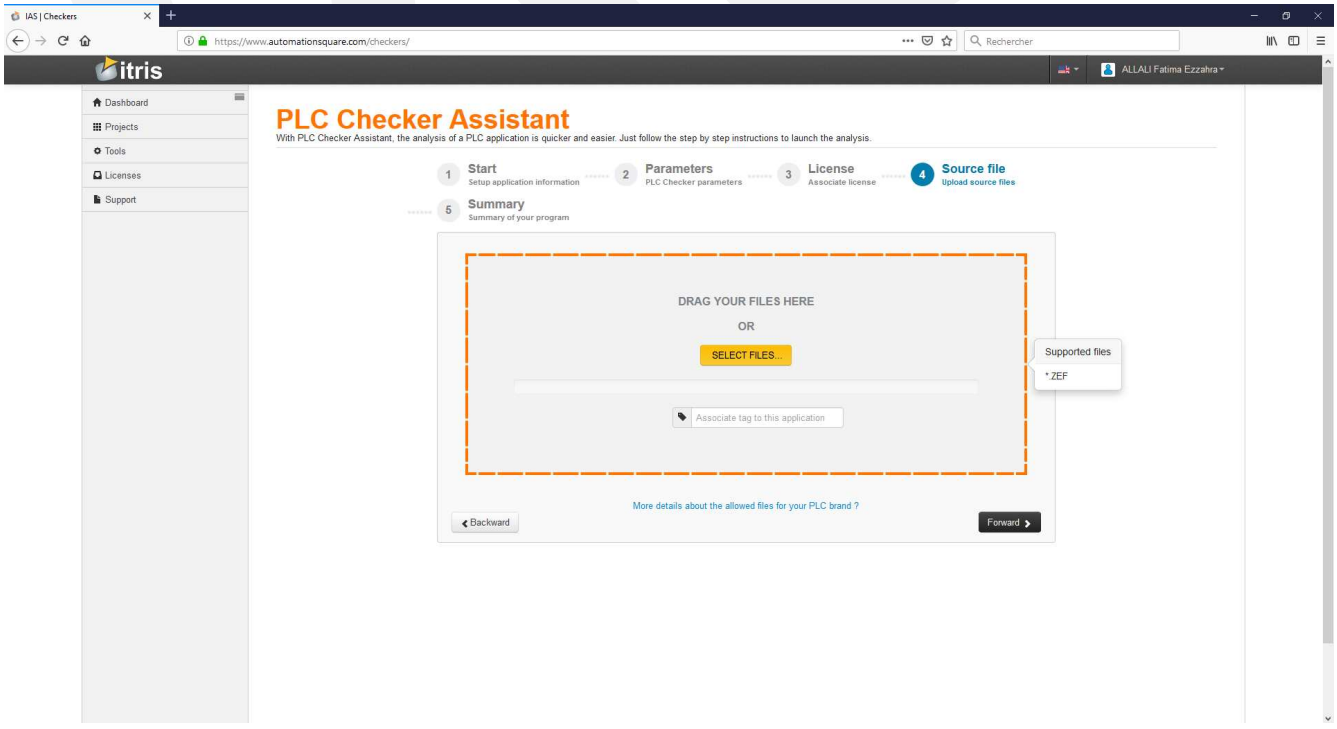


Figure 4.6: Selecting files

- Summary of the application (step 5).

In this step, you find the information you have entered on the application. To modify them, you can click on the "Backward" button, otherwise you click on "Create" to create the application or "launch" to create and launch Plc Checker.

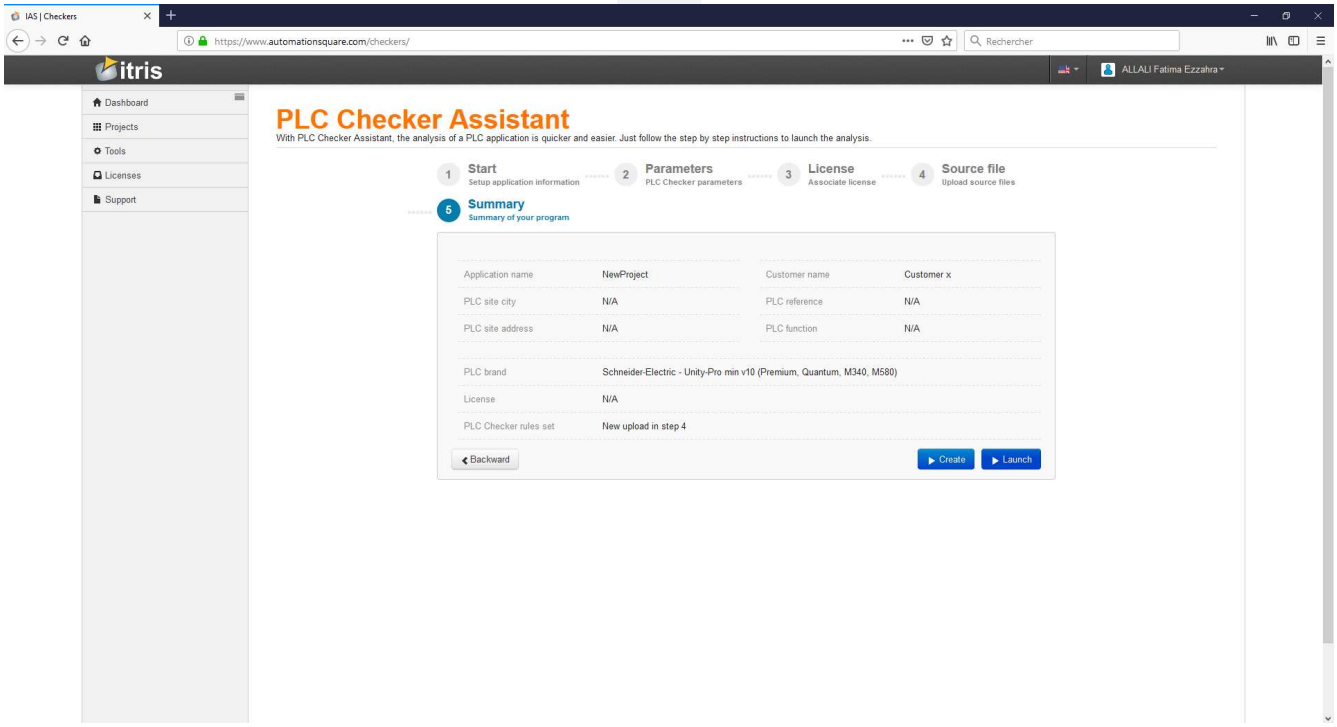


Figure 4.7: Summary

- Note the 6-character code of your program, it will be used for retrieval of project information by another user on the website or on the Eclipse platform of the PLC Checker.

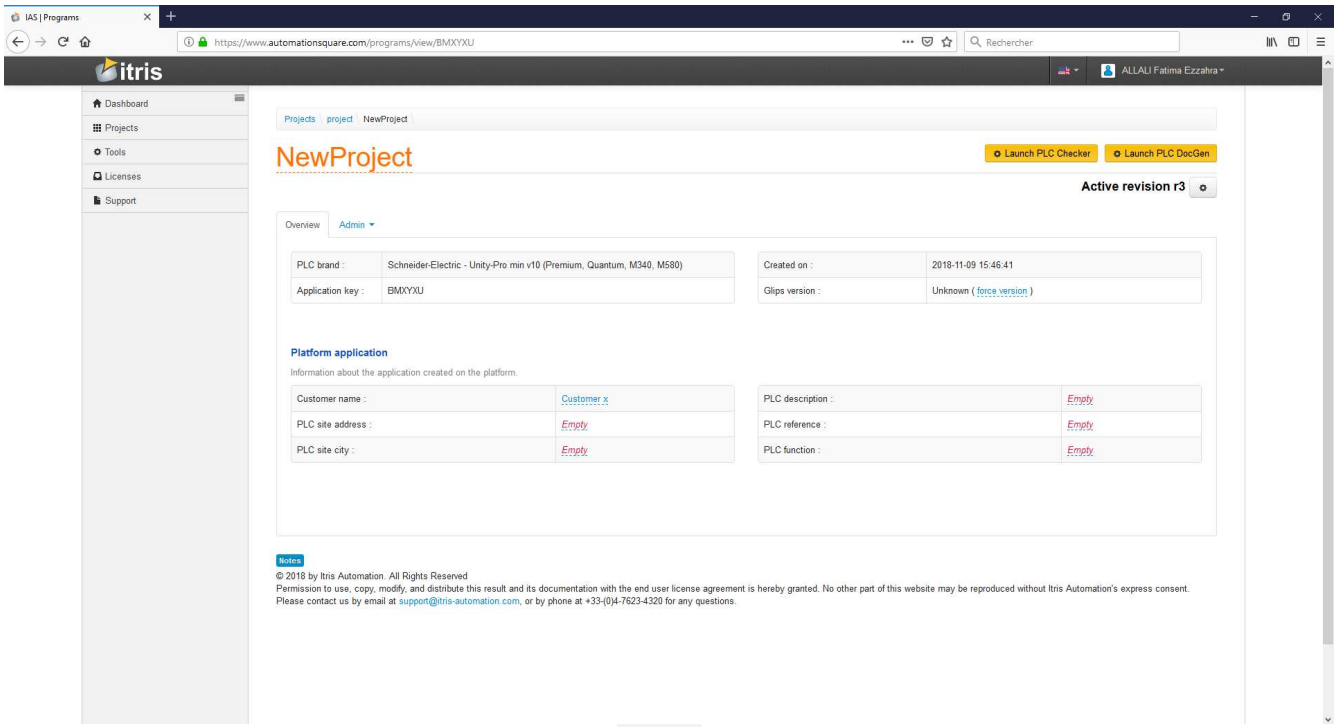


Figure 4.8: Code of the application

4.1.2 Creation with wizard of a PLC Checker application on the Itris Automation Square website (Plc Checker project to be defined)

- Access to projects page ("My projects" tab in the menu on the left).

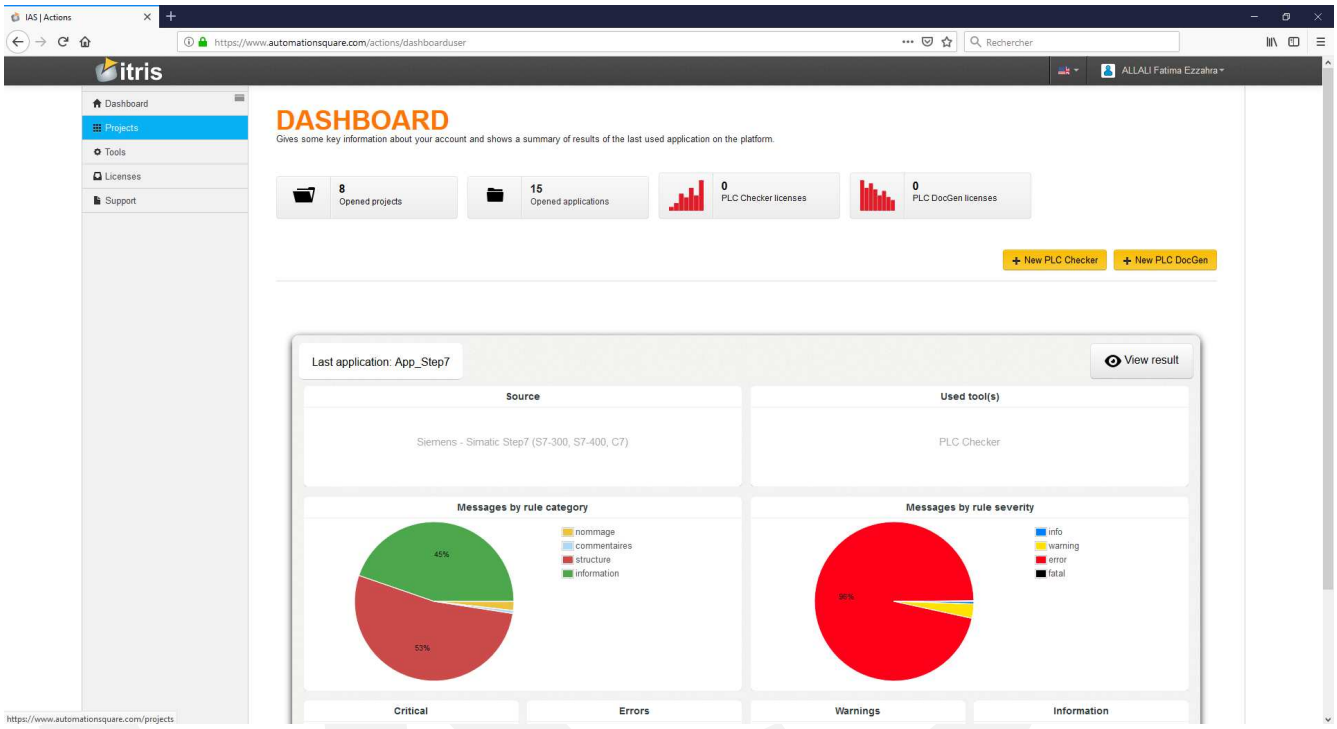


Figure 4.9: Go to my projects page

- Create a new project to contain the programs: please press Actions then New project

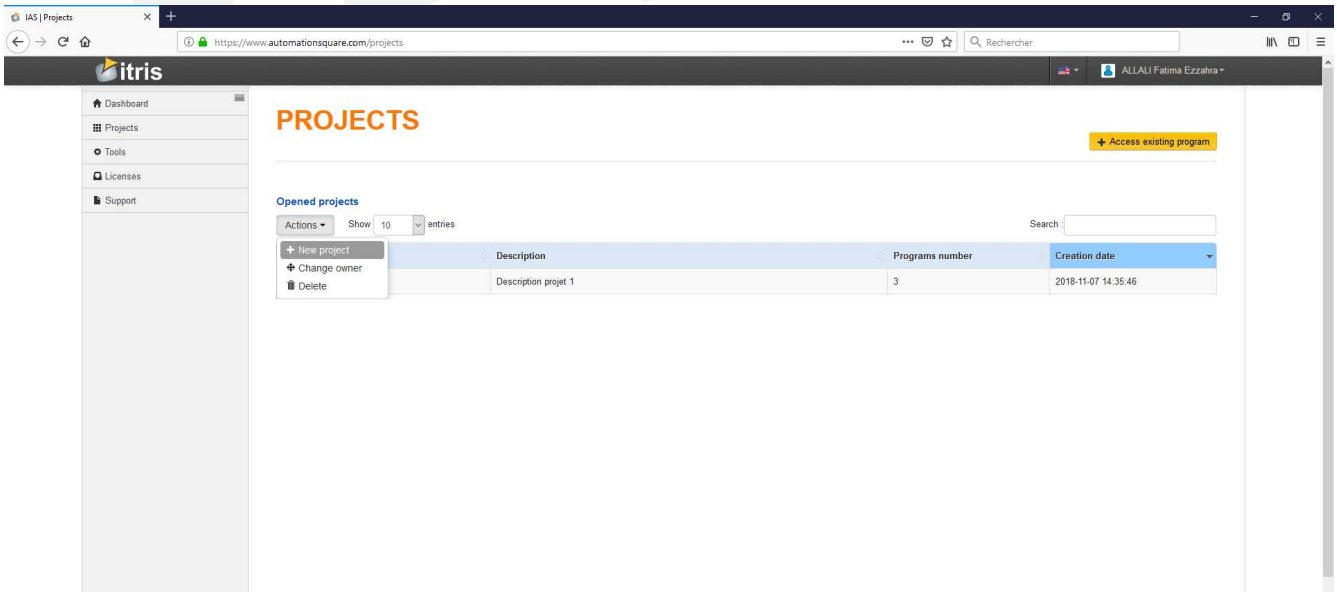


Figure 4.10: create a new project

- The page below allows you to enter a name and description of the project. Please click on "Add" to finish the entry.

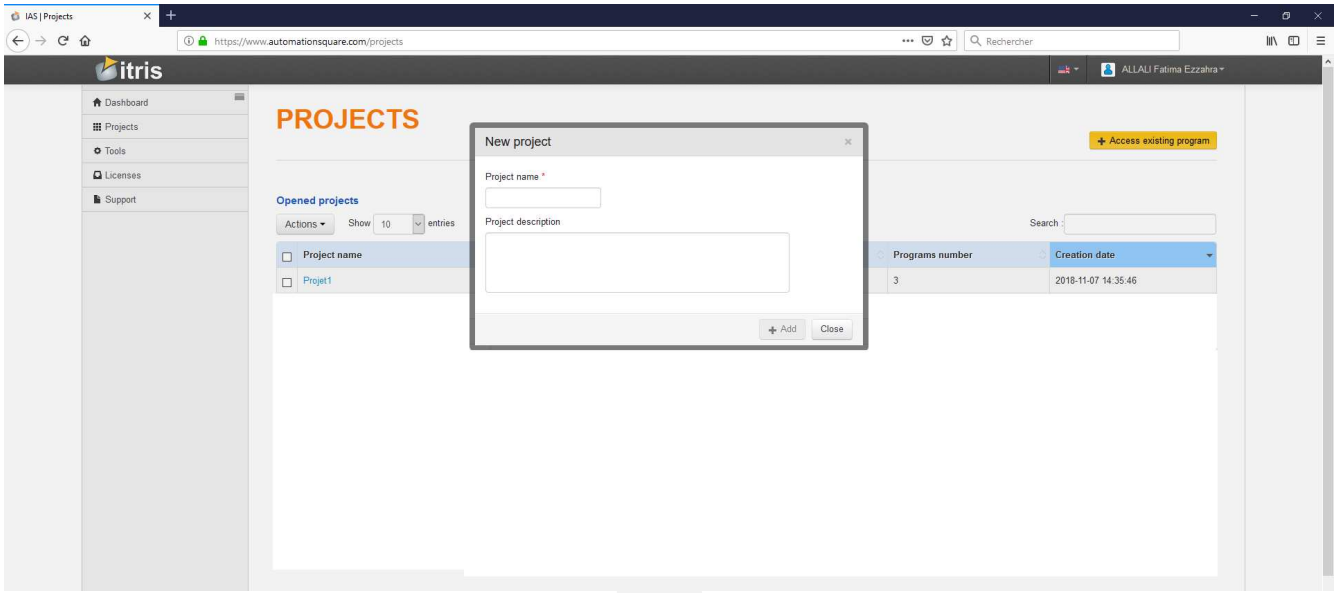


Figure 4.11: name and description of the project

- Go to the applications page ("Applications" tab)

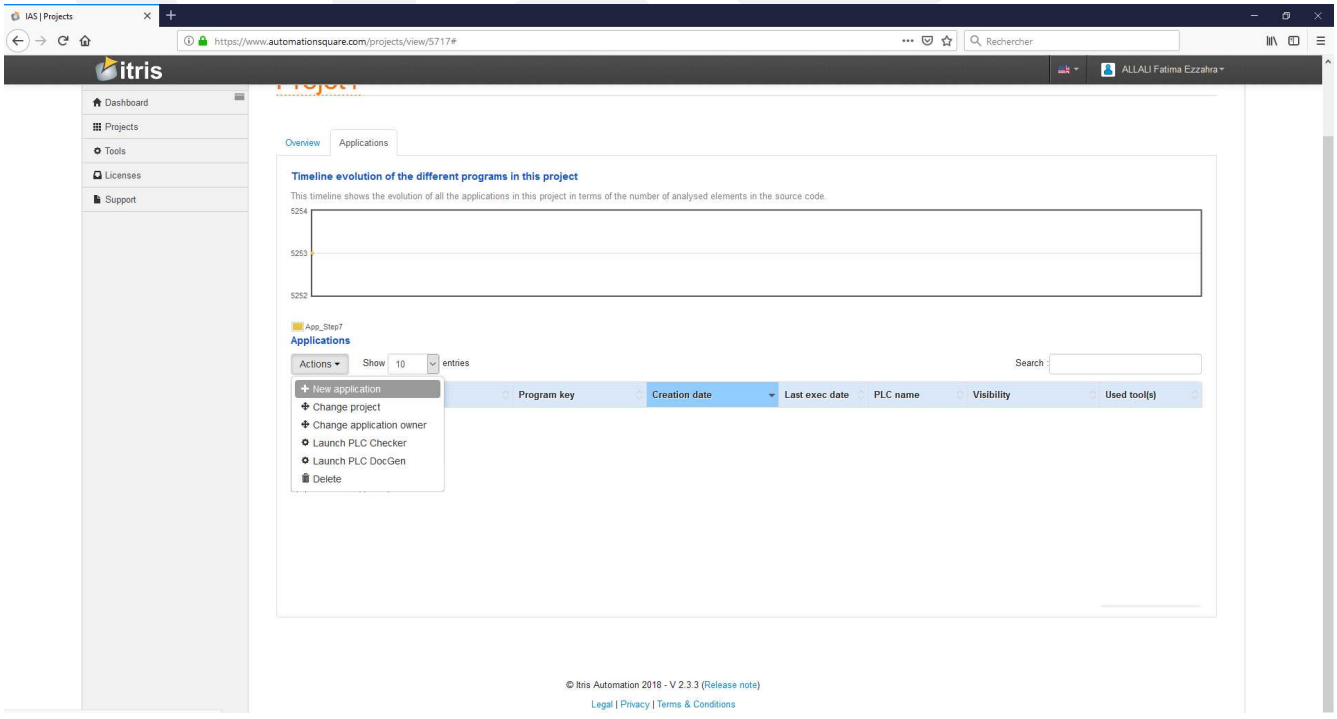


Figure 4.12: new application

When you click on "New application", please choose the tool (in this case Plc Checker) on the page that opens and validate by clicking on "Launch assistant".

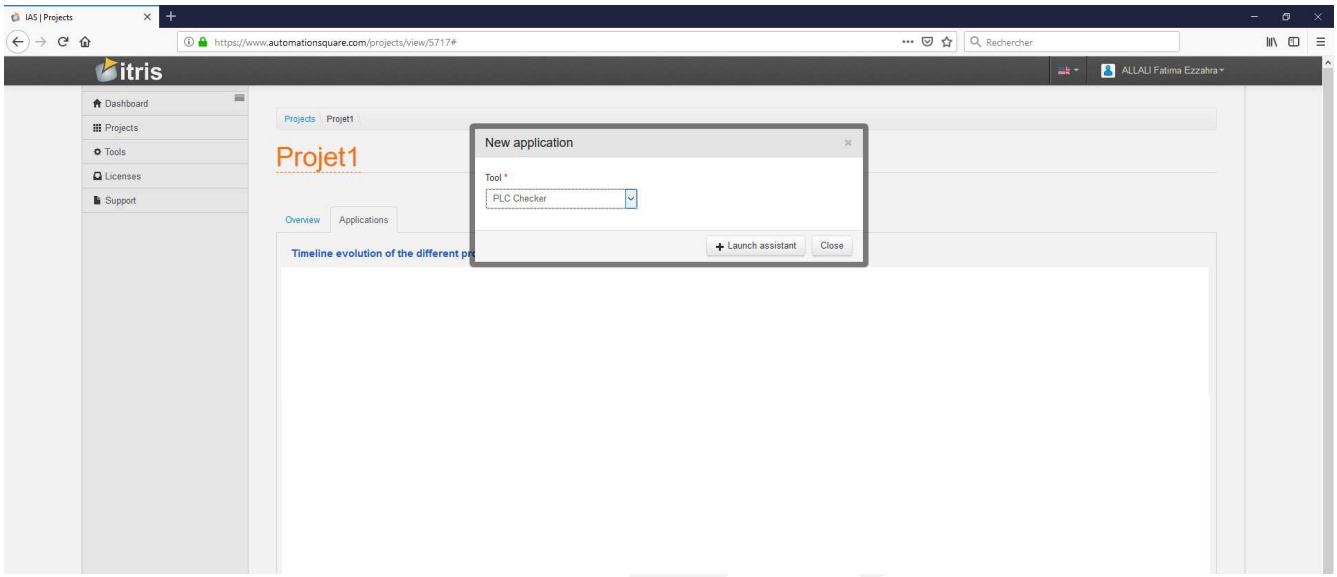


Figure 4.13: create a new application

By clicking on "Launch assistant, you will be directed to the page of the Plc Checker wizard detailed in the previous chapter Creation with wizard of a default PLC Checker application on Itris Automation Square website (Default Plc Checker project

4.1.3 Access to an existing PLC Checker program on the Itris Automation Square website

- Access the projects page ("My projects" tab in the menu on the left).

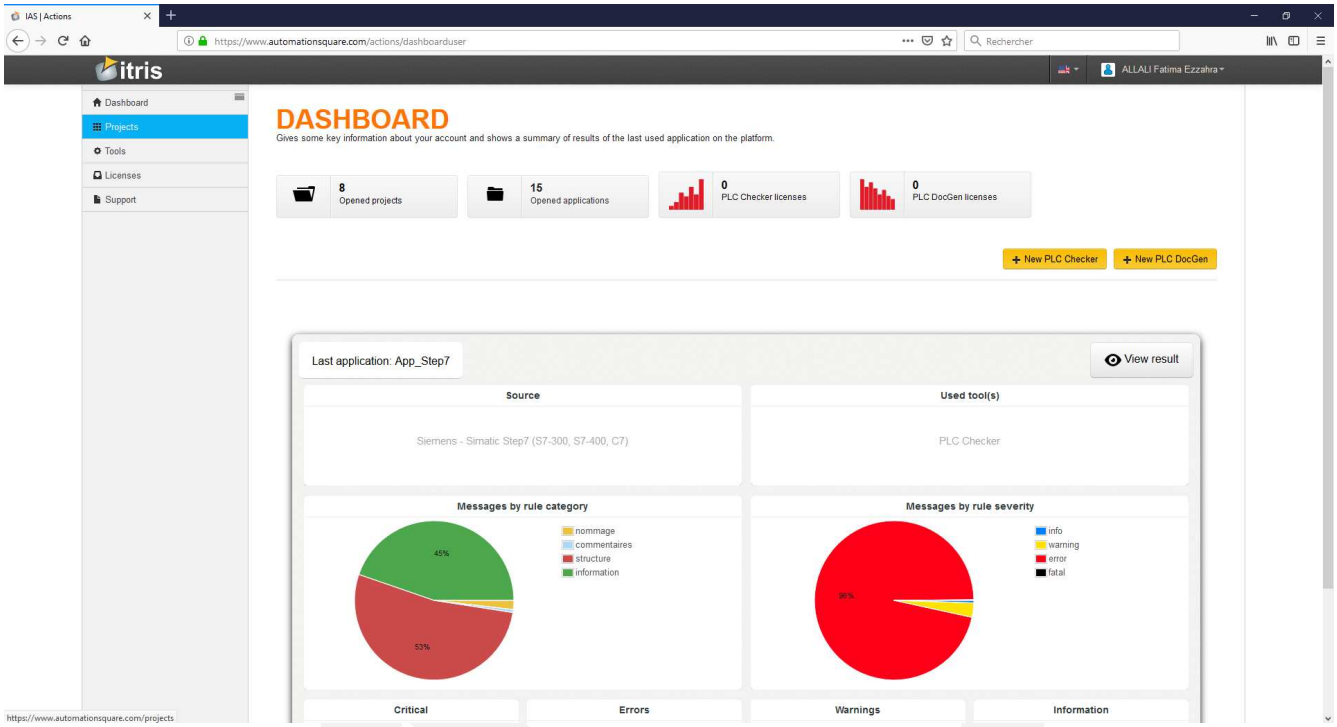


Figure 4.14: Go to the projects page

- Enter the program key of the existing program by clicking on "Access existing program"

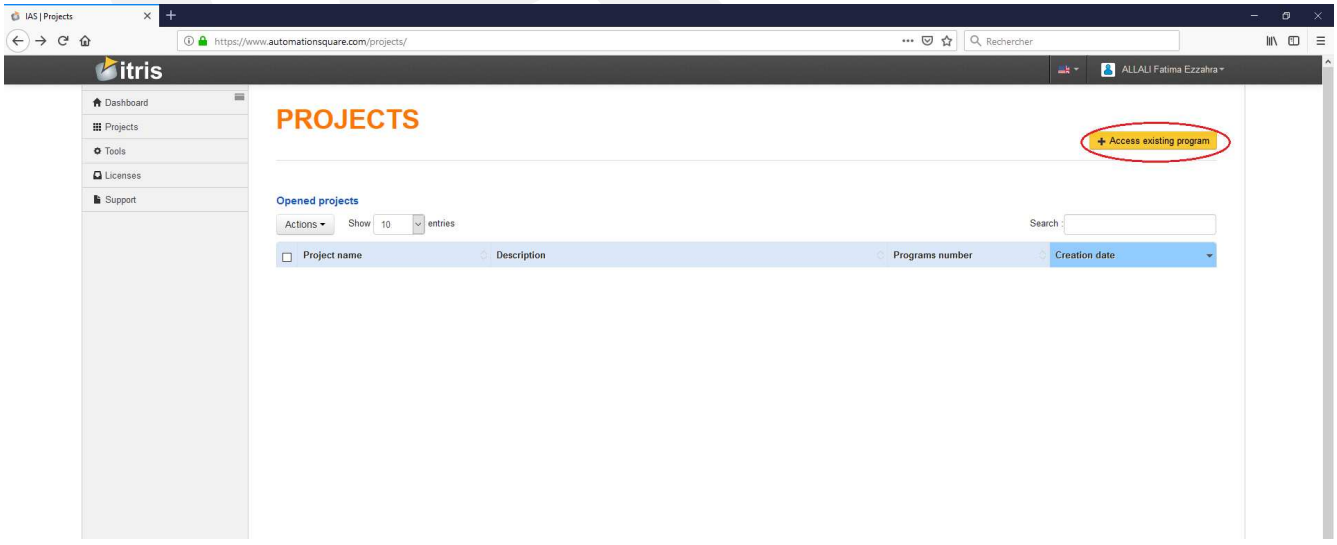


Figure 4.15: Access existing program

Please enter the program key in the "Program key" box and click on "Access program". The program is then added to the "Invited programs" table.

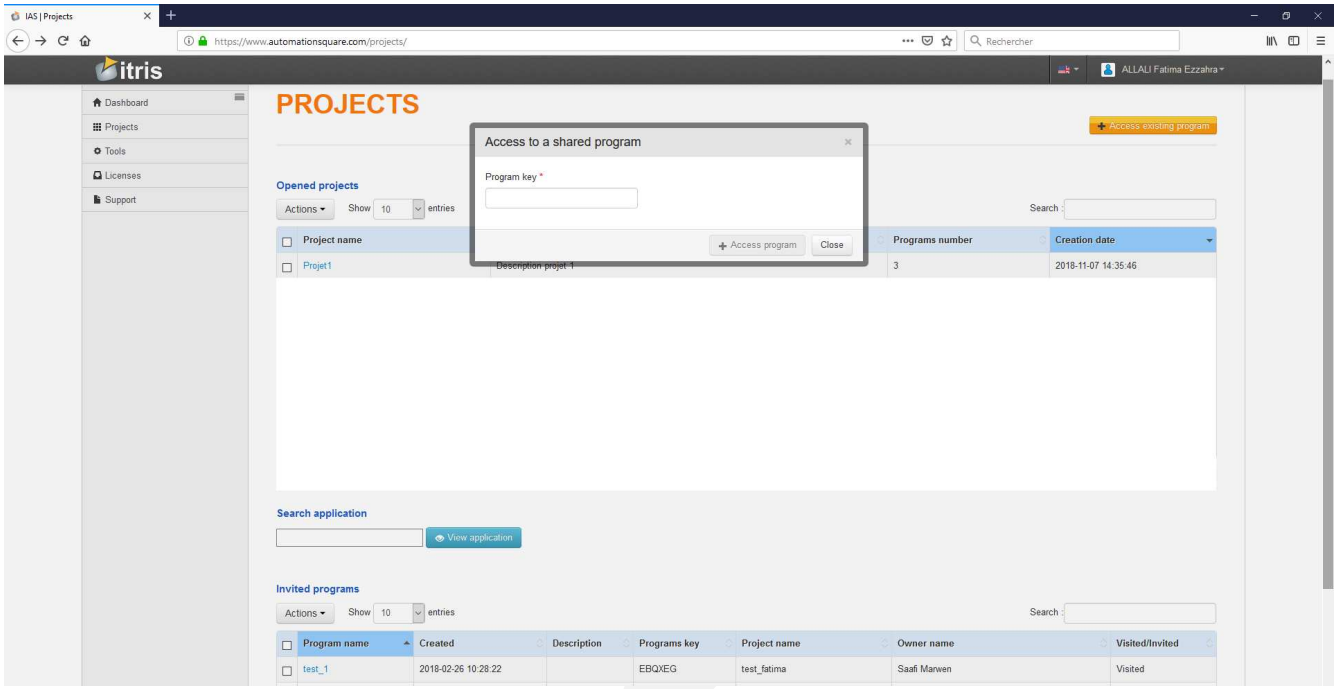


Figure 4.16: Enter the program key

4.1.4 New PLC Checker project creation on Eclipse platform

- Create a PLC Checker project in the Eclipse platform, then click on **Next** button

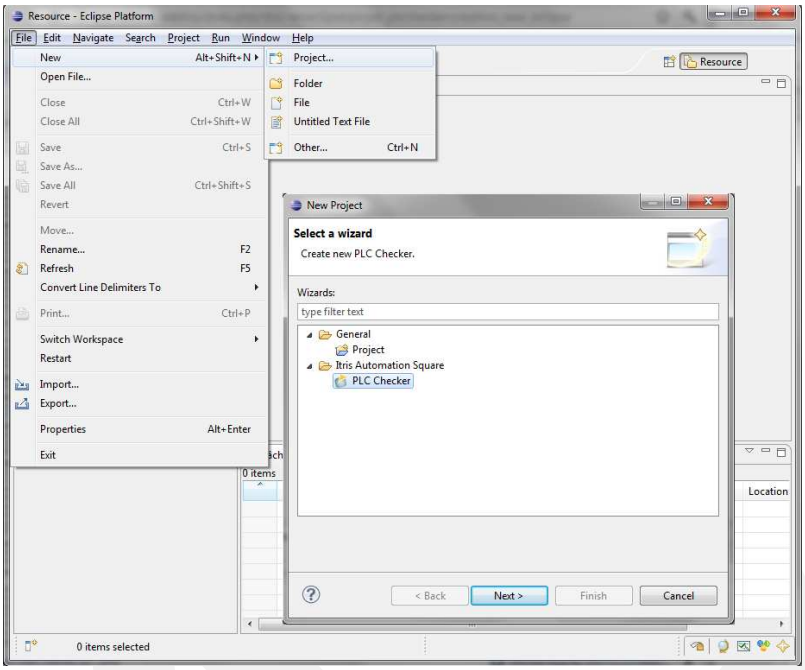


Figure 4.17: New project

- In the **New PLC Program** field, choose the PLC type (PLCOpen, Codesys, TIAPortal, Step7, Step5, Unity-Pro, PL7-Pro, RSLogix5000, SCL500, PLC5, SMC, . . .)

- Enter the name of the project and the name and description of the program inside this project.

- Associate a right to the program if available:

  - If you do not already have license rights for this action, please use the rights order interface or contact the commercial@itris-automation.com.

  - If you have rights for this action, they are displayed **IAS Right** drop-down list. You can then select the right you want to associate to this program.

  - You can let the **IAS Right** field empty, in order to just create the project and program structure. But a right will be asked later if an analyze is required by a user on this program directory.

- Click on **Finish** button. The platform ask to the server for a project and program creation and then the platform creates in your workspace a local copy of the project and program directory from the server.

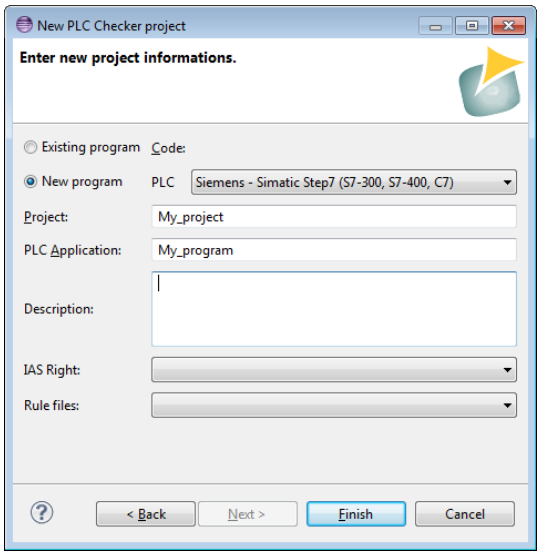


Figure 4.18: New PLC Checker project

4.1.5 Creation of copy of existing PLC Checker project on Eclipse platform

- Create a PLC Checker project in the Eclipse platform, that gets the content and characteristics of the project previously created on the website
- On the Eclipse platform, select a new PLC Checker project and then click on the **Next** button.

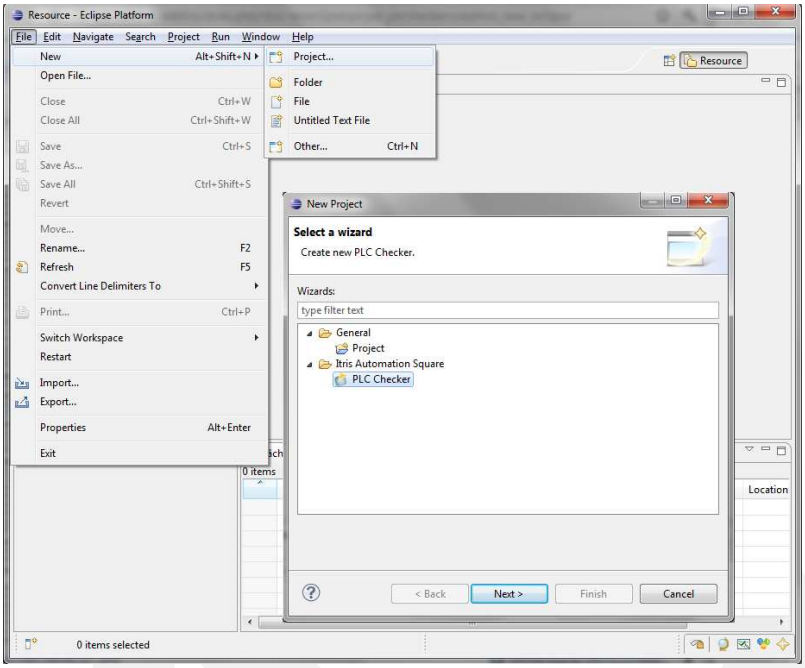


Figure 4.19: New project

- Enter the program code of your project in the **Code** field. Click on the **Finish** button. The platform create on your workspace a folder named like the project defined on the website. The platform also copy in that folder the files contained in you project on the IAS server.

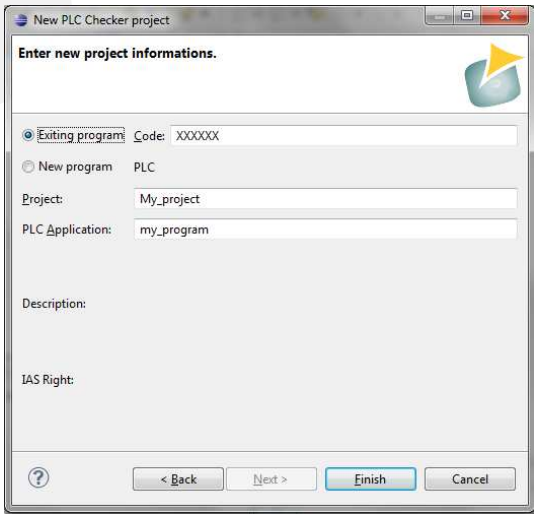


Figure 4.20: Enter program code

4.2 Run PLC Checker

4.2.1 Run PLC Checker from an Eclipse platform :

- Import application file(s) in your workspace.

  1. Use the file **explorer** of your OS[1] to **copy** the application export file(s) to be used.
  2. **Past** this(those) file(s) in your project folder on the **Navigator** view of Eclipse platform.

- Import rules file in your workspace.

  1. Use the file **explorer** of your OS[1] to **copy** the rules file to be used.
  2. **Past** this file in your project folder on the **Navigator** view of Eclipse platform.

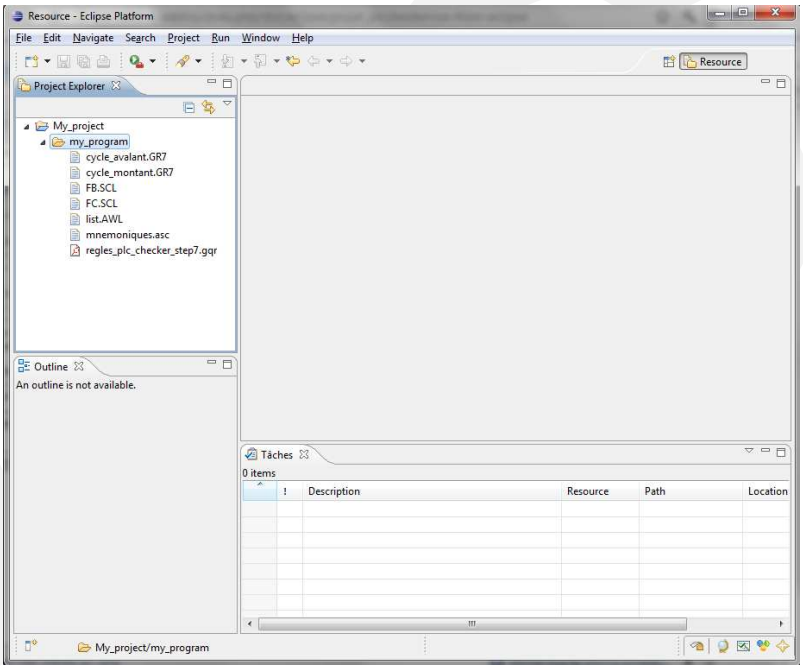


Figure 4.21: Project folder view by Eclipse Navigator

- Run PLC Checker from the Eclipse **Navigator**.

  1. **Select the application export file(s)** (*.FEF pour PL7PRO, *.ASC, *.AWL, *.GR7 et *.SCL pour STEP7, *.XEF pour UNITY-PRO, *.L5K pour RSLogix5000) to check and the **rules file** (*.GQR) to be used to.
  2. Open **right click** contextual menu
  3. Launch the analyze using the **GLIPS tools / Run PLC Checker for . . .**

[1] Operating System

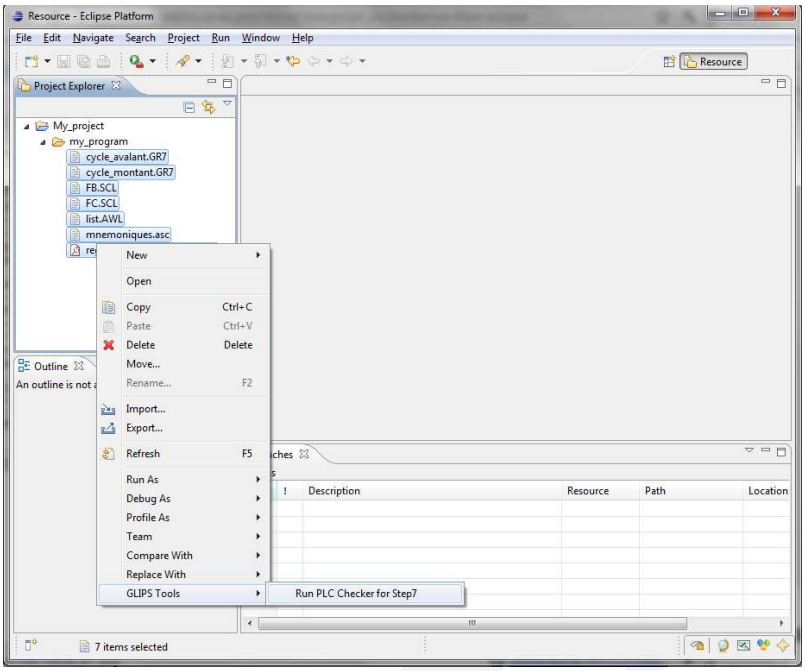


Figure 4.22: Launch PLC Checker via Eclipse platform

- If some files contains unsaved modifications, then the following dialog box appears.
  - Click on **Deselect All**, if you DO NOT want to take account of the unsaved modifications for this analyze. Note that the unsaved modifications wont be deleted.
  - If you want to save modifications before to launch the analyze, then click on **Select all**.
  - Click on **OK** to continue. Click to "Cancel" to cancel the analyze.

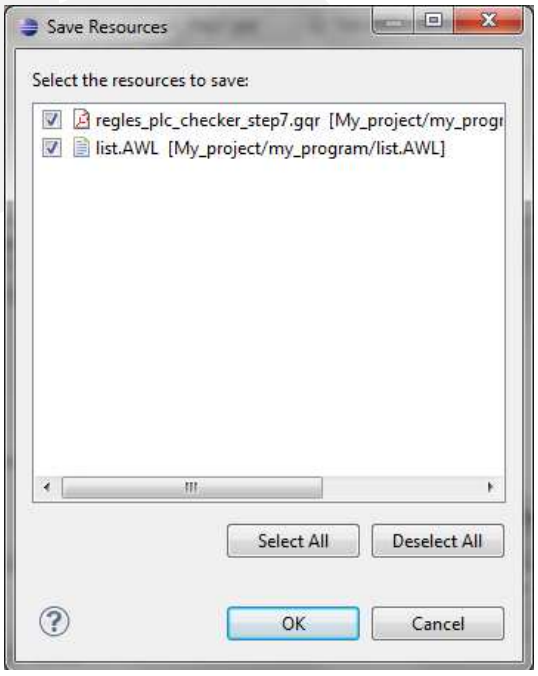


Figure 4.23: Save files modifications before the analyze

- A dialog box opens to save the result file (. gqre). We have the ability to rename it or change its directory.

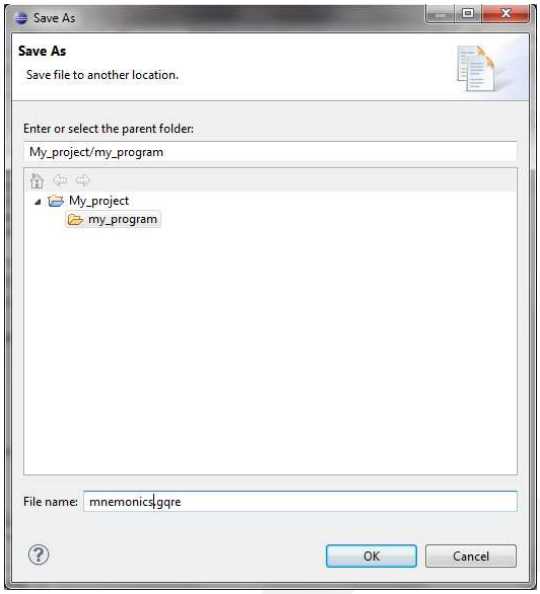


Figure 4.24: Save rules file modifications before the analyze

- During the analyze, a dialog box allow you see progression and to "Cancel" execution. The analyze phases are:
  - Sent files (application and rules) to the IAS server,
  - Read application file,
  - Analyze application file following rules file,
  - Update results,
  - create results file on Eclipse workspace,
  - Open results file with result viewer embedded in the Eclipse platform.

4.2.2 Run PLC Checker from the website:

Please click on "Launch PLC Checker"

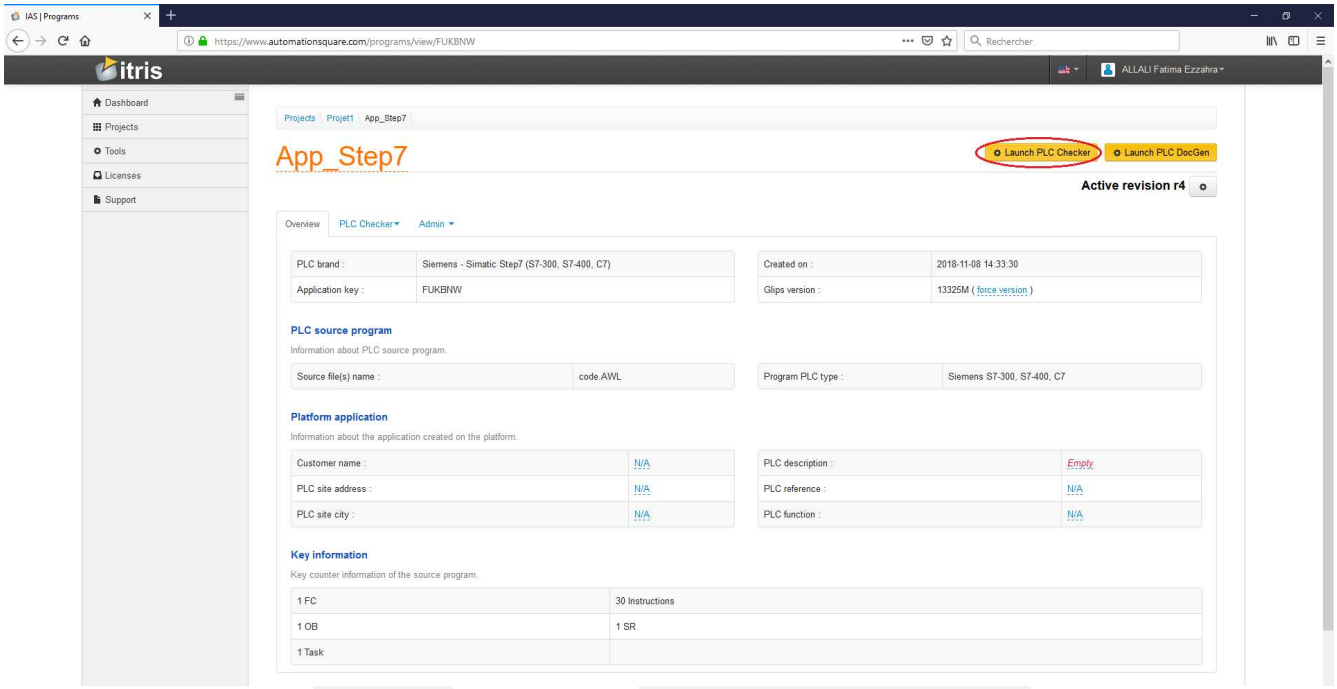


Figure 4.25: Plc Checker execution button

- If you have already downloaded the necessary files for execution (Source files and rules file), please click on "Launch" to start the Plc Checker analysis.

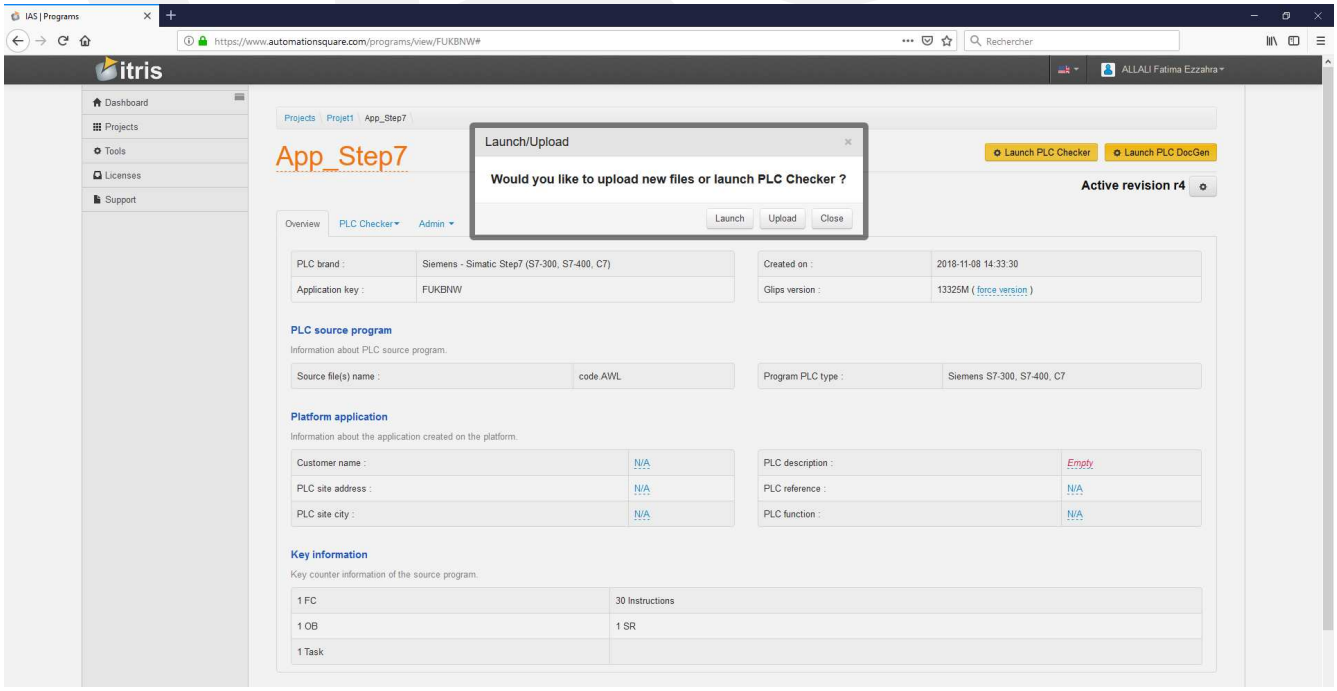


Figure 4.26: Plc Checker execution page

- Otherwise, please click "Upload" to go to step 4 of the Plc Checker wizard. At the last step of the wizard (step 5), you can start the analysis by clicking on "Launch"

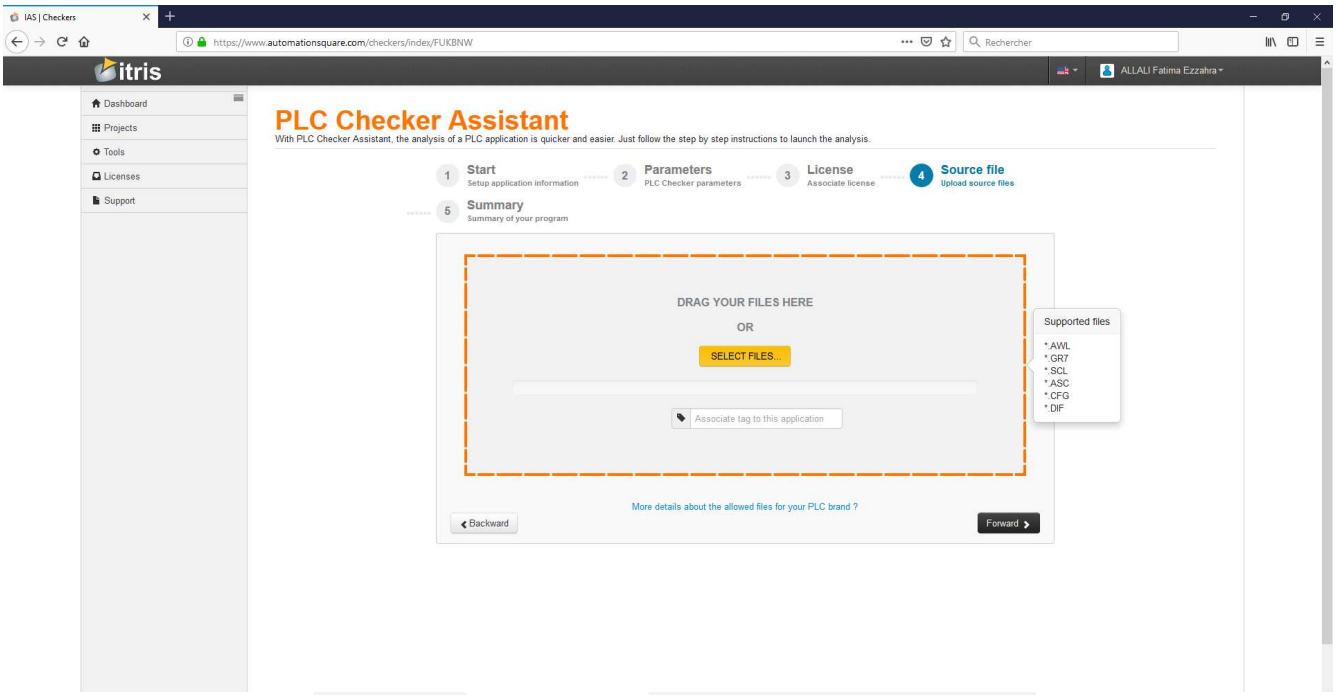


Figure 4.27: Step 4 of download

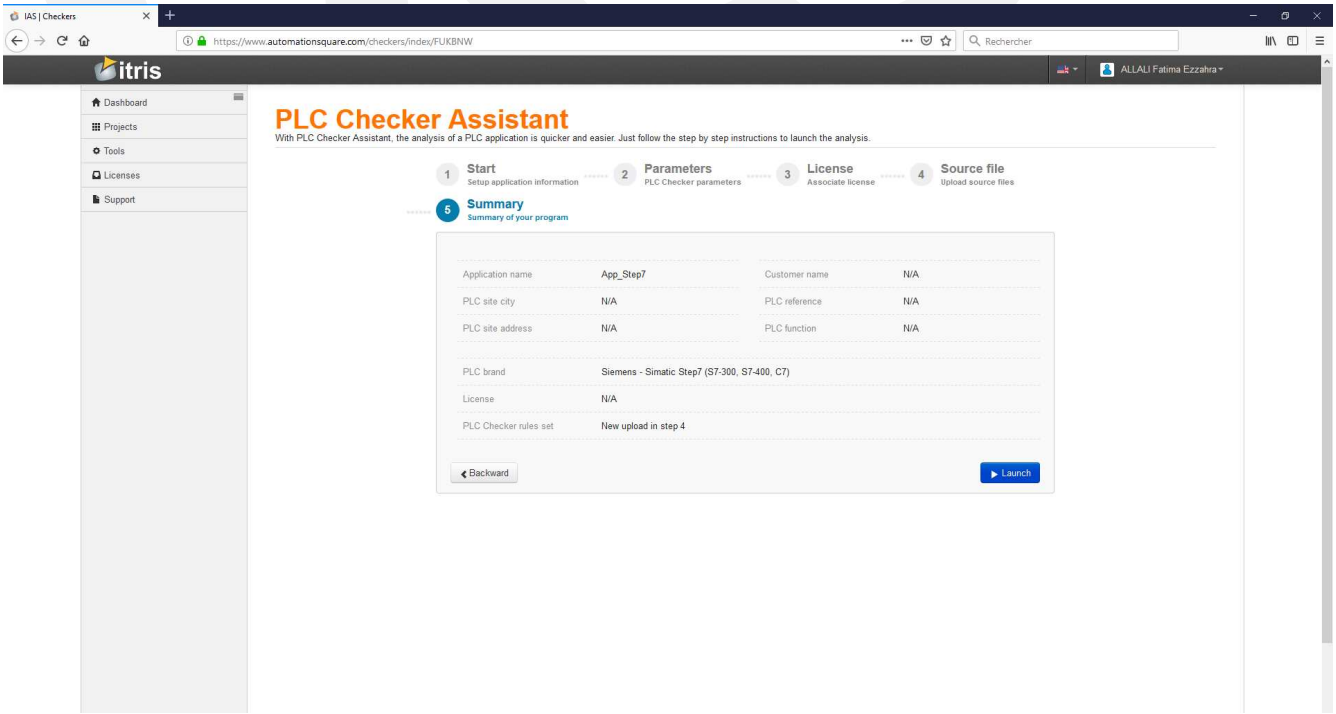


Figure 4.28: Run PLC Checker step5

4.3 View PLC Checker results

4.3.1 View results on Eclipse platform

- Get last action results (**ONLY in the case of existing PLC Checker project creation on Eclipse platform**):

1. **create a new project** on the Eclipse platform (see Existing PLC Checker project creation on Eclipse platform) getting files and characteristics of the project located on IAS server and previously created via the website.
2. **Select the rules file**
3. open the **right-click** contextual menu
4. Get the last action result using this contextual menu: **GLIPS tools / Get last action results**

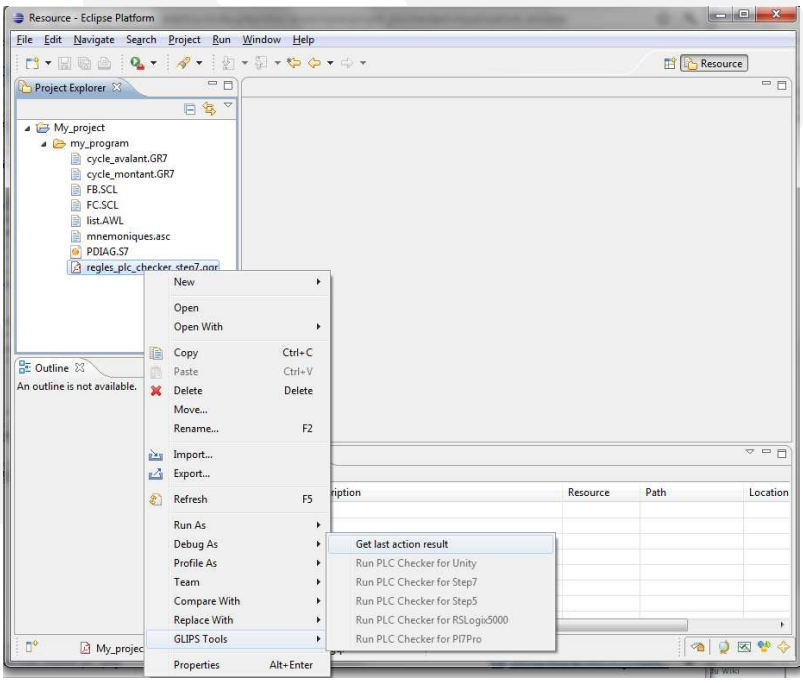


Figure 4.29: Get last action results

- View result file (*.gqre) on Eclipse platform.

  – When asked to, by the Eclipse platform, you have to choose the result file name. Click on **OK**.

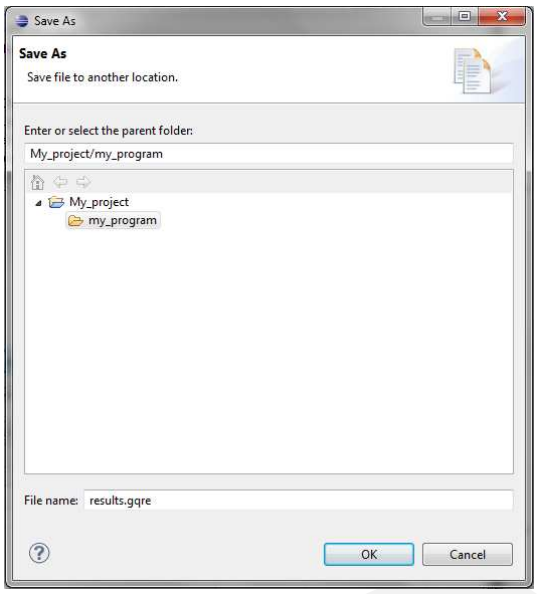


Figure 4.30: Result file name

- If the result file already exists the following dialog box appears.
  - Click on **Yes** to delete the existing result file.
  - Click on **No** to go back to the result file name definition.
  - Click on **Cancel** to cancel getting results.
- The result file (*.gqre) is displayed on the result viewer embedded in the Eclipse platform.

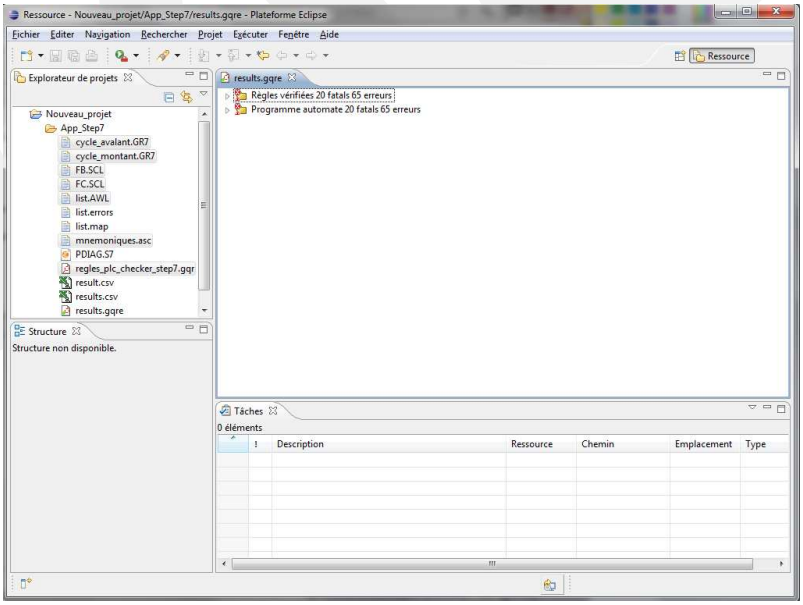


Figure 4.31: Result File

4.3.2 View results on the website

- After launching Plc Checker, a tab with the name "PLC Checker" is added to the application page.

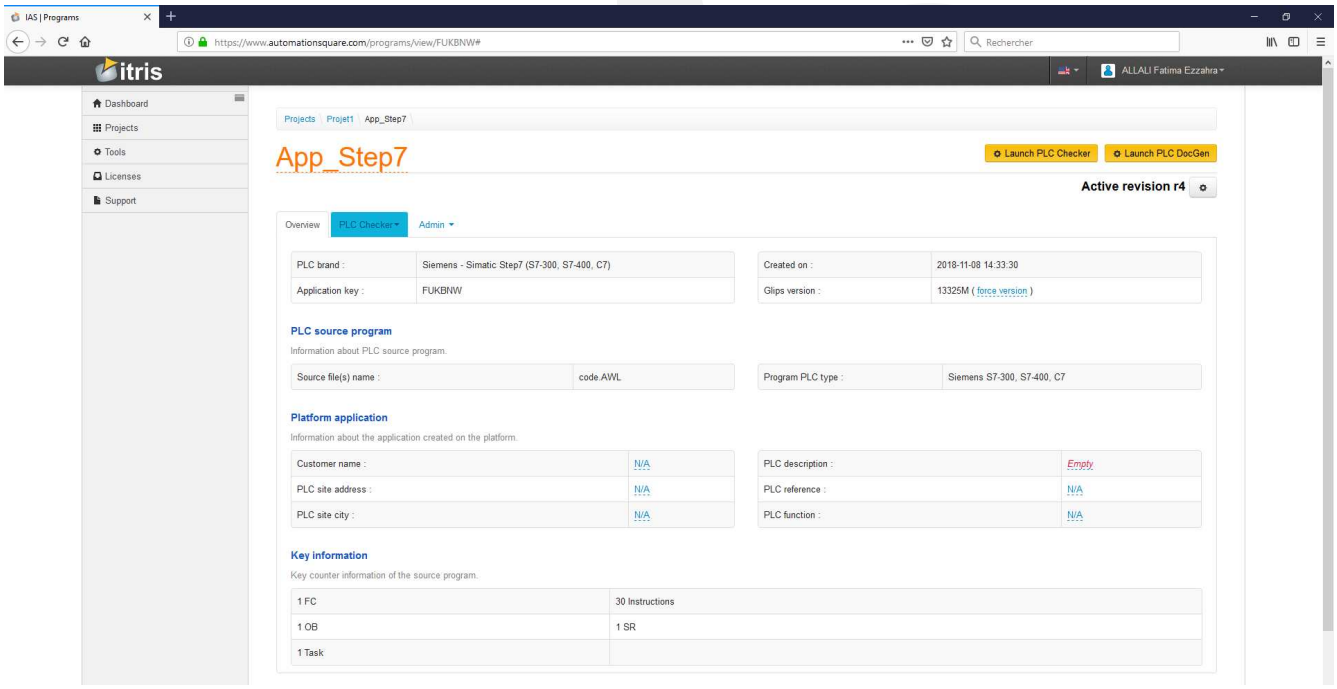


Figure 4.32: PLC Checker Tab

- For an analysis of the results, please click on PLC Checker / Dashboard. You can find graphs showing the evolution of the results between the different analyzes and a classification of the results by category of rule and by severity.

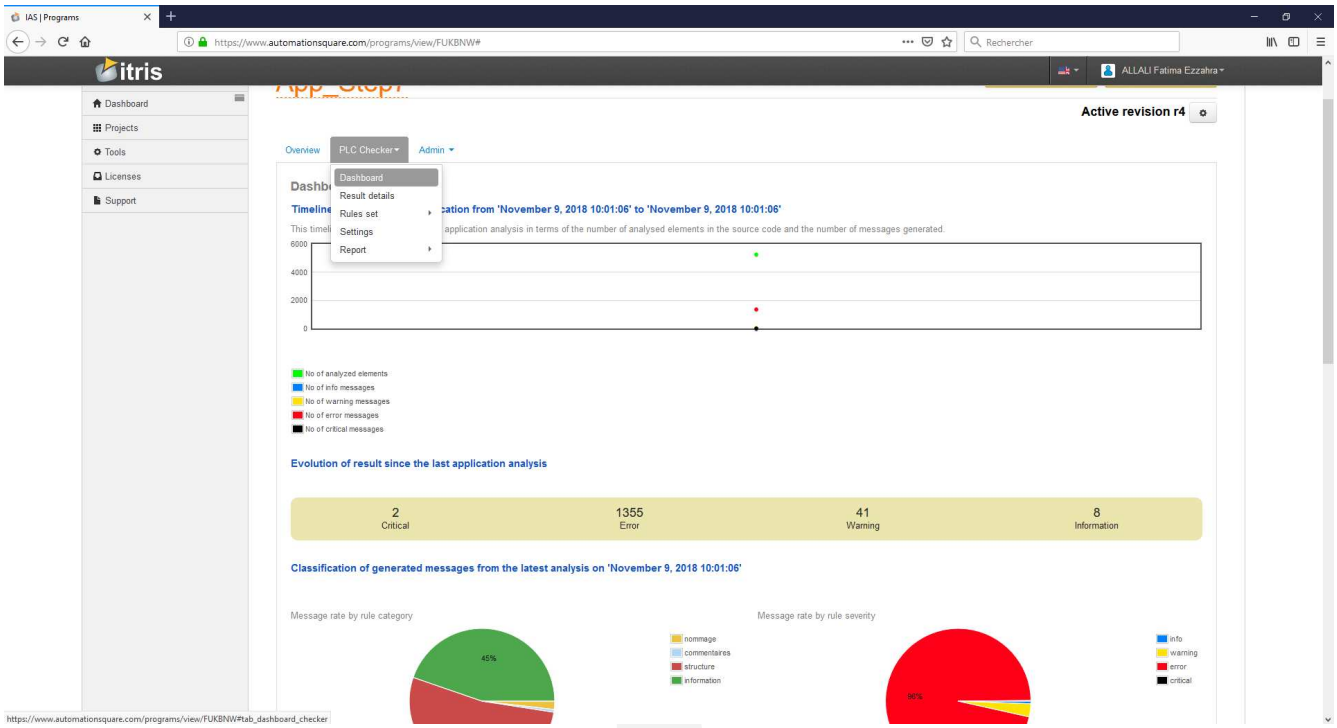


Figure 4.33: Dashboard

- To view detailed results, please click on PLC Checker / Result details. you find the list of results by rule category.

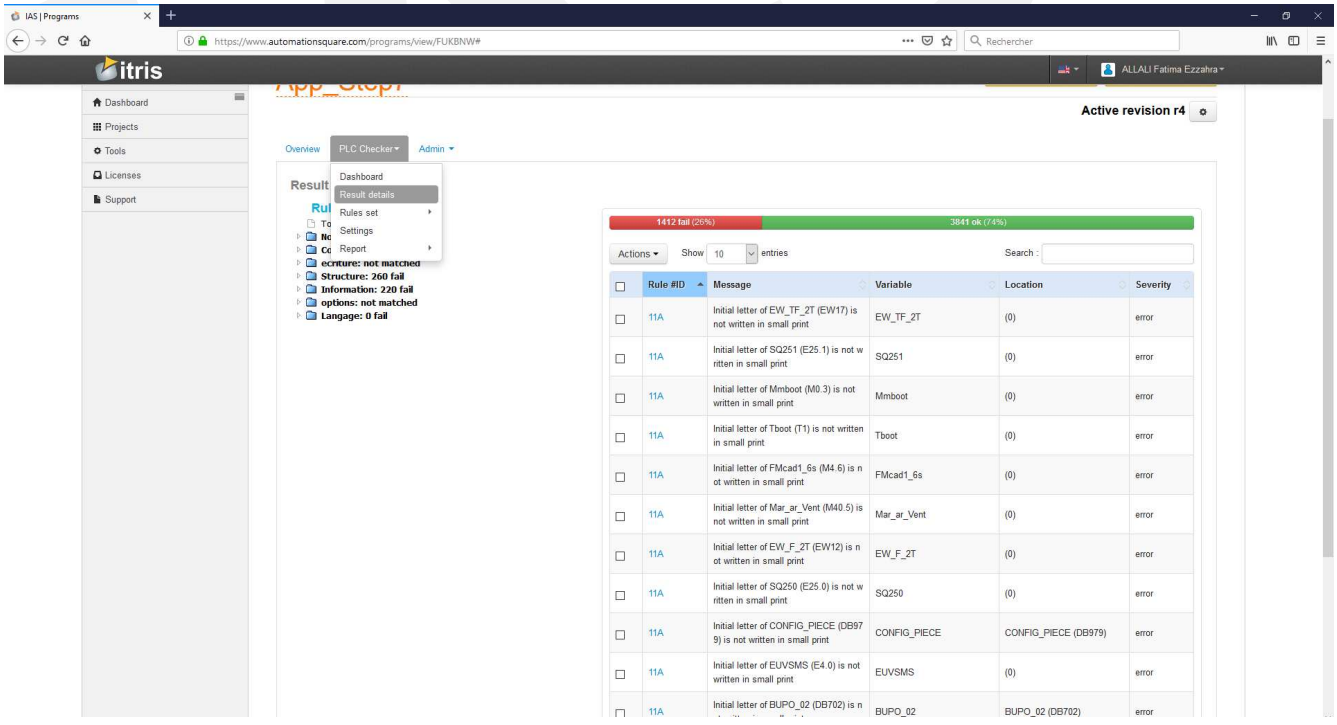


Figure 4.34: Result details

Chapter 5

PLC export files generation

5.1 FEF export file generation from PL7-Pro

PL7-Pro application must be exported in a FEF export file in order to be analyzed by PLC Checker.
You can create an FEF export file with the **File** > **Export application...** menu in PL7-Pro software.

1. In PL7-Pro menu, select **File** > **Export application...**
2. In the "Export application" dialog box, give a **name** to the FEF file
3. If needed, change directory where the FEF file will be created,
4. and click on **Save**

Before to be analyzed by the PLC Checker, the FEF file must be imported in the Eclipse platform workspace.

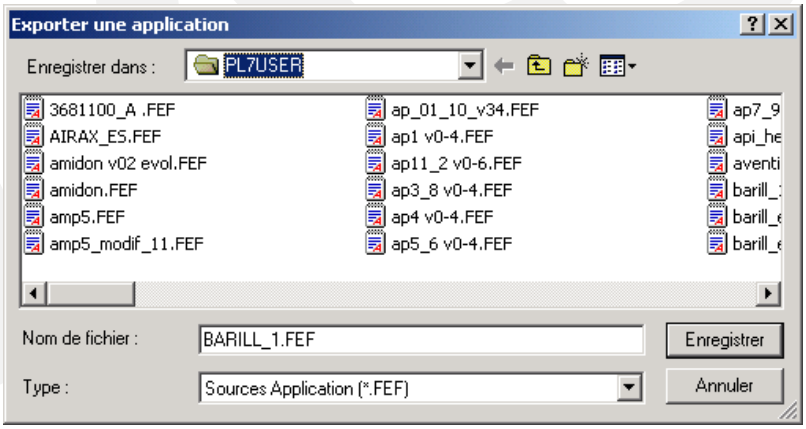


Figure 5.1: Export FEF file

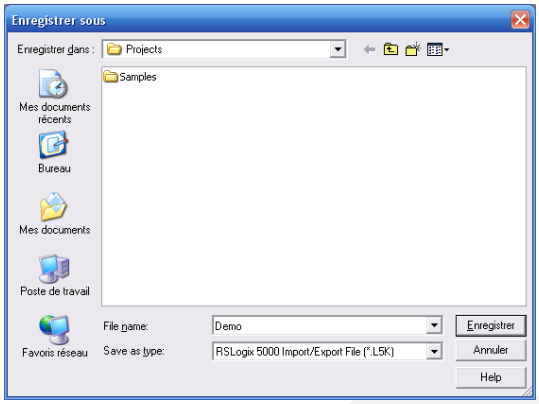
5.2 L5K export file generation from RSLogix5000

RSLogix5000 application must be exported in a L5K export file in order to be analyzed by PLC Checker.
You can create an L5K export file with the **File** > **Save as...** menu in RSLogix5000 software.

1. In RSLogix5000 menu, select **File** > **Save as...**
2. In the "Save as" dialog box, give a **name** to the L5K file
3. choose the file type : **RSLogix 5000 Import/Export File (*.L5K)**

4. if needed, change directory where the L5K file will be created
5. and click on **Save**.

Before to be analyzed by the PLC Checker, the L5K file must be imported in the Eclipse platform workspace.

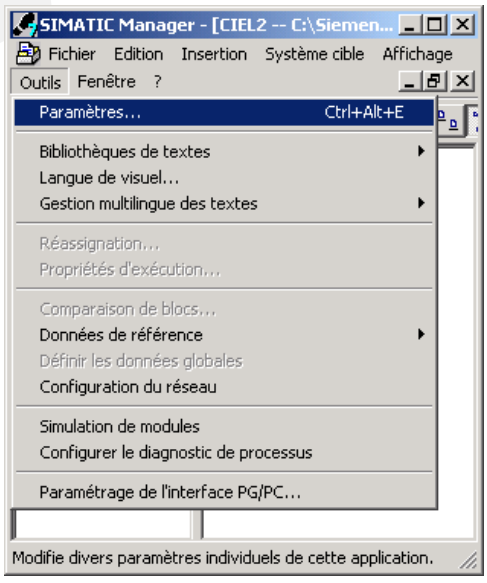


5.3 ASC, AWL, GR7 and SCL Export files generation from Step7

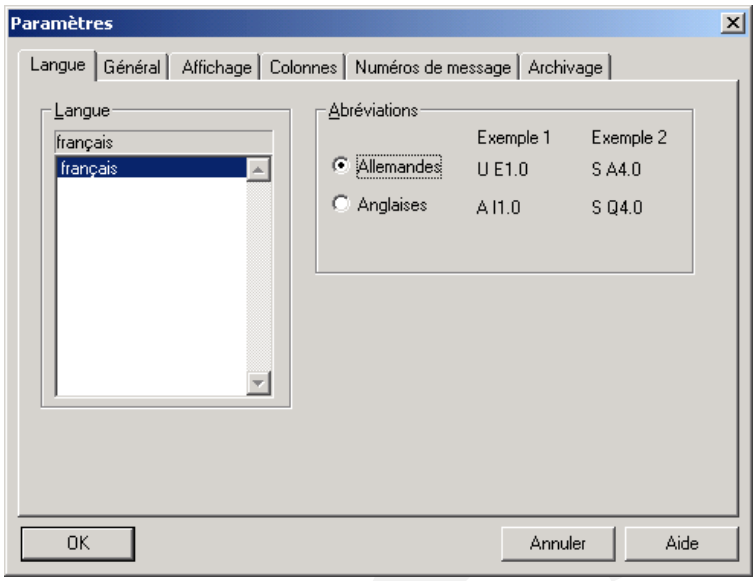
To analyze a STEP7 application with PLC Checker, you have to:

- create and export the symbols of this application in an **.ASC** file, using the German abbreviations,
- create and export the contact code (CONT) in an **.AWL** file,
- create and export all graph blocks (GRAPH) in **.GR7** files,
- compile CFC codes, if any, to create the corresponding **SCL** sources,
- export structural codes (SCL) in **.SCL** files.

To create the export files, you have first to insure that your STEP7 workshop uses the German abbreviations.(menu **Tools** > **Parameters...**)

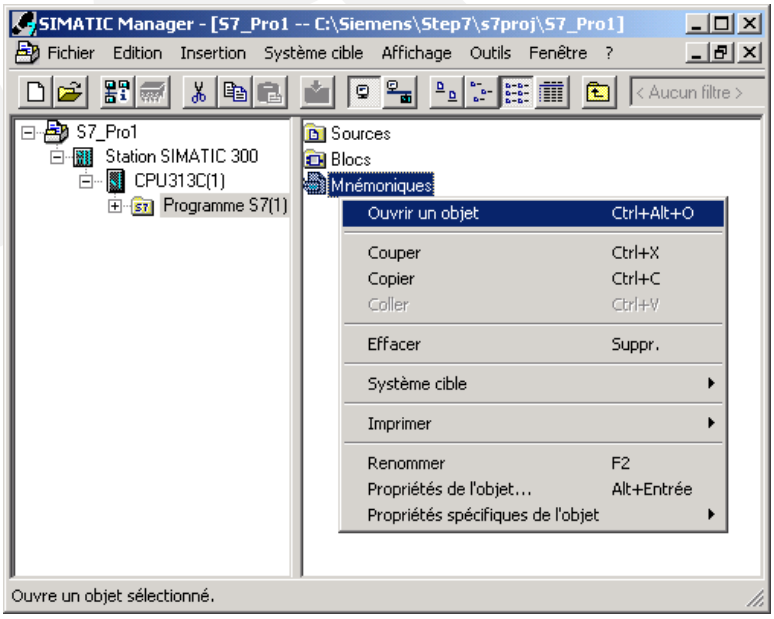


- In **Language** tab, select **German abbreviations** and click on **OK**.

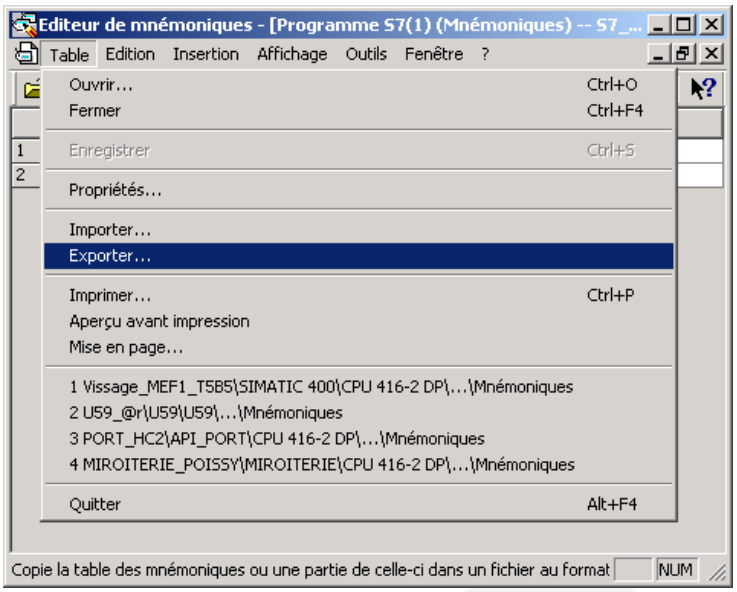


5.3.1 Create ASC file

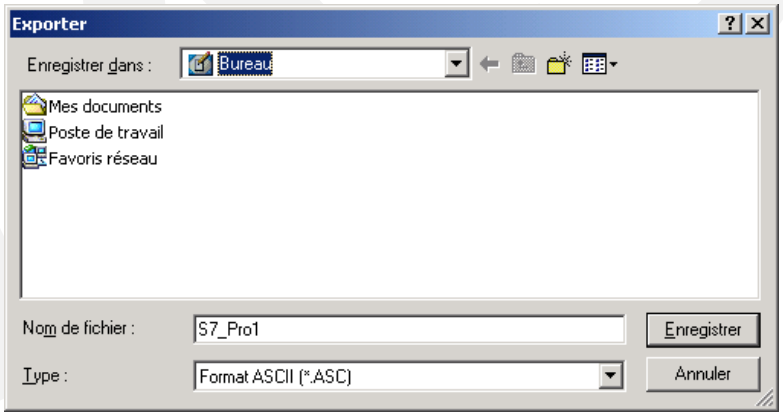
- To create an **.ASC** file, you have to open the symbols editor.



- In the symbols editor, use the **Table** > **Export...** menu



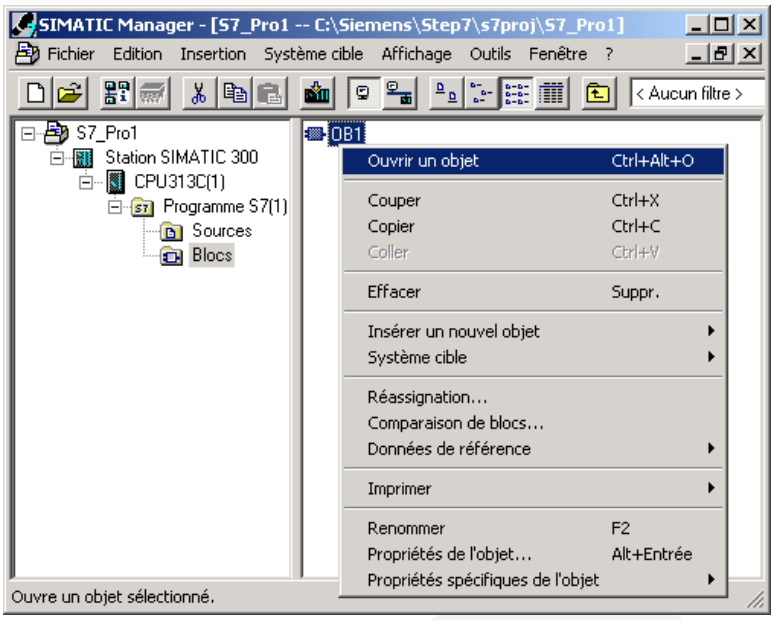
- Choose the **ASCII**[1] **Format** (*.ASC) file type, the **name** and the **directory** of the **.ASC** file to create, and click on **Save**.



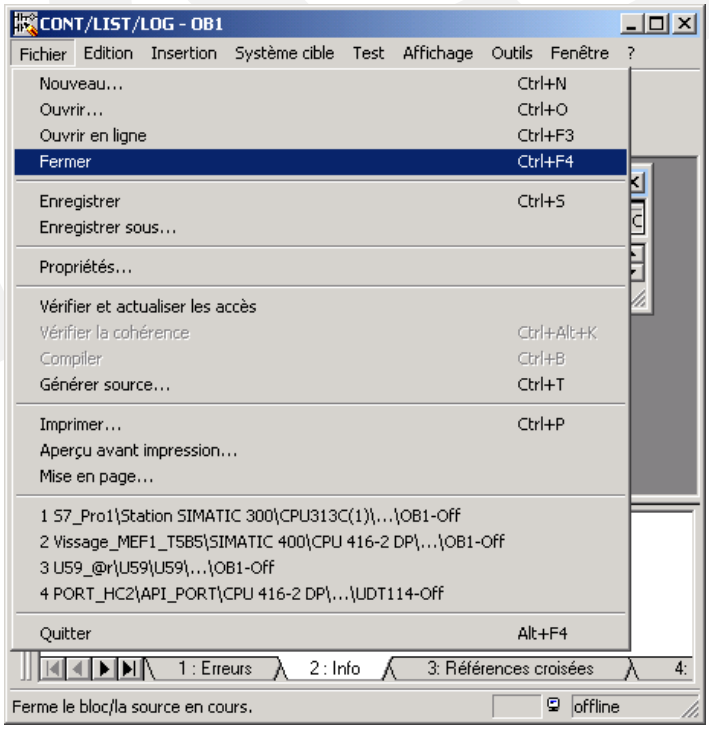
5.3.2 Create AWL file

- To create the **.AWL** file, source LIST must be generated. Open CONT/LIST/LOG code editor.

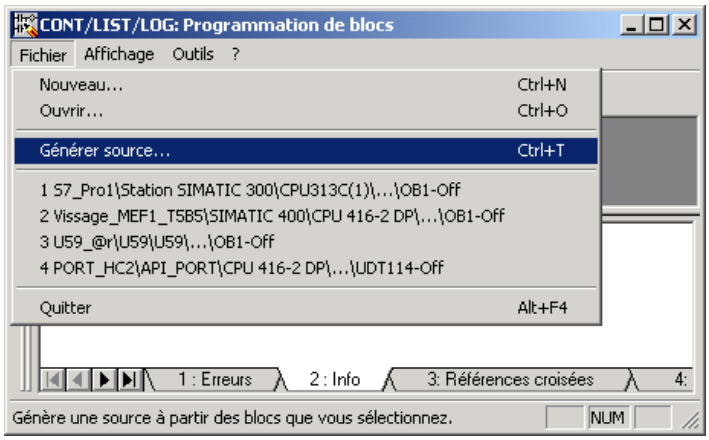
[1] American Standard Code for Information Interchange



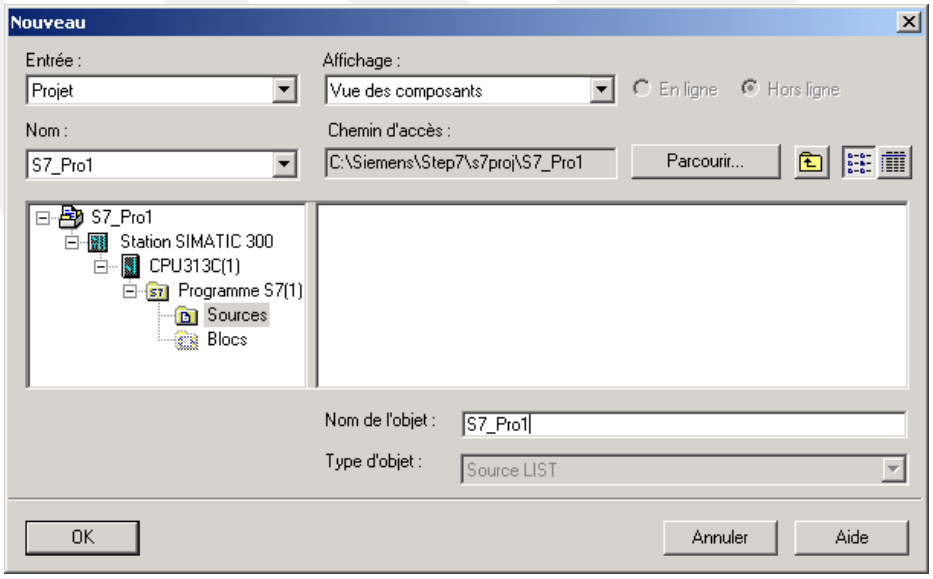
- Close all edited objects (Menu **File** > **Close**)



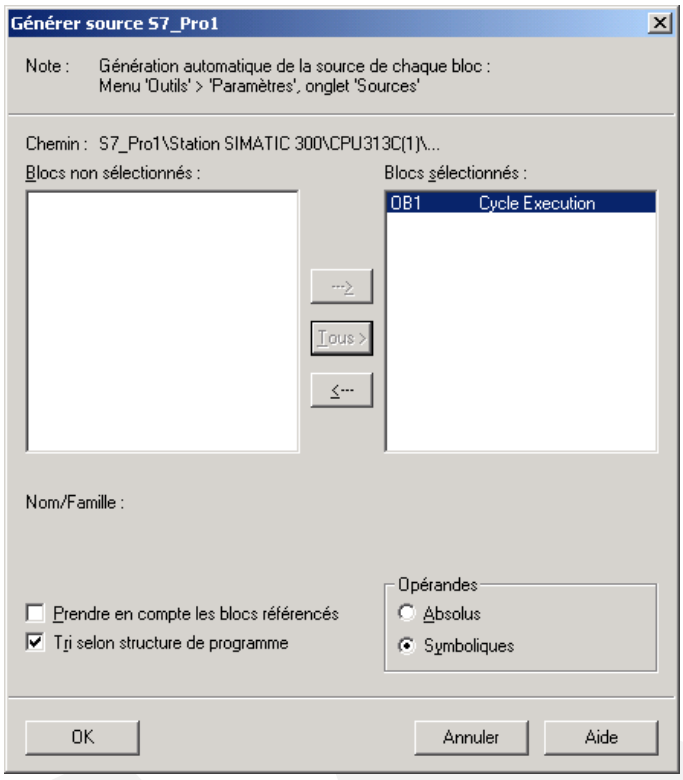
- Generate the program source (Menu **File** > **Generate source...**)



- Give a name of the object to be generated.



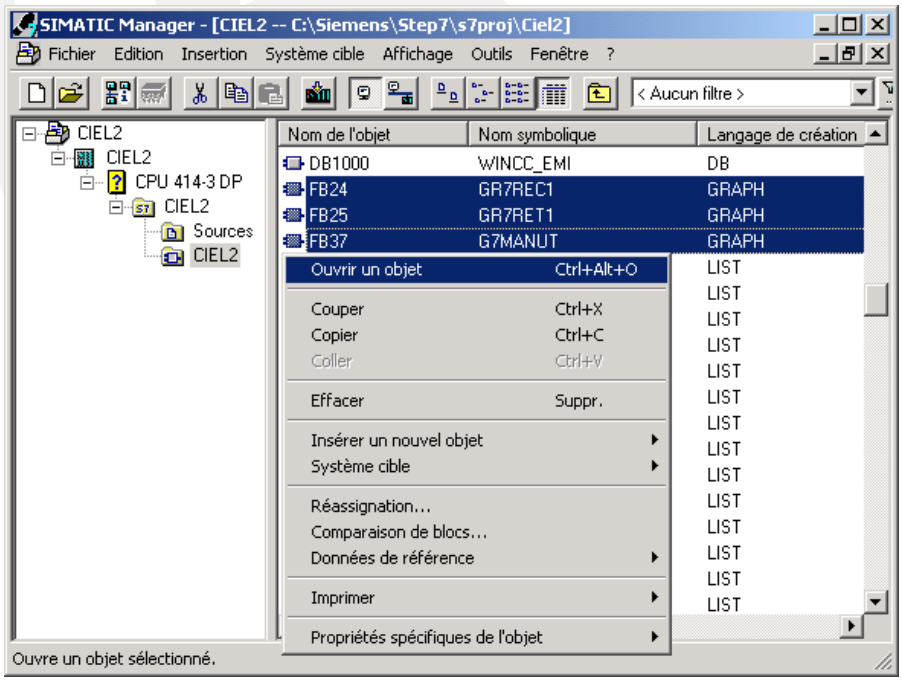
- Use button **All** > **select all blocs**, select only the **Sort by program structure** option. Select the **symbolic operands** and click on **OK**.



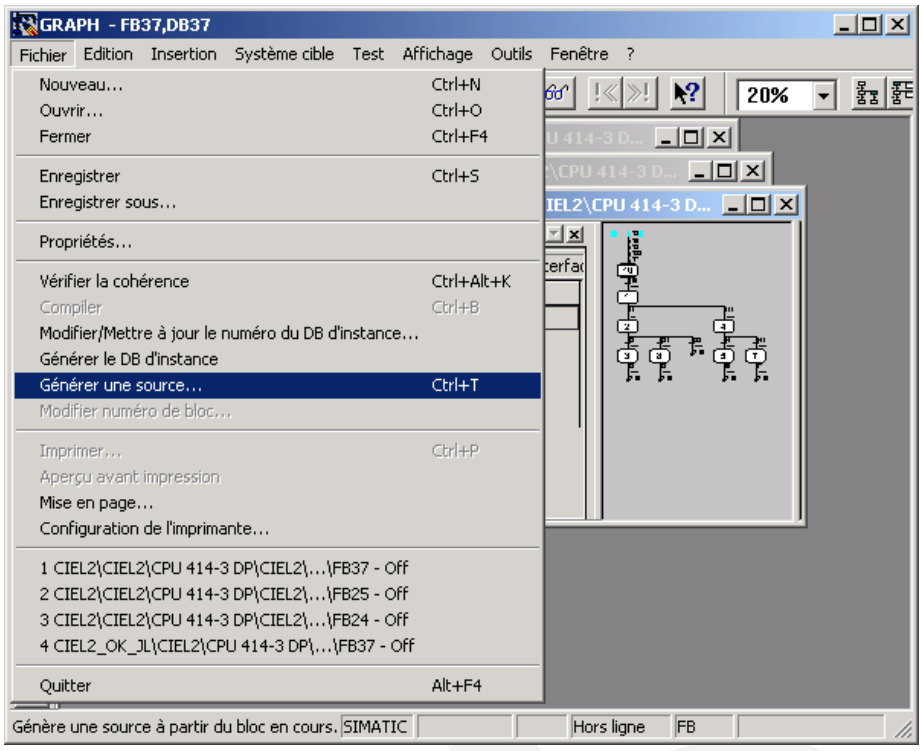
- If your application contains SAFETY blocs, you need to export those blocs in a separated **AWL** file. See below the chapter for safety application.

5.3.3 Create GR7 file

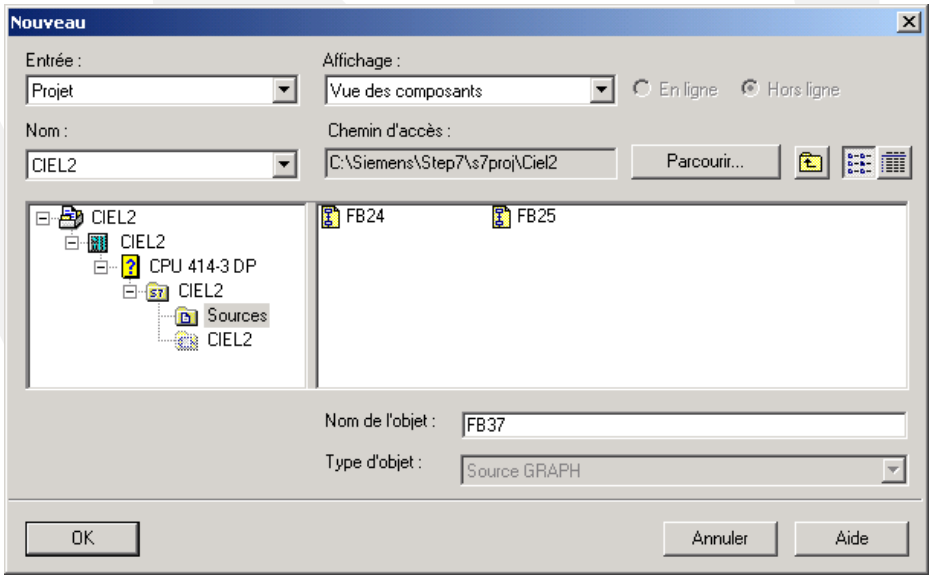
- To create the GR7 sources, select all GRAPH blocs and open the GRAPH editor



- For each GRAPH bloc, generate the source (menu **File** > **Generate a source**).



- Give a name to each GR7 source file and press **OK**.



5.3.4 Create CFG file

The CFG file is required for analysing the communication and the hardware configuration of the Station. To get the CFG, you need to open NetPro to edit the network configuration.

Then select the corresponding station by double clicking on the PLC. Then choose export in the file menu.

5.3.5 Create SCL source for CFC codes

- If the application contains some CFC codes, those codes must be compile to generate the corresponding SCL source files. Open CFC code editor.

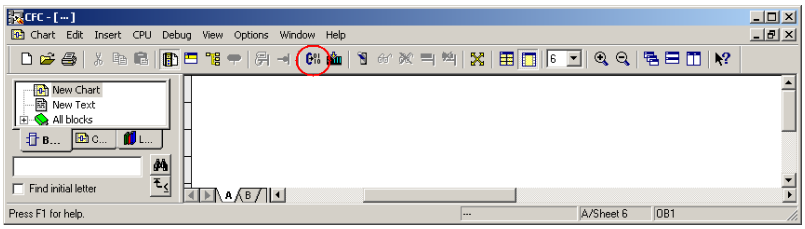


Figure 5.2: Compile CFC

- Compile the code with options **Entire Program** and **Generate SCL sources**

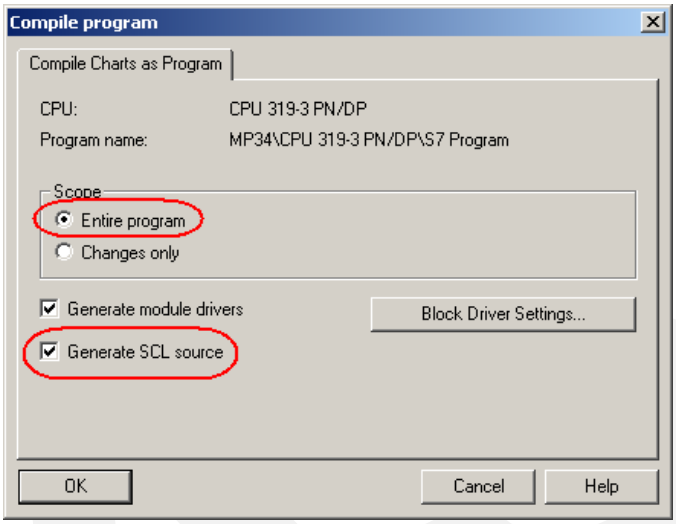


Figure 5.3: Compile Option with SCL source generation

- The generated SCL source file can then be exported like other source files, see export files.

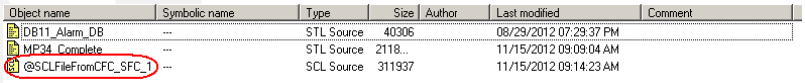


Figure 5.4: Right-Clic

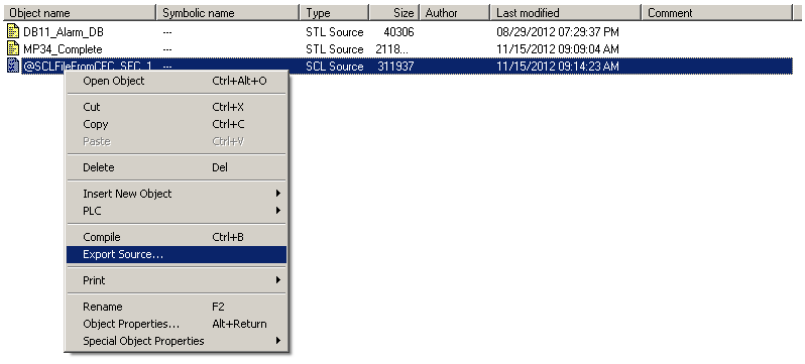


Figure 5.5: Export Files

5.3.6 Safety applications

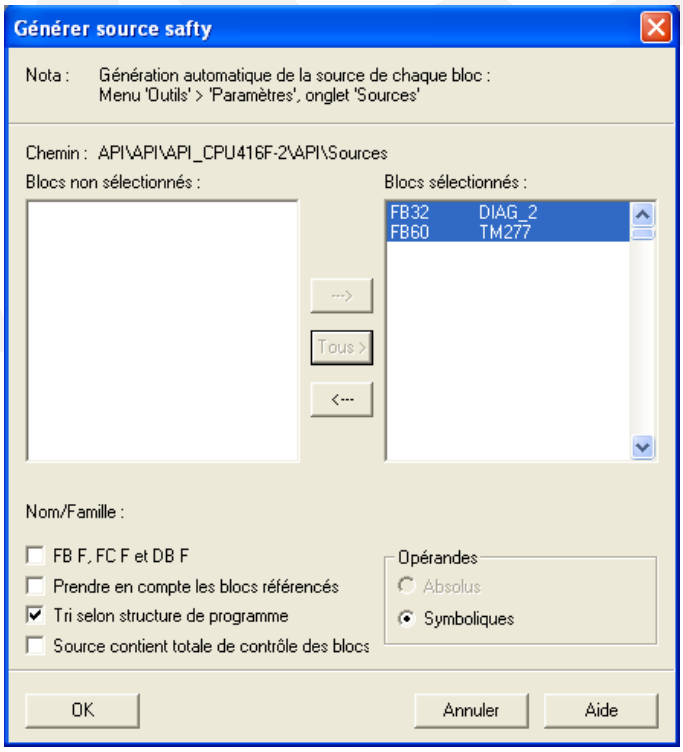
A Safety application contains two parts:

- the standard part
- the safety part

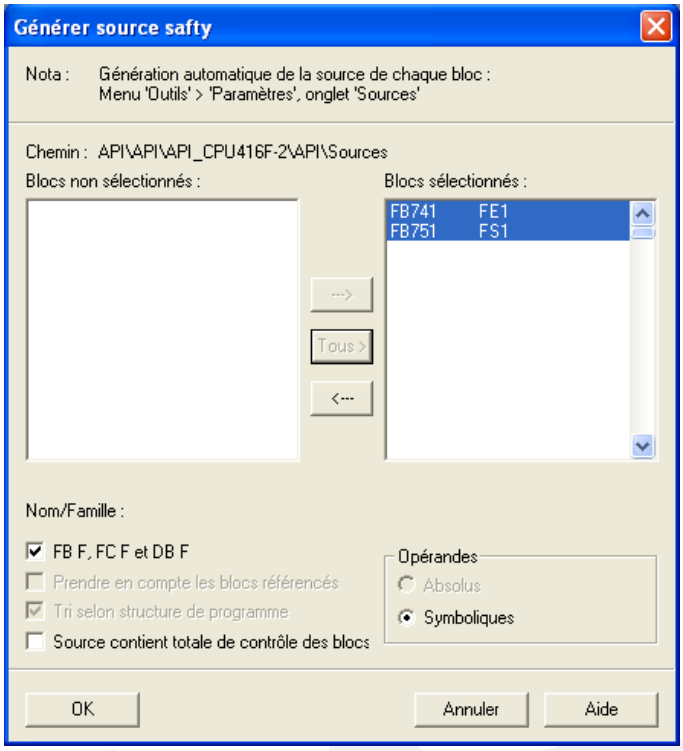
Exports for safety application

The export for safety application is made in four steps.

- First export the **.asc** file that contains the data base for the both parts. see above how to export asc file.
- Generate the sources for the standard part (non-safety). see above how to generate AWL and GR7 files.



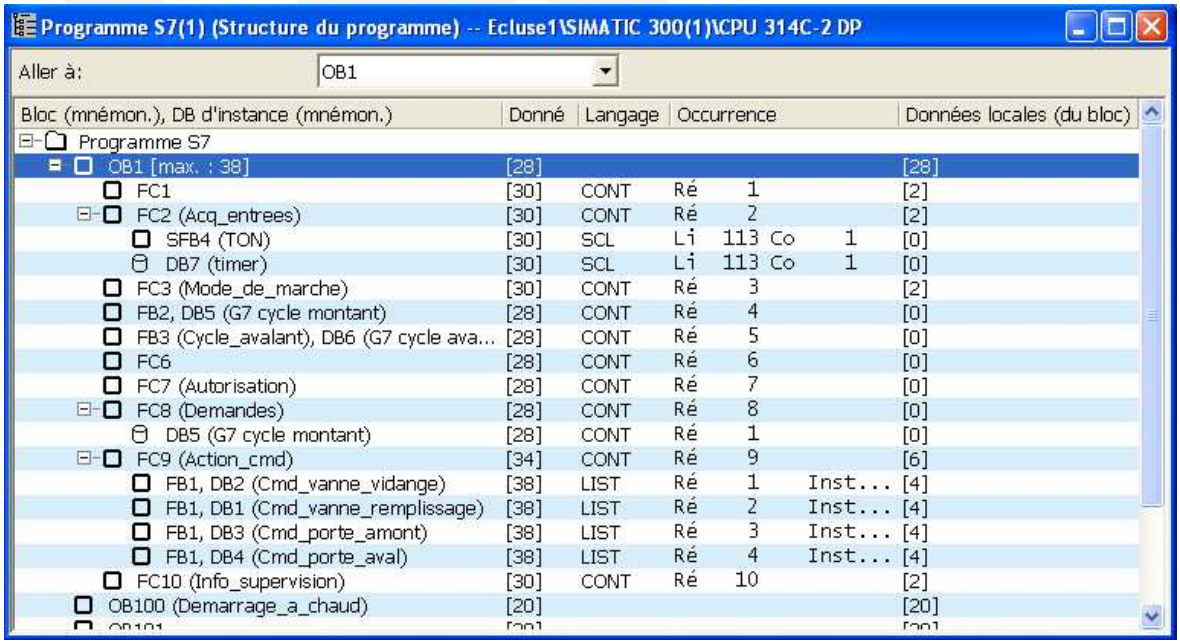
- Generate the source for the safety part in an other **AWL** file



- Export the program structure in a **.dif** file. It will be used to determine whether a module is safety or not.

Create dif file

- To create the **.dif** file, in the Simatic Manager of Step7 use the menu **Tools** > **Reference Data** > **Display**.
- In the parameter window, select the **program structure** and clic **OK**.
- The program structure is displayed.

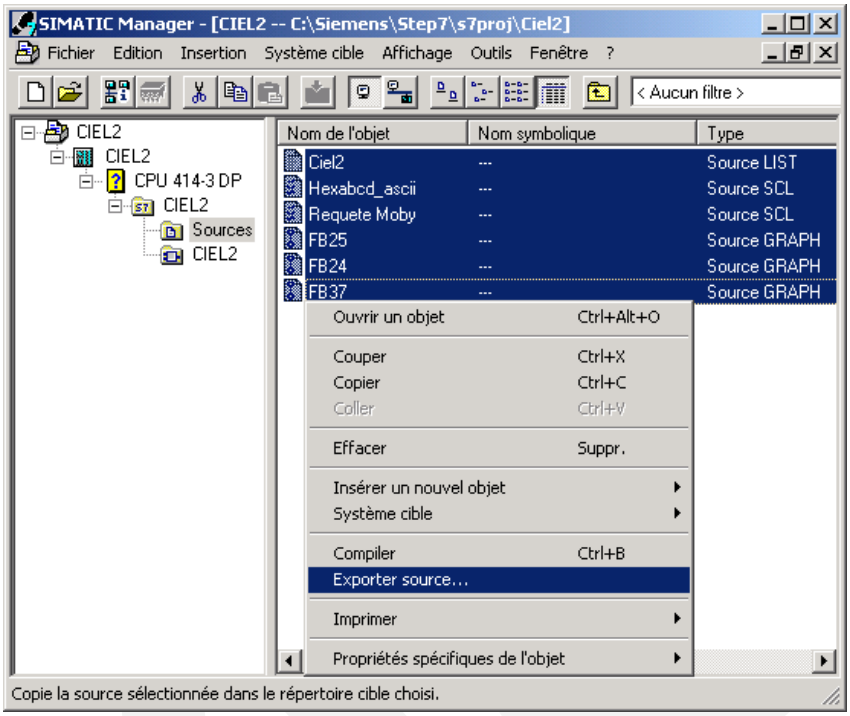


- use menu **Display** > **Exportation**.

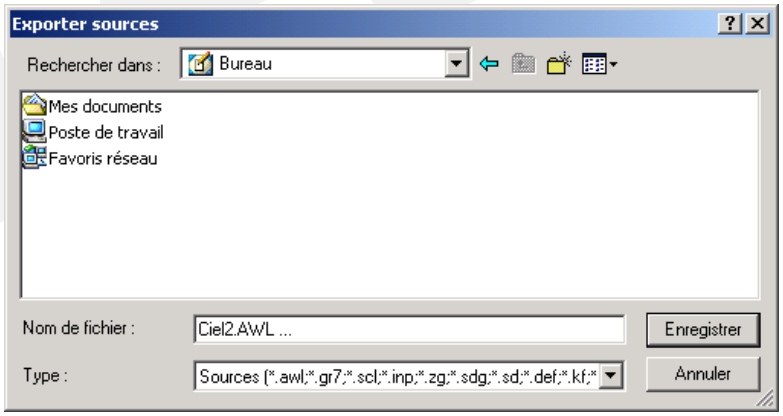
- Give a name to the file (eg: structure.dif) and save. The dif file is generated.

5.3.7 Export files

- Select LIST and GRAPH sources (resp. AWL and GR7 files) previously generated and also the SCL sources. Copy sources using contextual menu **Export source…**.



- **Select folder** where to export sources files.



Before to be analyzed by the PLC Checker, the **ASC**, **AWL**, **GR7** and **SCL** files must be imported in the Eclipse platform workspace.

5.4 XEF or ZEF export file generation from Unity

Unity application must be exported in a XEF or ZEF export file in order to be analyzed by PLC Checker.

You can create an XEF or ZEF export file with the **File** > **Export application...** menu in Unity software.

1. In Unity menu, select **File** > **Export application...**
2. In the "Export application" dialog box, give a **name** to the XEF or ZEF file
3. If needed, change directory where the XEF or ZEF file will be created,
4. and click on **Save**

Before to be analyzed by the PLC Checker, the XEF or ZEF file must be imported in your workspace (website or Eclipse platform)

5.5 SMC file retrieval from OMRON Sysmac Studio

No export operation is required from OMRON PLC Checker. PLC Checker understands directly the application format used by OMRON Sysmac Studio (from v1.0.3).

The correct location of application directory depends on your Microsoft Windows installation. It is most probably located in your document folder. Just search in your application files directory, the file with SMC extension.

5.6 XML export file generation from Codesys V3

Codesys V3 application must be exported in a XML export file in order to be analyzed by PLC Checker.

You can create an XML export file with the **Project** > **Export...** menu in Codesys V3 software.

1. In Codesys V3 menu, select **Project** > **Export...**

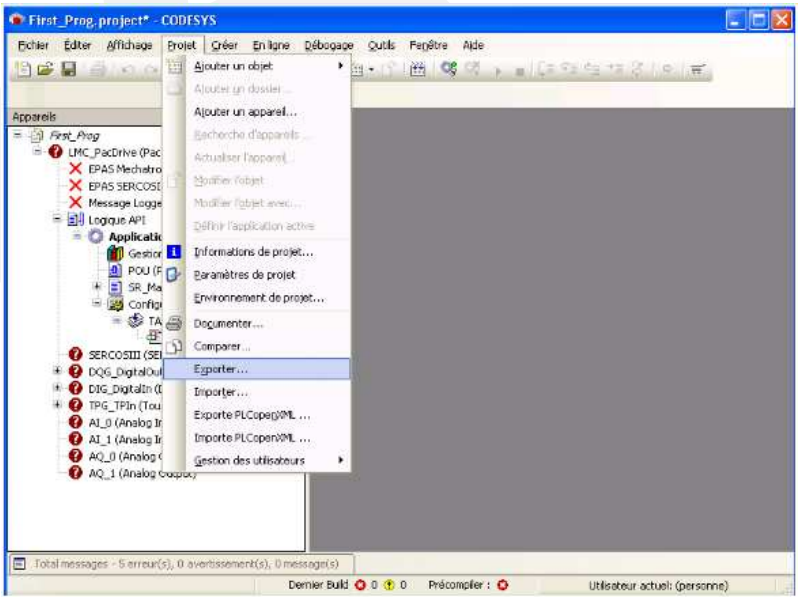


Figure 5.6: Export XML file

2. According to your needs, choose the export version

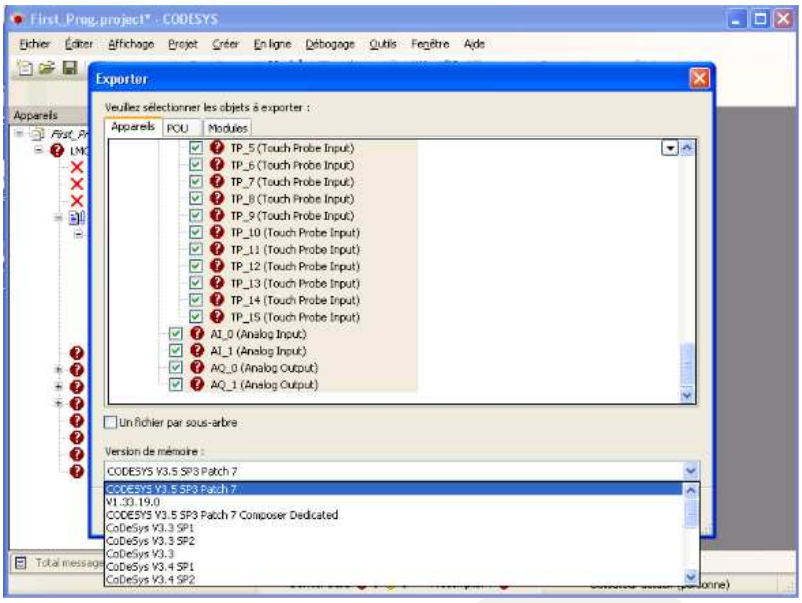


Figure 5.7: Choice of export version

3. Renames the exported file using the **XML** extension instead of *.export

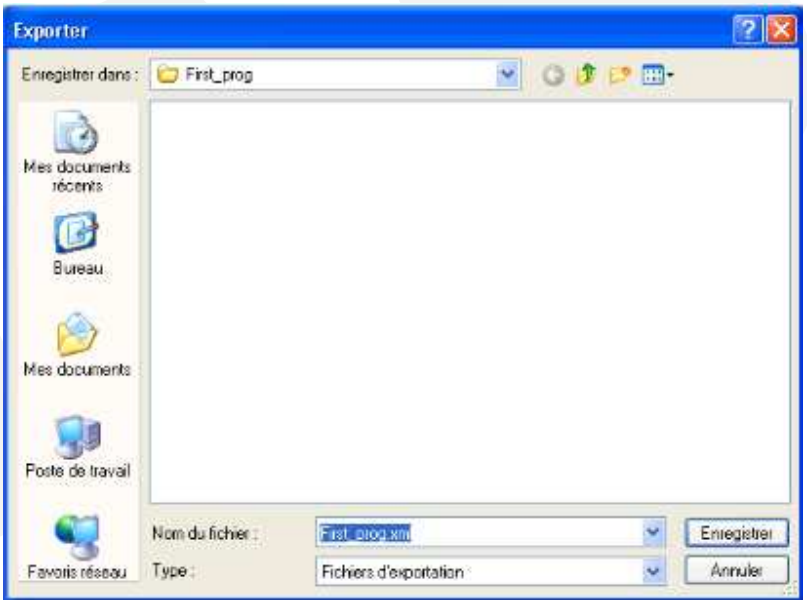


Figure 5.8: Change extension

4. and click on **Save**

Before to be analyzed by the PLC Checker, the XML file must be imported in the Eclipse platform workspace.

5.7 EXP export file generation from Codesys V2

Codesys V2 application must be exported in a EXP export file in order to be analyzed by PLC Checker.

You can create an EXP export file with the **Project** > **Export...** menu in Codesys V2 software.

1. In Codesys V2 menu, select **Project** > **Export...**

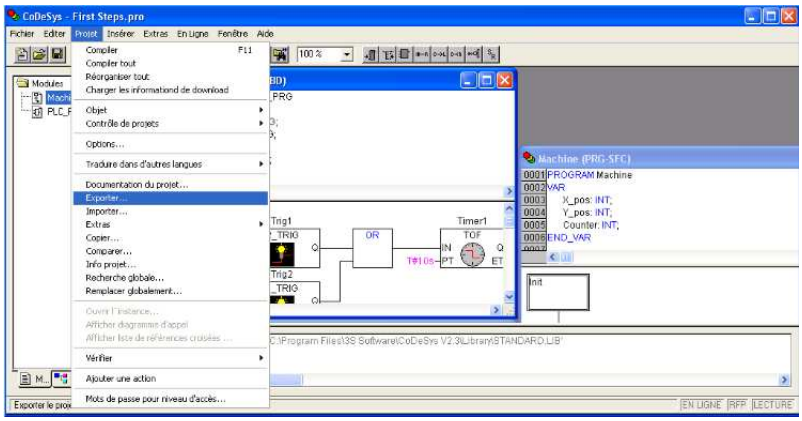


Figure 5.9: Export EXP file

1. In the "Export application" dialog box, give a **name** to the EXP file

2. and click on **Save**

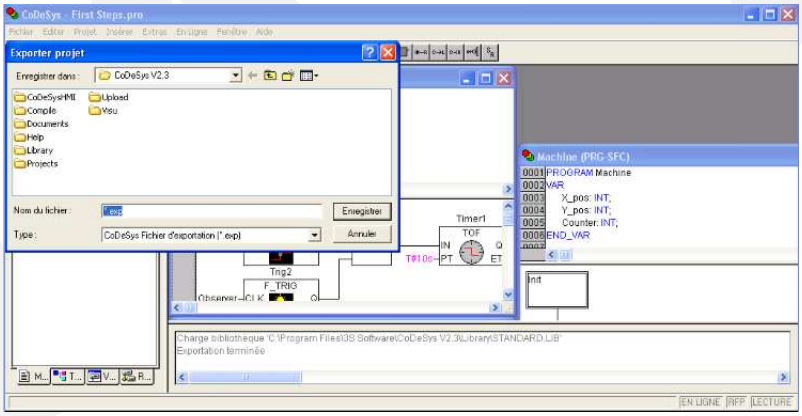


Figure 5.10: Save EXP file

Before to be analyzed by the PLC Checker, the application file must be imported in the Eclipse platform workspace.

5.8 XML export file generation from SoMachine

SoMachine application must be exported in an XML export file in order to be analyzed by PLC Checker.

You can create an XML export file with the **Project** > **Export...** menu in SoMachine software.

1. In SoMachine menu, select **Project** > **Export...**

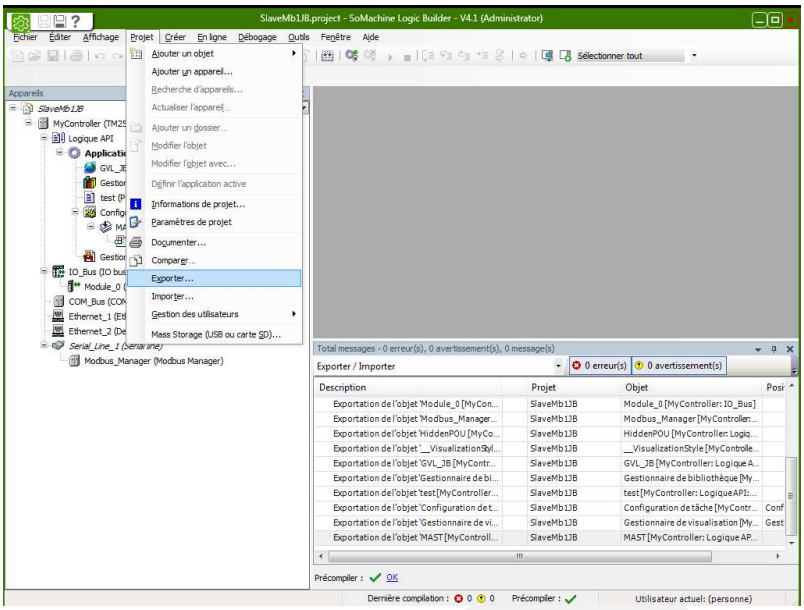


Figure 5.11: Export XML file

2. According to your needs, choose the storage version

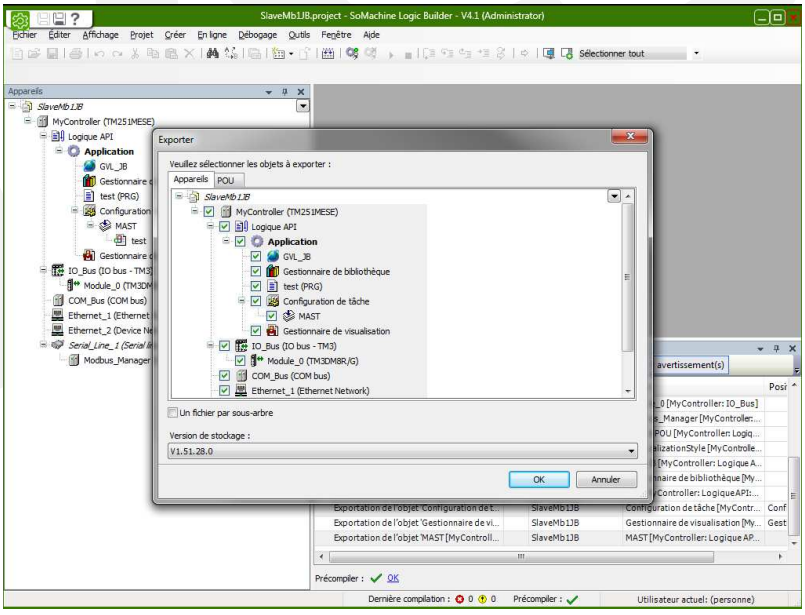


Figure 5.12: Choice of storage version

3. Renames the exported file using the **XML** extension instead of *.export

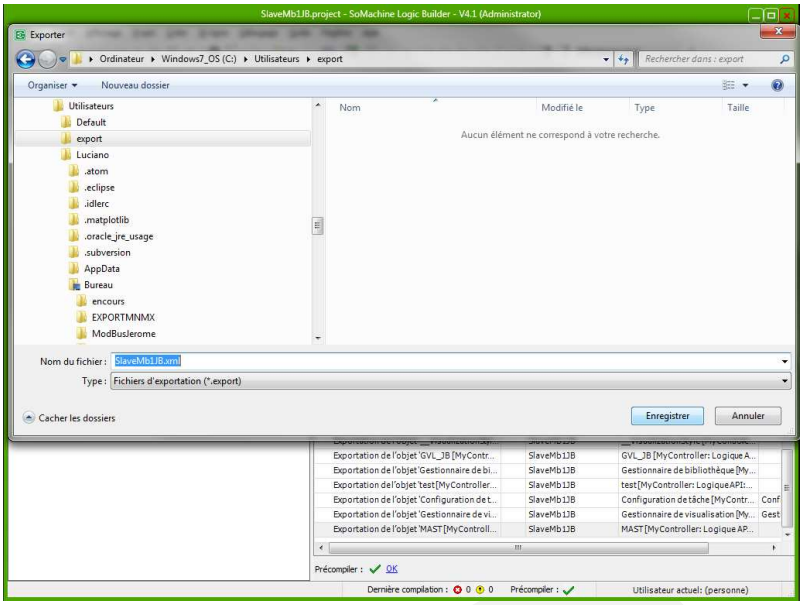


Figure 5.13: Change extension

4. and click on **Save**

Before to be analyzed by the PLC Checker, the XML file must be imported in the Eclipse platform workspace.

5.9 XML export file generation from Beckhoff Twincat

Beckhoff Twincat application must be exported in an XML export file in order to be analyzed by PLC Checker.
You can create an XML export file with the **Solution Explorer** > **Export PLCopenXML...**.

1. In the Solution Explorer window , select the project

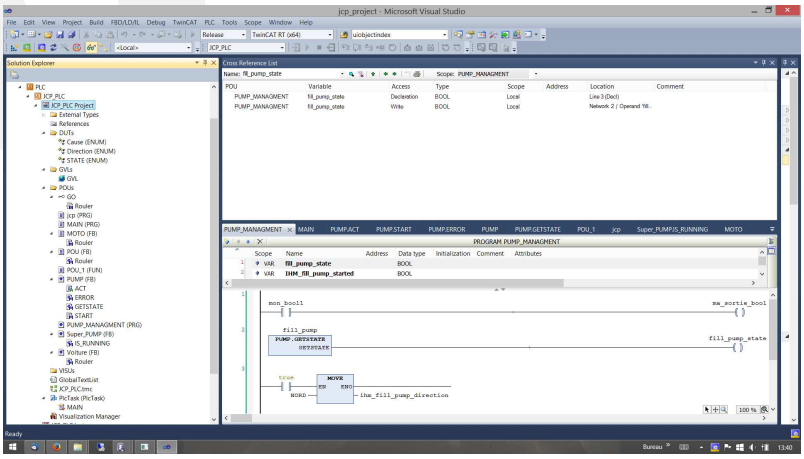


Figure 5.14: Select the project

2. Clic the right button and choose **Export PLCopenXML...**

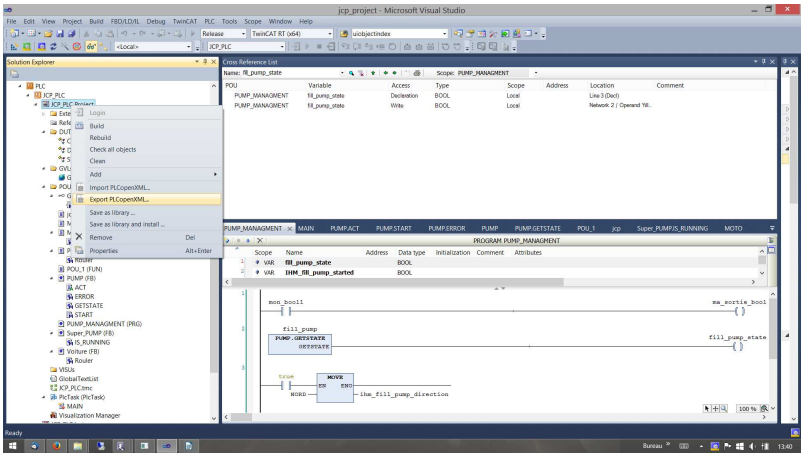


Figure 5.15: Export fu fichier XML

3. and click on **Save**

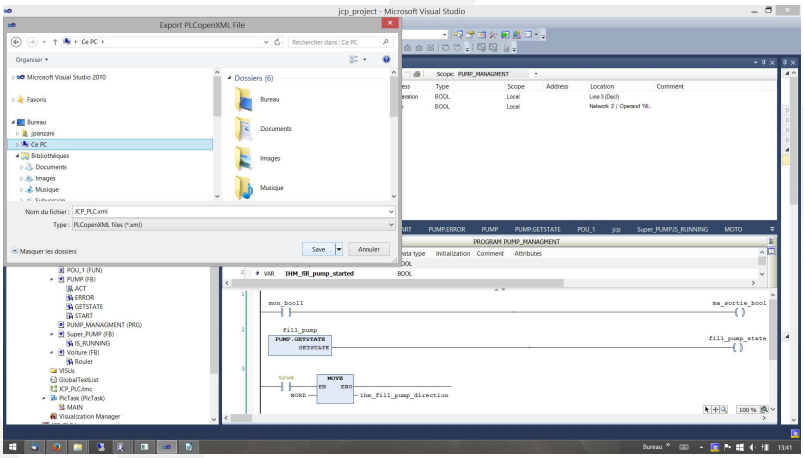


Figure 5.16: Sauvegarde du fichier XML

Before to be analyzed by the PLC Checker, the XML file must be imported in the appropriate interface (Itris Website or Eclipse platform workspace).

5.10 EXP export file generation from Beckhoff TwinCAT V2

TwinCAT V2 application must be exported in a EXP export file in order to be analyzed by PLC Checker.

You can create an EXP export file with the **File** > **Export application...** menu in TwinCAT V2 software.

1. In TwinCAT V2 menu, select **Project** > **Export...**

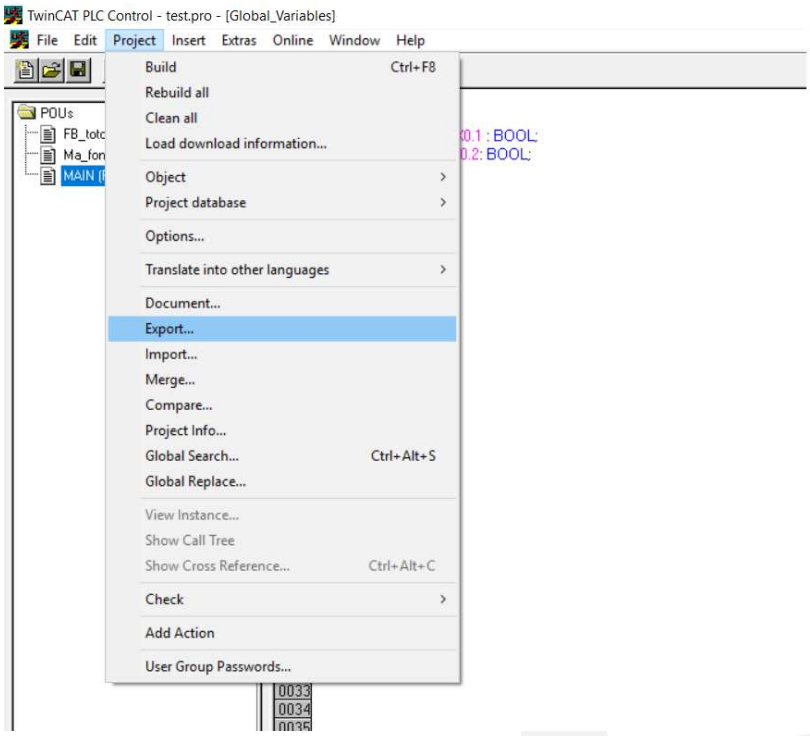


Figure 5.17:

1. In the "Export application" dialog box, give a **name** to the EXP file
2. and click on **Save**

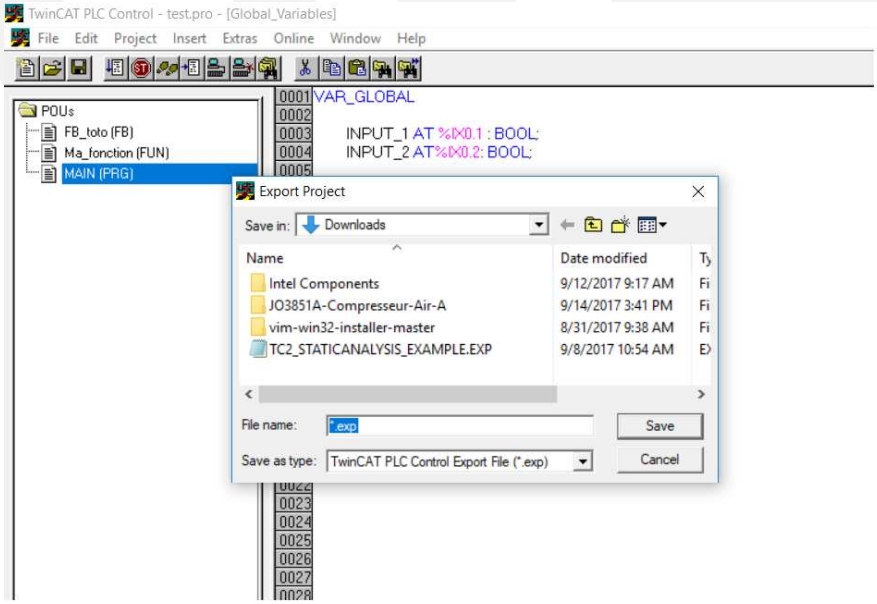


Figure 5.18:

5.11 PC5 export file generation from RSLogix 5

RSLogix application must be exported in a PC5 export file in order to be analyzed by PLC Checker.

Open the project with RSLogix500 programming workbench and :
File→Save As→Type choose export **Library Files (*.PC5)**
Files generated are :

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

Only PC5 and SY6 files are required for Checker.
Before to be analyzed by the PLC Checker, the application file must be imported in the Eclipse platform workspace.

IAS

Chapter 6

Itris Import/Export Tool

6.1 Introduction

Itris Automation's **Itris Import/Export Tool** is a simple way to proceed with code importation and exportation from and to several Software Development Environments, in a unified user experience. It is suitable for use with PLC Checker, PLC DocGen, ICS Monitoring, PLC Converter, ...

This tool connects with your usual PLC program development software, so you need to set it up on the same computer as the one you are using to develop your applications.

The most advanced users are able to use **Itris Import/Export Tool** in command line mode, allowing them to use it with their favourite continued integration software or to create automatic scripts to be more efficient.

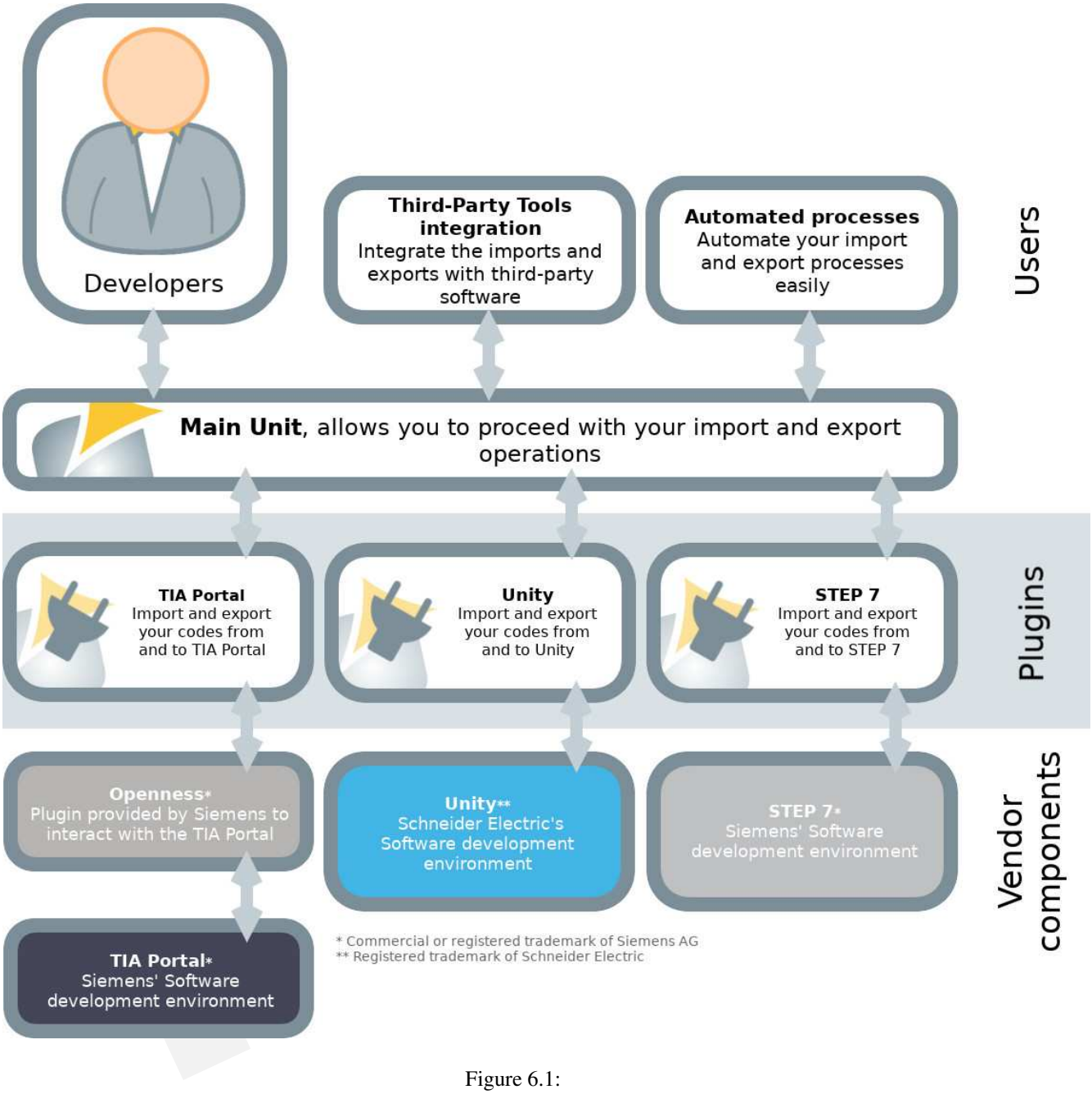


Figure 6.1:

6.2 Installation

Itris Import/Export Tool is based on a modular architecture. You will have to setup the **Main Unit**, which will provide you with the graphical and command-line interfaces. Then, you will have to add some plugins. Each plugin provides you with a functionality, for instance, **Code Exportation for STEP 7** or **Code Importation for TIA Portal**. Generally, if you are using PLC Checker or PLC DocGen, you will need a code exporter, and if you are using PLC Converter, you may need a code exporter and you will need a code importer.

In order to easily find the plugins that you need, Itris Automation set up a web selector, available at the given address: http://www.itris-automation.com/itris-importexport-tool/ .

To use it, simply choose the Software Development program that you are using and the Itris tools that you want to use with it. The online selector will then tell you what plugins to download and setup.

The setup process in itself does not require specific operations. By default, the main unit is put in the "Program

Files" folder and its plugins are added here in sub-folders. You may change the location of the main unit, however the plugins must remain as sub-folders to the main unit for them to work.

NOTE	**This software replaces TIA Portal Import/Export Tool. It remains compatible with the former setup and the former plugins (called "modules"). You can use the Itris Import/Export Tool installer to update the TIA Portal Import/Export tool, however you will need to be vigilant when setting up new plugins as the default setup folder is not the same for the two tools.**

6.3 Usage

After the setup of the main unit and your required plugins, you will immediately be able to proceed with importations and exportations of your PLC code.

NOTE	**Some PLC programming software may require additional configuration. Please check the manuals dedicated to the particular software for more information.**

Two interfaces are available when you are using the **Itris Import/Export Tool** :

- The graphical UI : allows you to easily perform your exportation and importation operations. This interface is the one that you have to use if you have not automatised the processes.
- The command-line interface : allows you to automatise the importation and exportation processes by providing a user interaction-less process to perform the importations and exportations of code. You will need this interface if you want to integrate the importation and exportation processes with a third party tool or in your continued integration system.

6.3.1 Graphical UI for the daily usage

This interface allows you to perform the importation and exportation operations easily. It is composed of three parts:



Figure 6.2:

The first part is used to perform a **code exportation**. You have to click on the button "Select a file" in the area called "Export a project" and localise your project's code. Then you will have to run the process by clicking on "Proceed".

NOTE	**The PLC programming interfaces do not all react in the same way, so some of them may need some additional information from you during the process. Please check the manuals dedicated to each software for more information.**

Once the exportation process is finished, **Itris Import/Export Tool** will tell you where the exported file (in ZIP format) is located and will ask if you want to open the folder, for your convenience.

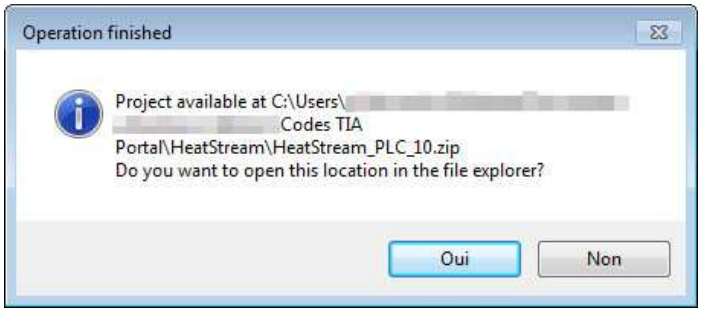


Figure 6.3:

The second part of the screen is used to perform a **code importation**. To do so, you just have to click on the button "Select a file" in the area "Import a project". **Itris Import/Export Tool** will ask you to choose the ZIP file to import, and a project to import it to. The projects that you can import are dependant on the plugins that you have setup.

NOTE	**If the code that you want to import contains a block named the same as an existing block inside the project you have selected, then the newly imported block will replace the existing block.**

To run the operation, just click on "Proceed".

Finally, the third part of the screen is used to validate the operation and to get some information about what is happening.

6.3.2 Command line interface to automate the processes

This interface is designed to be used by advanced users for advanced usages without requiring any user interaction (no need to select the project and then click on "Proceed"). It allows the integration of the importation and exportation processes into automated tasks for instance.

NOTE	**This interface is not aimed at configuring the system. Some software development environments require additional configuration that you won't be able to manage using this interface. It is advised that you use the graphical interface at least once to check that all is running as expected and to make any configurations, if required, before using the command line interface.**

To use this interface, you will need to use a command prompt or any tool able to pass arguments to a program (a Windows' shortcut for example). The executable is located inside the same folder as the main unit, and is called **ImportExportMainUnitCli.exe**.

By default, you have to pass one or two arguments to the executable, which are both files paths.

- If there is only one argument, then it means that you want to run an export process, and the path you will give is the path to a supported project.
- If there are two arguments, then it means that you want to run an import process, the first path that you will give is the path to the ZIP file to import, the second one is the path to the project in which to import the ZIP file.

At any moment, you can get help using the argument **-h** (or without any argument).

Some usage samples :

- ImportExportMainUnitCli.exe -s project.ap13: Export the project located inside **.\project.ap13** (TIA Portal Version 13)

- ImportExportMainUnitCli.exe -s -p=2 project.ap13 : Exports the second program of the project located in **.\project.ap13** (TIA Portal Version 13)
- ImportExportMainUnitCli.exe -s codeFromConverter.zip project.ap13 : Silently import the code inside **.\codeFromConverter.zip** into the project located in **.\project.ap13** (TIA Portal Version 13)

Arguments list:

- ? : Displays the help
- p : If your project contains several programs, then define the program to export (1 stands for the first program...), the default value is **?** meaning that in such a case, the utility will ask the user to choose the project during the process (it could be useful to run it once in this way to get the right project number)
- d : **p**'s synonym
- t : For code exportation, allows the user to choose where they want the exported ZIP file to be saved. It will not override an existing ZIP file
- T : Does the same as **t** but overrides any existing file
- s : Displays fewer messages during the process
- v : Displays more messages during the process
- i : Displays some information about the installation (plugins, versions...)

6.4 Visualise The Export Content

You are able to see an export's content by using the tool **Archive Viewer**, which is included in the **main unit**.

To do this, launch the Archive Viewer and find the export file (ZIP format) you want to use by going to the menu "Archive..." and then "Read". You will then be able to see the export's content as well as find information on the state of export and the content of the different code blocks.

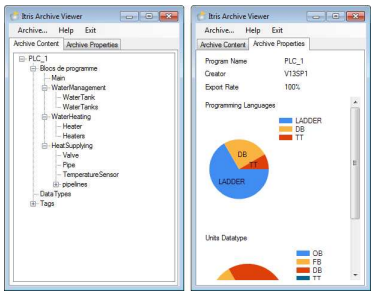


Figure 6.4:

6.5 Get the list of installed plugins

Both interfaces allow you to gather information about the plugins installed on your system.

When you are using the Graphical User Interface, right-click anywhere in the window and select "Plugins".



Figure 6.5:

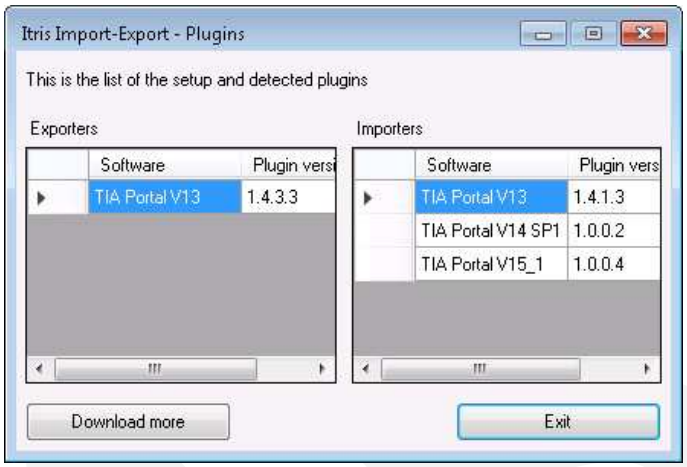


Figure 6.6:

NOTE	This functionality is available from the version 1.5.3.4 of the main unit

If you are using the Command Line interface, use the "-i" argument to get these information.

Chapter 7

Itris Import/Export Tool - STEP 7

7.1 Introduction of Itris Import/Export Tool

Itris Automation's **Itris Import/Export Tool** is a simple way to proceed with code importation and exportation from and to several Software Development Environments, in a unified user experience. It is suitable for use with PLC Checker, PLC DocGen, ICS Monitoring, PLC Converter, ...

For more information about how to setup the process and how to use it, please check the tool's manual. This documentation is only applicable to the STEP 7 plugin.

7.2 STEP 7 Plugin Pre-requisite

Itris Import/Export Tool is able to export the PLC code from STEP 7, as long as the **main unit**, the **STEP 7 plugin**, and the STEP 7 development environment are installed on the same computer.

7.3 Specific warnings about STEP 7

7.3.1 Programming languages

The codes that you create with STEP 7 are not all able to be exported in the same way due to limits in the interconnection interface.

Your codes in Ladder (Contact), LIST and LOG, and the mnemonics tables are able to be exported without any intervention of any kind from the user. If you are using only these languages/elements, then you are not concerned by this paragraph.

Codes written in **SCL, CFC and SFC** are not able to be exported automatically. If you are using one of these languages, please read this paragraph carefully. The plugin for STEP 7 is able to include these codes in the exports if you have generated the sources for these languages before running the exportation process.

To do so, for the codes in **SFC**, you will have to proceed with a generation operation every time you run an exportation process. This will create a copy of your code in a location that **Itris Import/Export Tool** can access:

- Select each block that you want to export and then open it using the editor
- Click on "File" and then "Generate a source..."
- Give a name to the source, and validate. You will have to repeat this operation for each block that you want to generate

For the codes written in **CFC**, you will have to export them in the SCL format, using the following procedure:

- Compile the sources and select the following options:
  - Entire program
  - Generate SCL source

For the SCL blocks, if they are linked to a valid SCL source, then you don't have anything to do. Otherwise, if the source has been deleted, you will have to open the block using the STL editor, and generate a new source.

These procedures are explained in more detail in the PLC Checker manual (have a look in the section "ASC, AWL, GR7 and SCL Export files generation from Step7").

7.3.2 Command line interfaces

Due to some limitations in the interconnection interface, STEP 7 may ask the user to interact during a code exportation process. This is a problem when you want to completely automate the process. You should consider checking that the process works without interaction before running it automatically.

7.3.3 Compatibility with the old versions of PLC Checker, PLC DocGen and PLC Converter

Considering that the import/export system is more recent than some versions of the Itris Automation products, the export format is not compatible with all of them.

In order to get compatible files, you will have to use the software **STEP 7 Retro-Compatibility Tool**, included with the export plugin. This software will repack the export in a format that you will be able to use with Itris Automation products.

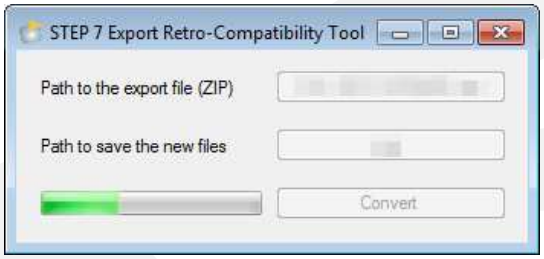


Figure 7.1:

To proceed, you just have to give the location of the ZIP export file and a file in which to save the repacked application. Then just click on "Convert".

At the end of the operation, you will get several files that you can use directly with Itris Automation products:

- A file whose extension is AWL, containing code written in CONT (Ladder), LIST and LOG (Logigramme),
- A file whose extension is SCL, containing code written in SCL and CFC (if any),
- A file whose extension is GR7, containing code written in SFC (if any),
- A file whose extension is DIF, containing a description of your application's hierarchy
- A file whose extension is ASC, containing the mnemonics table

Chapter 8

Itris Import/Export Tool - TIA Portal

8.1 Introduction of Itris Import/Export Tool

Itris Automation's **Itris Import/Export Tool** is a simple way to proceed with code importation and exportation from and to several Software Development Environments, in a unified user experience. It is suitable for use with PLC Checker, PLC DocGen, ICS Monitoring, PLC Converter, ...

For more information about how to setup the process and how to use it, please check the tool's manual. This documentation is only applicable to the TIA Portal plugins.

8.2 Available plugins for TIA Portal

Every version of the TIA Portal requires a different plugin in order to work properly. In some cases, the compatibility between plugins is not possible for the same version number of TIA Portal (for example V14 and V14 SP1).

Currently, these plugins are available for the TIA Portal:

- Version 13 SP1
- Version 13 SP2 : Works with the same plugin as Version 13 SP1
- Version 14 : This plugin does not work with the Version 14 SP1 and it also cannot co-exist with the SP1 plugin.
- Version 14 SP1 : This plugin does not work for the Version 14 and it also cannot co-exist with the Version 14 plugin.
- Version 15
- Version 15.1

8.3 Specific Pre-requisite for TIA Portal

To use the plugins to proceed with import/export operations, you will have to meet the following conditions (the procedures to meet them are described below):

- Have the TIA Portal installed on the computers that will run the operations,
- Have all the optional modules (such as Safety, WinCC...) used in the projects to export setup on the computer that will perform the exports,
- Have Openness (interconnection interface)
- Have the authorization files for the version 13 SP1, 13 SP2 and 14 of the TIA Portal
- Handle the projects in the same version as the TIA Portal

8.4 Plugins Setup

The TIA Portal plugins have to be set up in the same way as the plugins for the other Software Development Environments. They all require Openness to be setup too. Openness is a piece of software provided by Siemens as the interconnection system between **Itris Import/Export Tool** and the TIA Portal. To learn how to setup Openness, please read the section "Openness installation".

Once Openness is ready, you will have to authorize the users to use it, only once per computer. To learn how to do so, have a look at the section "Users authorization".

The versions **13 SP1, 13 SP2 and 14** of Openness require you to provide some authorization files to work. To learn how to get these files and set them up, read the section "Authorisation files".

8.4.1 Openness installation

For the versions **13 SP1, 13 SP2, 14 et 14 SP1** of the TIA Portal, Openness has to be setup independently from the TIA Portal. You will find it inside the "support" folder of the installation DVD of the TIA Portal. If you don't have the TIA Portal installation DVD, then you can download a trial version of the TIA Portal from Siemens' website and unzip it into a known folder. You will find the "support" folder after the unzip process. The TIA Portal license is used for the Openness package, so even if you have downloaded a trial version of the TIA Portal, the Openness package will not be a trial version as long as you have a valid license for your TIA Portal.

*If you are running the versions **13 SP1, 13 SP2 or 14** of the TIA Portal, you will need to get the authorisation files, please read the section "Authorisation files".*

For the versions **15** and **15.1** of the TIA Portal, the installation of Openness has to be done during the installation process of the TIA Portal. During the step where you choose the components to setup, you have to check the Openness Package in the list. If you didn't do this when you set up the TIA Portal, then restart the installer and choose to update the current installation.

Regardless of the version you are using, to complete the setup, you will have to authorise the users to use Openness, so read the section "Users authorisation" for the procedure.

8.4.2 Users authorisation

Once Openness is setup, you have to authorise the computer's users to use it.

To do so, you have to go on your computer's management panel:

- Right click on "Computer" (or "This PC" depending on which version of Windows you are using)
- Select "Manage"
- In the window that opens, go to "System tools", "Local users and groups" and then "Groups"

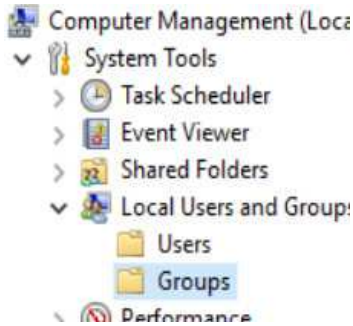


Figure 8.1:

- You should see a group called "Siemens TIA Openness" in the central part of the screen, double-click on it
- If the name of the user that you want to authorize appears in the screen, then the procedure is already complete, so you do not need to do anything else. Otherwise, click on "Add..."

- Type the name of the user that you want to authorize and click on "Verify names", the name of your user will now appear underlined, meaning that Windows recognises it. If not, try to complete it or search for it.
- Once you're done, click on "OK" followed by "Apply"
- Reboot your computer

Once this procedure is done, if you are using the versions **13 SP1, 13 SP2 or 14** of the TIA Portal, please complete the procedure "Authorisation files" if you have not already done so. For the other versions, you're ready to use your exporters and importers.

NOTE	**On Windows 10 Home edition, it is not possible to add users to the "Siemens TIA Openness" group**

8.4.3 Authorization files

If you are using the versions **13 SP1, 13 SP2 or 14** of the TIA Portal, Siemens requires you to declare the use of Openness. To do so, you just have to follow the following procedure :
TIA Portal Openness Guidance Usage File Enabler File.

You don't have to setup the files by yourself as **Itris Import/Export Tool** can do it for you. When you use a plugin that requires these files, a window will prompt you to provide them. They will be setup for you automatically at this time.

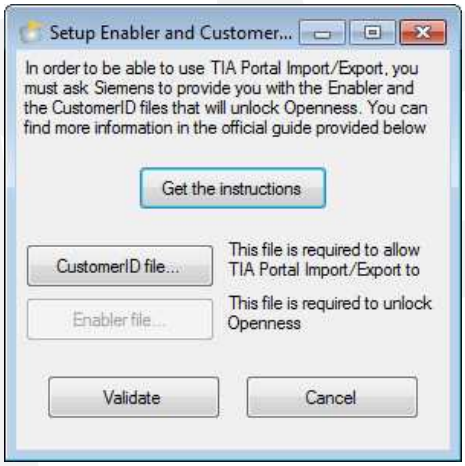


Figure 8.2:

8.5 TIA Portal specific information

8.5.1 Non compiled codes exportation

Non compiled codes cannot be exported by Openness. We suggest you to compile the entire project before running an exportation phase to prevent compilation-related exportation errors.

NOTE	**The import and export plugins do not compile your program to prevent any modifications to it. It's your responsibility to compile your programs.**

8.5.2 Protected code exportation

The plugins won't export any protected code in order not to break the code's protection rules applied by the customer's company.

If the user wishes to export all their code, it's up to them to unlock any protected code before running the exportation process. This allows the user to choose which blocks are allowed to be handled by Itris' tools.

For the Failsafe blocks, the protection can be removed as follows :

- Click on **Safety Administration**



Figure 8.3:

- In the entry **Access protection** click on **Change**
- In the window, enter your current password
- Click on "Revoke" to tell the TIA Portal that you no longer want to protect the blocks

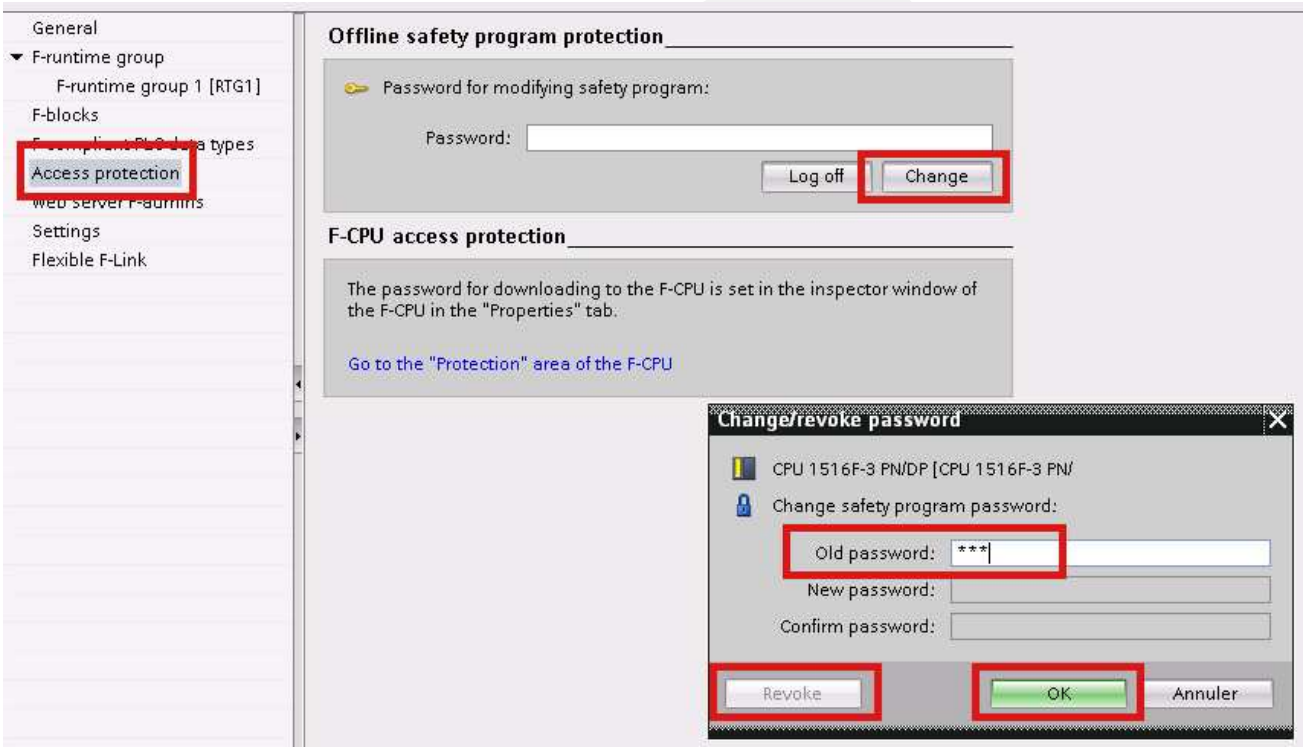


Figure 8.4:

- Save the settings and compile your code

NOTE	**Logging in to the program for edition does not unlock the codes for external access**

8.5.3 Project versions

The project files are saved in a version that is independent from that of TIA Portal. For instance, you can save a project as Version 14 and edit it with the TIA Portal Version 14 SP1 without realising.

Openness does not support this behaviour and requires you to only open projects that are in the same version. So if you are using the TIA Portal Version 14 SP1 and the Openness Version 14 SP1, then your projects must also be in the Version 14 SP1 format in order for Openness to open them.

To ensure that your projects are saved in the right format, open them using the TIA Portal, then go on the "File" menu. If the option "Upgrade" is not in a grey tint, then it means that your project is not in the same version as the TIA Portal. To update its version, just click on this menu entry.

8.5.4 Plugin authorisation

When you are running an operation for the first time with the TIA Portal, it will ask you if you want to authorise the plugin to access Openness via the following window:

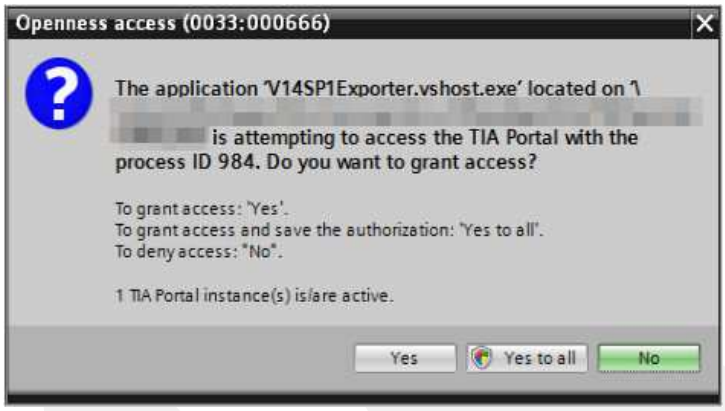


Figure 8.5:

To continue with the operation, two options are possible. The second one, "Yes to all" will ensure that this window will no longer appear. This is particularly adapted to when you want to use the tool with the command line interface or if you don't want to click on "Yes" every time you run an operation.

8.6 Introduction of Itris Import/Export Tool

Itris Automation's **Itris Import/Export Tool** is a simple way to proceed with code importation and exportation from and to several Software Development Environments, in a unified user experience. It is suitable for use with PLC Checker, PLC DocGen, ICS Monitoring, PLC Converter, ...

For more information about how to setup the process and how to use it, please check the tool's manual. This documentation is only applicable to the Unity plugins.

8.7 Specific Pre-requisites for Unity

Itris Import/Export Tool is able to export code from Unity, without any intervention other than the setup of the **main unit** and the **Unity plugin**, and given that the Unity development environment is installed, all on the same computer.

8.8 Specific Information for Unity

8.8.1 Compatibility with the old versions of PLC Checker, PLC DocGen et PLC Converter

Considering that the import/export system is more recent than some versions of the Itris Automation products, the export format is not compatible with all of them.

If you want to use the export plugin to use PLC Checker or PLC DocGen, then you will require a XEF file. This XEF file can be found in the ZIP file created by the exporter plugin. To get it, you just have to open the ZIP file and copy/paste the XEF file.

8.9 XML export file generation from Phoenix Contact Multiprog

Phoenix Contact Multiprog application must be exported in an XML export file in order to be analyzed by PLC Checker.

You can create an XML export file with the **File** > **Export...** menu in Phoenix Contact Multiprog software.

1. In Phoenix Contact Multiprog menu, select **File** > **Export...**

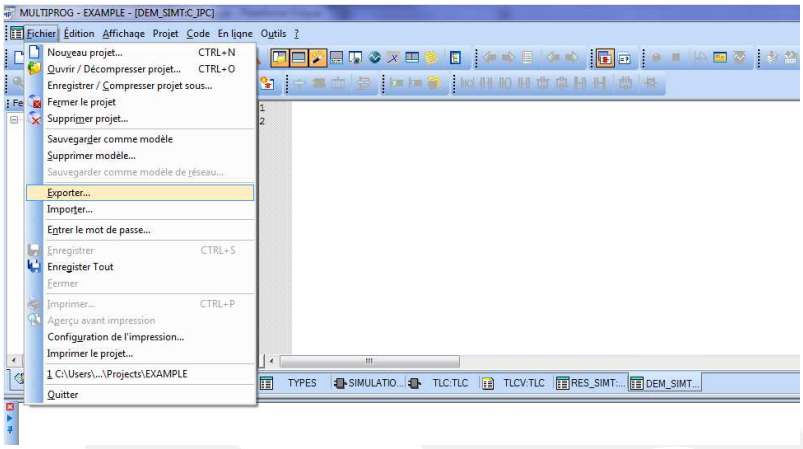


Figure 8.6: Export file

2. Choose "Export PLCopen XML file..."

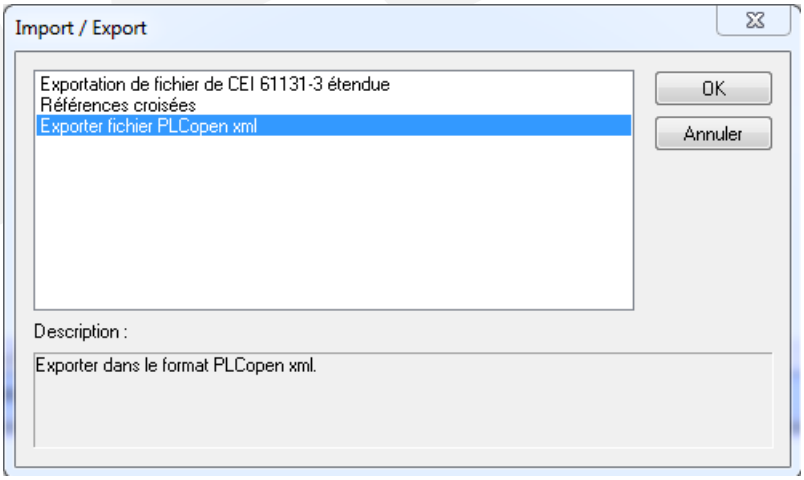


Figure 8.7: choice of extension file

3. Renames the exported file and click on **Save**

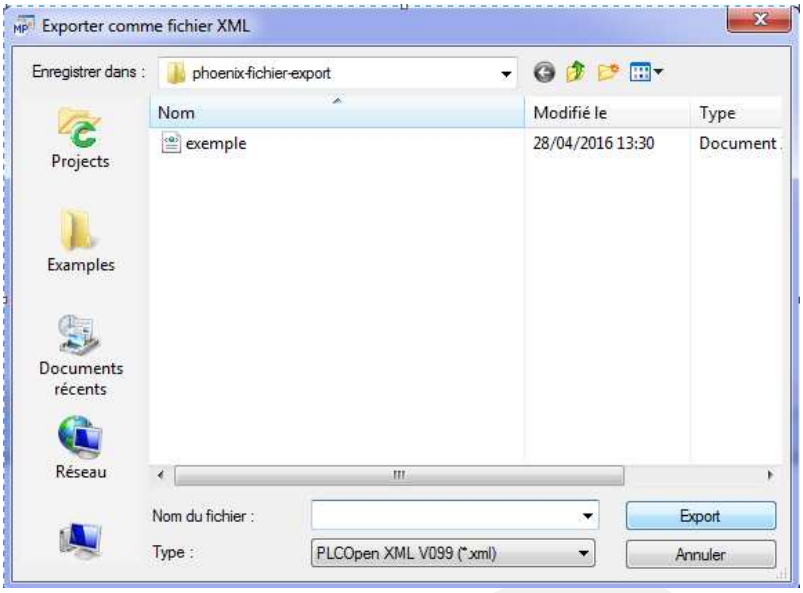


Figure 8.8: save PLCopen XML file

Before to be analyzed by the PLC Checker, the XML file must be imported in the Eclipse platform workspace.

Note : You will find some difference between files results and this depending on the version of the PLCopen file selected !

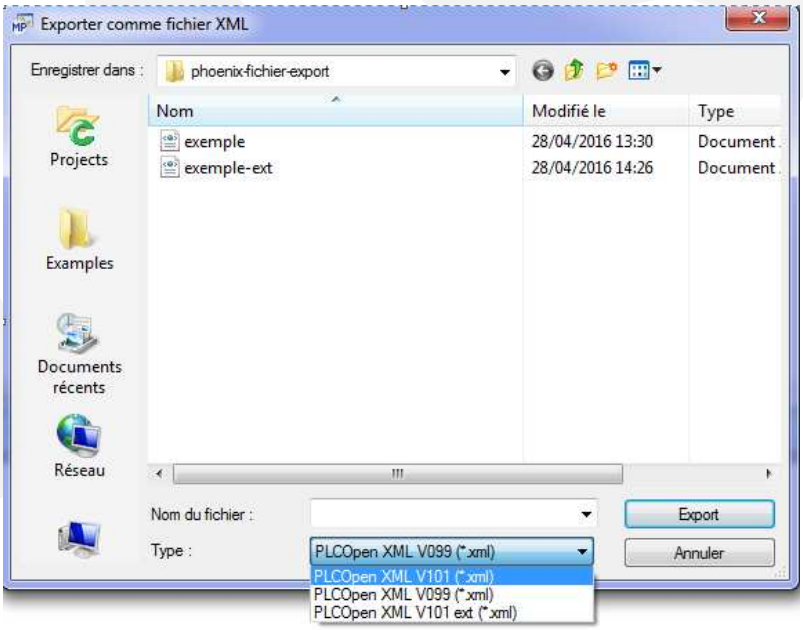


Figure 8.9: Choice of the version

Chapter 9

Concepts

9.1 Rules

This section introduce part of the concepts used for the rules files.

9.1.1 Generality

A rules file has a hierarchic structure that can be represented by a tree. The nodes of this tree are rules groups (rulesGroup). The leaves of this tree describe the rules to check or are rules groups used to define reusable filters.

9.1.2 Rules activation

The nodes and leaves of rules file can be activated or deactivated using the associate box displayed at their left. A deactivated node (and his children) will not be read nor used during PLC Checker execution. By this way, with rules file split in independent parts, it is possible to check only some parts of this rules file with PLC Checker.

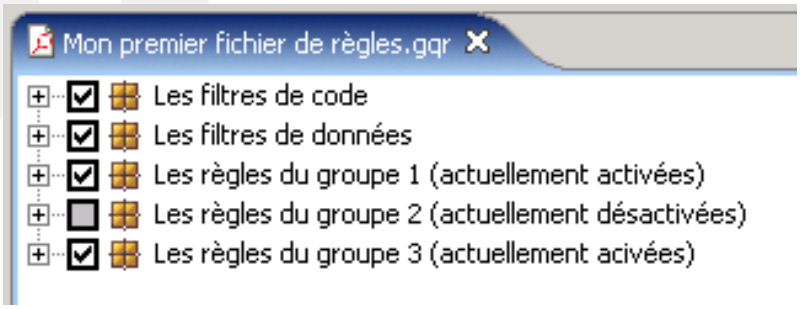


Figure 9.1: Rules activation

9.1.3 Rules name

This property is named "**name**".

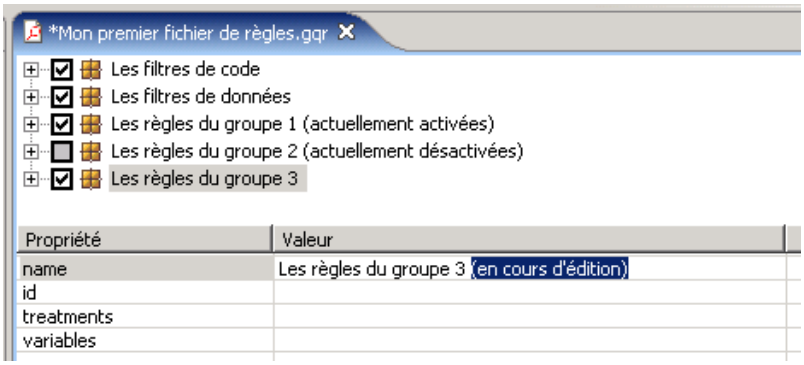


Figure 9.2: Rules naming

Each element of the rules file has a name. This is a comment that tells what the element represents. To change it, enter the name property.

Note: This name is used in the results file in the "Checked rules" section. You better have to systematically name each rules file element.

9.1.4 Rules Identity

This property, named "**id**", is used when you want to reuse, elsewhere in the rules file, the set of code or data defined by this element (via treatments and/or variables properties).

- For example, you can define the set of all remote control variables in your application.

  1. create a rules group in your rules file that will represent this set.
  2. Define his name (name property) "Define set of remote control variables".
  3. Define his ID (id property) "Remote_Ctrl".
  4. Define the set of remote controls via one or more address intervals or variable name formats in the "variables" property.
  5. When you need to add a rule that check all remote controls, you will use #Remote_Ctrl as variables property for your new rule.

Note: The ID property must be limited to alphanumeric characters (A..Z a..z 0..9) and underscore (_). The special characters are not allowed (+ - # @ : . ? * ...).

9.1.5 Code filter

This property, named "**treatments**", allows to define set of codes selected from the whole application code.

9.1.6 Data filter

This property, named "**variables**", allows to define sets of data from the whole application data.

9.1.7 To enable a rule using other rules

This property, named "**enabledIds**", contains a boolean expression of rules IDs. This allows to validate (statically, not dynamically) a rule or a rules group using one or more other nodes previously defined.

A rule or rules group is enabled when this property is true. The rules IDs used in the boolean expression are evaluated as true or false depending if their are enabled or not.

This boolean expression supports the OR, AND and NOT operators but does not support parentheses. The standard notation (ID1 OR ID2) and the post-fixed one (ID1 ID2 OR) are supported.

The rules ID used in the boolean expression have to be defined before their use in the enabledIds property.

- **Examples:** Here are some cases for the enableIds property of rule named rule3, with ID1, ID2 and ID4 the values of the ID properties of three previously defined rules named respectively rule1, rule2 and rule4:

  1. enabledIds="ID1 OR ID2" : rule3 is enabled when rule1 OR rule2 is enabled.
  2. enabledIds="ID1 ID2 OR" : rule3 is enabled when rule1 OR rule2 is enabled.
  3. enabledIds="ID1 AND ID2" : rule3 is enabled when rule1 AND rule2 are enabled.
  4. enabledIds="ID1 ID2 AND" : rule3 is enabled when rule1 AND rule2 are enabled.
  5. enabledIds="ID1 ID2 ID4 OR AND" : rule3 is enabled when rule1 AND (rule2 OR rule4) are enabled.
  6. ~~enabledIds="ID1 AND (ID2 OR ID4)"~~ : The parentheses are not supported. Unexpected result may occurs.
  7. enabledIds="" : rule3 may be enabled or not by the end user, using the check box(see rule activation section above).
  8. enabledIds="true" : rule3 is always enabled, the final user can not use the check box to disable this rule.
  9. enabledIds="false" : rule3 is always disabled, the final user can not use the check box to enable this rule (may be you better have to delete this rule, no ?).

9.1.8 To enable one rule or rules group in a list

This operation uses the "**groupIds**" property.

This "groupIds" property contains a rules ID list.

This allows to enable one and only one rule or rules group of a list.

- **Example:**
  - The three nodes whose ID are ID1, ID2 and ID3, have the following groupIds property: groupIds="ID1 ID2 ID3". During the rules file edition, the rules editor plugin ensure that only one of the three nodes is enabled.

9.1.9 Identifier for .csv format export

This property, named "**csvId**", allows to add 4 columns in the csv ("Comma Separated Value") file which is generated by the Eclipse platform when creating the result file.

Those columns contain:

- the counter of all occurrences founded under the corresponding node
- the errors counter
- the warnings counter
- the infos counter

9.1.10 Node visibility

This property, named "**visible**", allows to mask, when false, the node and his sub-tree during rules file edition.

9.1.11 Automatic Node visibility using other rules

This property, named "**visibleId**", contains a boolean expression of rules IDs. This allows to hide or make visible (statically, not dynamically) a rule or a rules group using one or more other nodes previously defined.

A rule or rules group is visible when this property is true. The rules IDs used in the boolean expression are evaluated as true or false depending if their are enabled or not.

This boolean expression supports the OR, AND and NOT operators but does not support parentheses. The standard notation (ID1 OR ID2) and the post-fixed one (ID1 ID2 OR) are supported.

The rules ID used in the boolean expression have to be defined before their use in the visibleId property.

- **Examples:** Here are some cases for the visibleId property of rule named rule3, with ID1, ID2 and ID4 the values of the ID properties of three previously defined rules named respectively rule1, rule2 and rule4:

  1. visibleId="ID1 OR ID2" : rule3 is visible when rule1 OR rule2 is enabled.
  2. visibleId="ID1 ID2 OR" : rule3 is visible when rule1 OR rule2 is enabled.
  3. visibleId="ID1 AND ID2" : rule3 is visible when rule1 AND rule2 are enabled.
  4. visibleId="ID1 ID2 AND" : rule3 is visible when rule1 AND rule2 are enabled.
  5. visibleId="ID1 ID2 ID4 OR AND" : rule3 is visible when rule1 AND (rule2 OR rule4) are enabled.
  6. ~~visibleId="ID1 AND (ID2 OR ID4)"~~ : The parentheses are not supported. Unexpected result may occurs.
  7. visibleId="" : rule3 may be visible or not by the end user, using the check box(see rule visibility section above).
  8. visibleId="true" : rule3 is always visible, the final user can not use the check box to hide this rule.
  9. visibleId="false" : rule3 is never visible, the final user can not use the check box to make this rule visible (may be you better have to delete this rule, no ?).

9.1.12 List of editable properties for protected files

The property, named "**editable**", contains the list of properties that can be changed when the rules file is protected (locked property ="true" for the first node of the rules file)

9.1.13 Errors severity

This property, named "**severity**", allows to define the way the errors are added to the result file.

The value can be "fatal", "error", "warning", "info" or "nolog".

The value "nolog" allows to only count the objects (code or data) corresponding to one or more specification defined by one or more rules.

9.1.14 "warningThreshold" property

The "**warningThreshold**" property is used to define a threshold from which a warning message is sent. When the number of object (variable or code) that does not follow the rule, cross the Threshold, then a warning is sent. Most of the times, the "severity" is then set to "nolog" in order not to sent message for each object but only one message when crossing the threshold.

- if the threshold is positiv, the warning is sent when the value is greater than the threshold absolute value. eg: if threshold=4 and value=5 then value $>$ abs(threshold) and then a warning is sent.
- If the threshold is negativ, the warning is sent if the value is lower than the threshold value absolute. eg: if threshold=-4 and value=3 then value $<$ abs(threshold) and then a warning is sent.
- If the threshold is between -1 and 1, it is considered as a percentage and the value tested is then the ratio between the number of object that does not follow the rule and the number of objects tested by this rule. eg: if threshold=0.7, nbObjectTested=10 and nbObjectFailed=8 then nbObjectFailed/nbObjectTested = 0.8 $>$ threshold = 0.7 and then a warning is sent.
- If the threshold is lower than -1 or greater than 1, it is not considered as a percentage and the value tested is then the number of object that does not follow the rule. eg: if threshold=1.1 and nbObjetFailed=2 then nbObjetFailed $>$ threshold and then a warning is sent.

9.1.15 "errorThreshold" property

The "**errorThreshold**" property is used to define a threshold from which an error message is sent. When the number of object (variable or code) that does not follow the rule, cross the Threshold, then an error is sent. Most of the times, the "severity" is then set to "nolog" in order not to sent message for each object but only one message when crossing the threshold.

- if the threshold is positiv, the error is sent when the value is greater than the threshold absolute value. eg: if threshold=4 and value=5 then value > abs(threshold) and then an error is sent.
- If the threshold is negativ, the error is sent if the value is lower than the threshold value absolute. eg: if threshold=-4 and value=3 then value < abs(threshold) and then an error is sent.
- If the threshold is between -1 and 1, it is considered as a percentage and the value tested is then the ratio between the number of object that does not follow the rule and the number of objects tested by this rule. eg: if threshold=0.7, nbObjectTested=10 and nbObjectFailed=8 then nbObjectFailed/nbObjectTested = 0.8 > threshold = 0.7 and then an error is sent.
- If the threshold is lower than -1 or greater than 1, it is not considered as a percentage and the value tested is then the number of object that does not follow the rule. eg: if threshold=1.1 and nbObjetFailed=2 then nbObjetFailed > threshold and then an error is sent.

9.1.16 "defaultResultId" property

The "**defaultResultId**" property defines where to display the error in the PLC application tree of the result file.

9.1.17 "errorMessage" property

The "**errorMessage**" property defines the error message to be displayed in case of error. This property can contain variables between characters "" and "". When error message is emitted, those variables are replaced by their values.

Here are some useful variables (respect the uppercase):

- FUNCTION: contains the name of the bloc of code related to the error.
- LOCATION: contains the name and the offset of the bloc of code.
- VARIABLE: contains the name of the variable related to the error.
- STR: is an additional information depending on the checked rule.

9.1.18 "noMatchMessage" property

The "**noMatchMessage**" property, defines the message to be displayed when the set of data or code, on which the rule is checked, is empty.

9.1.19 "noMatchSeverity" property

The "**noMatchSeverity**" property, defines the severity (info, warning, error, fatal) of the noMatchMessage. If noMatchSeverity is not defined, the severity of the rule (see "severity" property) will be used if the noMatchMessage has to be displayed.

9.1.20 "isTextEditable" property

The "**isTextEditable**" attribute takes a boolean value. If its value is true, the action "Edit pattern" appears in the menu.

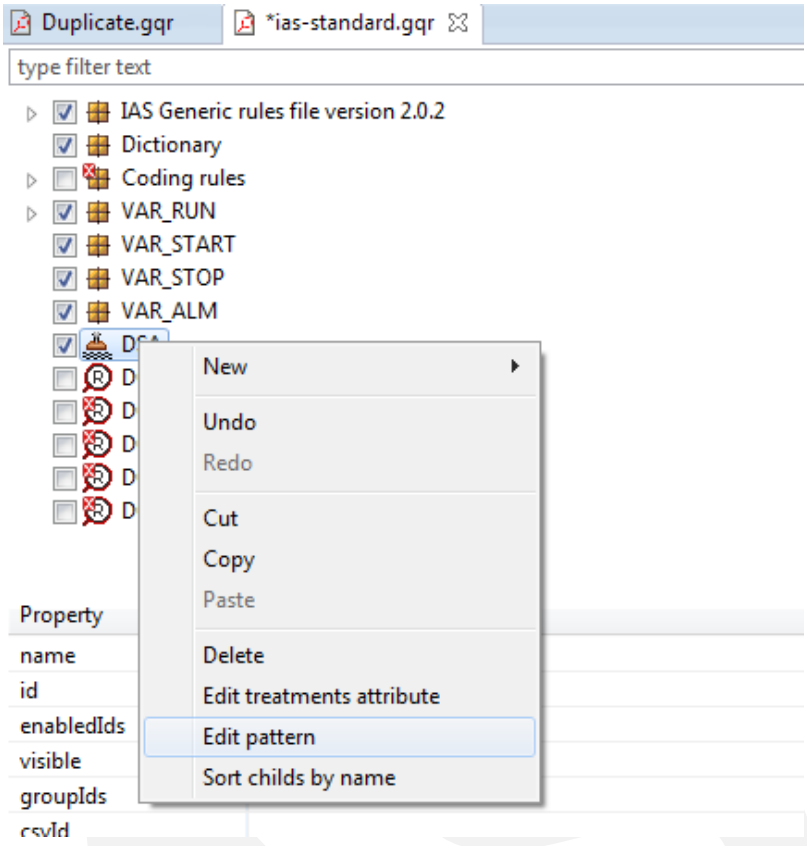


Figure 9.3: Edit pattern in the menu

The "Edit pattern" allows showing a text editor in which we can insert glips code related to the selected rule (e.g. patternMatchingCheck).

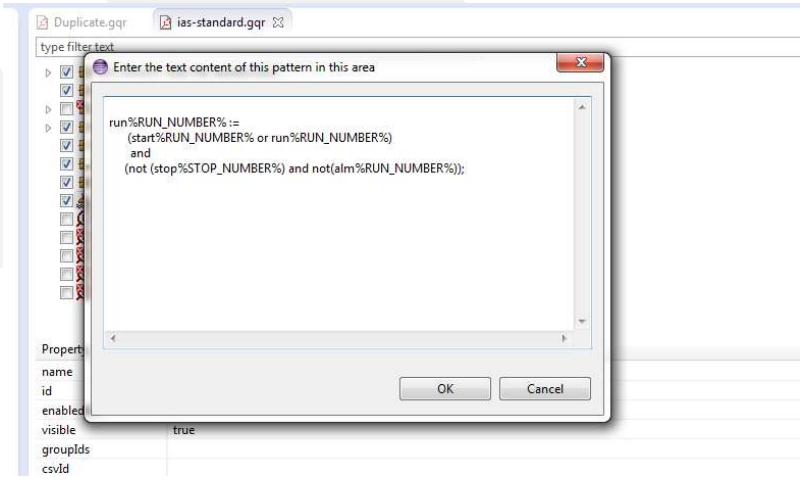


Figure 9.4: The text editor

9.1.21 Generator of attribute value

This mechanism alloww to generate automatically a value of an attribute of a node of a rules file.

It is activated when saving the rules file using the rules editor in the Eclipse platform.

To use this, an attribute named ".generator" must be added to the node. To add this ".generator" attribute, a text editor needs to be used because the rules editor display only the standard attributes.

Example:

<node attribute1="blabla" attribute2="blabla" **attribute1.generator="generation code"**>
To generate automatically the attribute1 attribute of this node, add the attribute1.generator attribute.
</node>

Language supported by the .generator attributes

Actually the rules editor supports 2 operators: UNION and List of id/value couples).

UNION This operator is used to generate automatically a set containing the union of items from the first one (FIRST_ID) to the last one (LAST_ID).

Syntax:

attribute.generator="UNION(FIRST_ID,LAST_ID)"

Behavior:

For the item whose id is FIRST_ID (<element id="FIRST_ID"/>) and for all its brothers down to the one whose id is LAST_ID, the following operation is maid:

If the item (<element id="MY_ID" enabled="true"/>) is enabled (enabled="true") then this item is added in the union, meaning the string +#MY_ID is added to the value of the generated attribute.

If no brother item is found with the LAST_ID id, then the union ends with the last brother of the item whose id is FIRST_ID.

Example:

```
<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=""/>
```

will generate the following when saving the rules file with the rules editor:

```
<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"/>
```

List of ID/value couples This operator contains a list of ID/Value couples. The first char of this list is interpreted as the separator between ids and values. Generally the semi-colon char ';' is used.

Syntax:

attribute.generator="(;.+;.+)+"

Note:

The separator (here ';') may be changed for example by '|' : attribute.generator="|ID1|+#TOTO-#TITI|ID2|+#TUTU"

Behavior:

For each IDx/VALUEx couple, if the item whose id is IDx exists and is enabled (<element id="IDx" enabled="true"/>) then the VALUEx string is added to the generated attribute.

Example:

```
<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="">
</element>
```

will generate the following when saving the rules file with the rules editor:

```
<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>
```

9.2 Filters

This section introduce the filters concept used in the rules file.

9.2.1 Generality

The filters allow to define sub-set of elements.

Two elements set are taken into account: the set of the application code, and the set of the application data. So there is two kind of filters, the code filters and the data filters:

- The code filters are defined using the **treatments** properties of the rules file elements.
- The data filters are defined using the **variables** properties of the rules file elements.

For each element to be checked by PLC Checker, the analyze is limited:

- to the sub-set defined by his **treatments** property, for the block of codes (FB/DFB/ADD_ON, FC/Routine, OB/Task, PROGRAM, SR/Subroutine, Action of SFC, . . .)
- to the sub-set defined by his **variables** property, for the data (variables, types, and all others elements).
- Example: A rule, verifying a variable is never written, gives the set of variables, in the variables sub-set defined by the **variables** property, never written in the sub-set of code defined by the **treatments** property.

9.2.2 Limitations

The **treatments** and **variables** properties can not contain space character (' ') and must begin by the '**+**' or '**-**' operator.

9.2.3 Symbols filters

A symbols filter allows to define sub-set of objects using the symbol (also called mnemonic or name) of those objects. The generic characters '*' and '**?**' are authorized and have the usual signification.

- '*' : means **1** or **more** characters
- '**?**' : means exactly **1** character
- **Example:**

- with filter DF_*, the DF_HIGHTEMP and DF_LOWTEMP are selected in the sub-set, but DF_ is not because some characters are missing after the _.
- with filter T?_*, the TC_ON and TR_TEMP_WATER are selected in the sub-set, but T_TEST is not because the third character is not an _.

9.2.4 Symbols filters using regular expressions

This kind of symbols filter allows to define sub-set of objects using the symbol (also called mnemonic or name) of those objects and using more powerful syntax than simple symbols filters.

This kind of filter is defined by a regular expression into parentheses "()". The regular expression describes the sub-set of symbols to be selected. The regular expression can contains meta-characters (having specific meaning) and quantifiers (defining quantity of previous set of characters).

Meta-characters:

- . : all characters. **Ex:**

```
(t.t.) => toto, tati etc...
```

- | : alternative. **Ex:**

```
(tata|titi) => tata or titi
```

- () : group. **Ex:**

```
(t(i|ou)ti) => titi or touti
```

- [] : characters class. **Ex:**

```
([a-cD-F]g) => ag, bg, cg, Dg, Eg ou Fg
```

- ^ : allow to define unauthorized characters. **Ex:**

```
([^a-y]z) => az, bz, ..., yz are not OK.
```

- \ : allow to use a Meta-character or a quantifier as simple character. **Ex:**

```
(\. \| \( \) \[ \] \\) => . | ( ) [ ] \
```

Quantifiers:

- * previous set of characters can appears **0** or **more** times. **Ex:**

```
(.*) => all string of characters even an empty string
```

- + previous set of characters can appears **1** or **more** times. **Ex:**

```
([0-9]+) => all numbers from 0 up to 9999999...
```

- ? previous set of characters can appears **0** or **1** time. **Ex:**

```
(test(_[a-z])?) => test, test_a, test_b, ... test_z
```

- n previous set of characters can appears exactly **n** times. **Ex:**

```
([a-z]{3}) => abc, abcd, kkkkkk, ...
```

- n, previous set of characters can appears at minimum **n** times. **Ex:**

```
([a-z]{3,}) => abc, abcd, kkkkkk, ...
```

- n,m previous set of characters can appears at minimum **n** times and at maximum **m** times. **Ex:**

```
([0-9]{2,4}) => 00, 01, ...,99, 000, 001, ..., 999, 0000, ..., 9999
```

For more explanations, see the regular expression content assistant in the search box of eclipse:

1. open eclipse
2. open a not empty text file or open an untitled text file and enter some text in this empty file.
3. open the search box (Ctrl+F or menu Edit / Find/Replace...)
4. select the "regular expression" option
5. select the "find" field
6. open the content assistant (Ctrl+Space) see the yellow light on the left of the find field

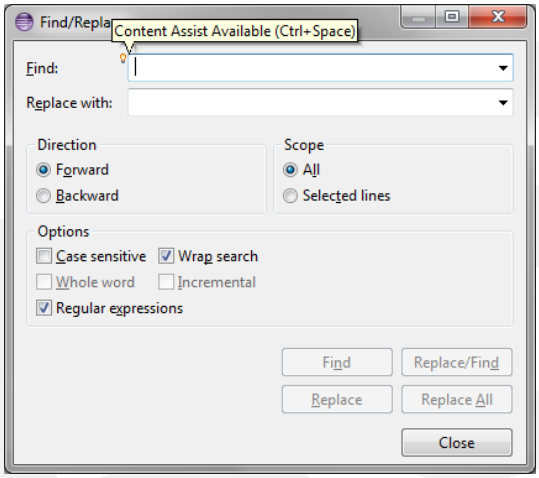


Figure 9.5: Open regular expression content assistant

7. search in the content assistant what is needed and select it to display the example

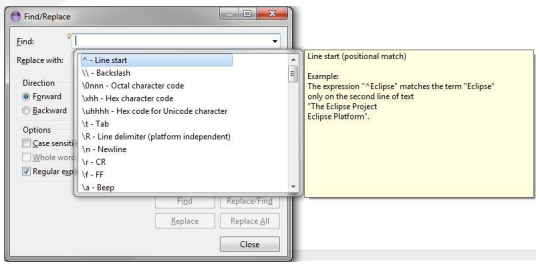


Figure 9.6: Regular expression content assistant

9.2.5 Attributes filters

Those attributes filters allow to define sub-set of objects using an attribute of the object (Data or Code). The list of available attributes depends on the kind of object and on the PLC programming tool (UNITY-PRO, PL7-PRO, STEP 7, RSLOGIX 5000).

An attribute filter always begins by the keyword "**^**" or "**!**".

The keyword "**!**" is used to search object that has the correct direct attribute value. The objects for which this attribute is not available are not selected.

The keyword "**^**" is used to search object that has the correct direct or inherited attribute value. If the attribute is not available for the object, then the attribute is also searched in the parents of the object.

The name of the attribute is put after the keyword.

The keyword "**=**" is used between the name of the attribute and the filter that defines the set of authorized values for this attribute.

The filter definition is into parentheses "()". If a regular expression is used in this filter, then the regular expression is also into parentheses "()" like for symbols filters.

The syntax is then:

- either: !<attribute_name>=(<list_of_text_filters_for_the_value_of_the_attribute>)
- or : ^<attribute_name>=(<list_of_text_filters_for_the_value_of_the_attribute>)

Examples:

- ^language=(+ladder) : used to select element (instruction, variable, procedure, ...) which is in LADDER or inside something which is in LADDER.
- !comment=(+(.*spare.*)) : used to select element (instruction, variable, procedure, ...) whose comment contains "spare"

9.2.6 Absolutes addresses filters

An absolutes addresses filter defines a sub-set using the object address.

An addresses filter appears like this: @<memory plan name>**:**<addresses interval>

It always begins by the '@' character followed by the memory plan name where to find the object (ex: MW or KD). The '**:**' character is between the memory plan name and the addresses interval. The filter ended by the limits of the addresses interval.

There are four kind of possible syntaxes for the addresses interval:

- unique address (for example: @MW:10)
- finite interval (for example: @MW:0..100)
- interval without upper limit (for example: @MW:100..*)
- the whole memory plan (for example: @MW:*)

NOTE: another way to filter an element according to its address, is to use an attribute filter to test the plc_address attribute.

9.2.7 Filters usage

To define a sub-set, rules file element of rulesGroup type are used.

This rulesGroup element must have an identifier (Id property).

To reuse this sub-set, use the "#" character before the element ID.

- **Example 1:**
  - rulesGroup named "digital inputs" as the ID: **dig_in**
  - rulesGroup named "analog inputs" as the ID: **ana_in**
  - a rule working on inputs can used the following variables property : **+#dig_in+#ana_in**
- **Example 2:**
  - rulesGroup named "digital inputs" as the ID: **dig_in**
  - rulesGroup named "analog inputs" as the ID: **ana_in**

- a rulesGroup named "inputs", with ID=**all_in**, can regroups all inputs using the variable property +#**dig_in**+#**ana_in**
- a rule working on inputs can used the following variables property : +#**all_in**

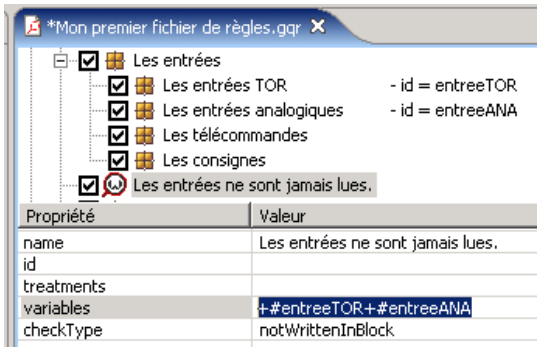


Figure 9.7: Re-use filter example

9.2.8 Filters intersection

The data and code sub-sets used for the analyze correspond to the intersection of the sub-sets defined for this rule and of those of her parents.

- **Example:**
  - the A sub-set is the father of
    - the B sub-set, which is the father of
      - the rule that works on the intersection of A and B sub-sets

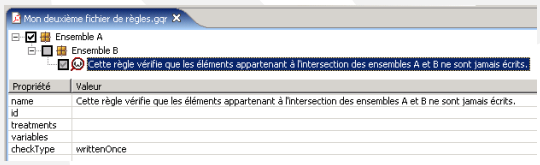


Figure 9.8: Filter intersection

9.2.9 Filters union

A sub-set can be defined as union of two or more sub-sets using the "**+**" operator.

- **Example:** +DF_*+AL_*+ER_*

9.2.10 Filters difference

A sub-set can be the difference of two sub-sets using the "**-**" operator.
In that case the **+A-B** sub-set is made of the **A** set from which the **B** set elements are removed.

- **Example:** +DF_*-DF_

9.3 Glips Attributes

This section introduces the Glips Attributes concept used in the rules file.

- **action_type** : returns the type (either function, procedure, task or reactive) of an action (= an entity which contains code and can be called). Otherwise (ex: when asking for the action_type attribute of a variable) nothing is returned. It is useful to distinguish:
  - function: which returns a value and have then to be called in an expression (ex: "a := b + myfunction(c, d); ")
  - procedure: which can be called by a calling instruction (ex: "myprocedure(a, b, c, d); ")
  - task: which can be started depending on a trigger (periodic, cyclic, trigger, ...) (ex: "start mytask trigger ... ; ")
  - reactive: which is a procedure that has a sequential evolution depending on steps and transitions (used for Sequential Function Chart language)
- **application** : returns the application name on Program node
- **artificial** : returns True if the node was created by Glips and doesn't exist in the original application.
- **author** : returns the name of the author of the application or of a prototype (FB, DFB, ADD_ON, ...)
- **auto_run** : for PL7Pro application, returns the value (1=enabled, 0=disabled) of the auto_run option.
- **backward** : on GOTO instruction, this attribute indicates if direction is backward. Ex:

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

- **calledcount** : returns the number of call of a given action/POU (procedure, function, FB, FC, Routines, Sub-Routine, ...)
- **calldepthmin** : returns the minimum depth of call of a given action/POU (procedure, function, FB, FC, Routines, Sub-Routine, ...)
- **calldepthmax** : returns the maximum depth of call of a given action/POU (procedure, function, FB, FC, Routines, Sub-Routine, ...)
- **callproc** : returns the number of function bloc in a routine.
- **comment** : contains the comment associated to a node (variableDeclaration, typeDeclaration, instruction, rung, ...)
- **comment.1** : contains the comment associated to the bit 1 of a node (variableDeclaration, ...). You can use comment[ii].oo to get the comment associated to the bit oo of the index ii of a node. Mainly useful for RSLogix5000 or UnityPro workbench (depending on version).
- **constant** : return the constant variable not localized
- **dateandtime** : as version but it contains the date and time associated to an entity
- **deadaction** : returns true if the code of the action/POU is seen as not executed (meaning POU is not called), otherwise returns false.
- **evg** : returns the value of the essential complexity (=nb of instruction flow discontinuity like GOTO, EXIT, RETURN, ..) measured on an action/POU/block of code (procedure, function, FB, FC, Routines, Sub-Routine, ...) (see literature for more explanation)
- **extvar** : number of external variables referenced in an action/POU/block of code

- **extvarref** : number of references to external variables contained in an action/POU/block of code
- **fanoutcount** : number of actions/POU/blocks of code called from a given action/POU/block of code
- **format** : depends on type of node:
  - for a list of instructions: returns its format (invisible, rung, instr_block),
  - for an affectation instruction: returns its format (Set, Reset, Affect, Not_Affect, Hash_affect),
  - for a conditional instruction (if, elsif, case, when, ...): returns its format (If_Then_Else, IfDo, Case_When),
  - for a loop instruction (loop, while, ...); returns its format (While_Loop, Loop_Loop, For_Loop).
- **fullkind** : complete Glips node kind (type, package, prototype, action...) from package to entity.
  - ex1: fullkind on the instruction "a := b + c ;" located in the following code, will return "package action if if affect" because the affect instruction is inside a IF which is inside a IF which is inside an action (here a procedure) which is inside a package:

```
package fb is
begin
   procedure My_FB ( 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_fb;
end fb;
```

  - Note: it is useful to detect the imbricated instructions ("if if if ...").
- **fullname** : contains the name(=symbol or mnemonic) of an entity prefixed by the name of all its fathers, with only small letters. In the above example, the fullname attribute for variable my_Var will return "fb.my_fb.my_var"
- **fulloriginalname** : contains the name(=symbol or mnemonic) of an entity prefixed by the name of all its fathers, with the same case (capital and/or small letters) than in the source file. In the above example, the fulloriginalname attribute for variable my_Var will return "fb.My_FB.my_Var"
- **functionalmodule** : for PL7Pro and UnityPro, it returns the name of the functional module associated to a section/sub-routine
- **g7initstepcount** : returns the number of initial steps in a SFC (=Sequential Function Chart) block. In the below example, the g7initstepcount attribute will return 2.

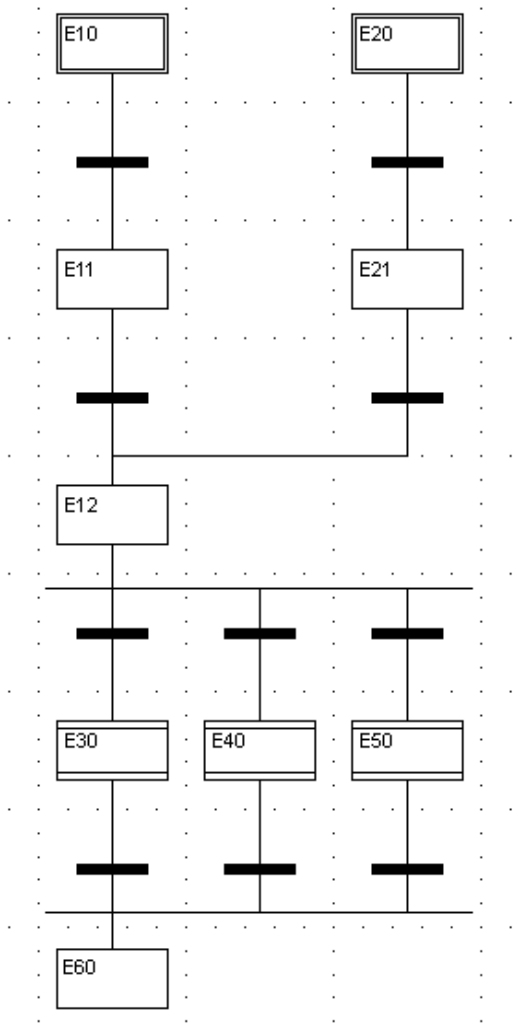


Figure 9.9: Grafcet example

- **g7stepcount** : returns the number of steps in a SFC (=Sequential Function Chart) block. In the above example, the g7stepcount attribute will return 9.
- **hasexternalreadref** : returns True if the entity has Read references outside of its scope (=where it is declared). It means if it has at least one designator pointing on the entity and which is in read mode and which is located outside of the scope where the entity was declared. Ex : returns true if a local variable of a block is read outside of the block.
- **hasexternalref** : True if entity is referenced outside of its scope (=where it is declared). It means if it has at least one designator pointing on the entity and which is located outside of the scope where the entity was declared. Ex : returns true if a local variable of a block is used outside of the block.
- **hasexternalwriteref** : True if entity has Write references outside of its scope (=where it is declared). It means if it has at least one designator pointing on the entity and which is in write mode and which is located outside of the scope where the entity was declared. Ex : returns true if a local variable of a block is written outside of the block.
- **hasreadref** : returns True if the entity has Read references. It means if it has at least one designator pointing on the entity and which is in read mode.Note: if the entity is referenced by its address then the mnemonic isn't seen as referenced in reading. In the below example, the hasreadref attribute returns true for toto because the mnemonic toto is used for reading. It returns false for titi because the mnemonic titi is only used in writing, the reading of the variable is done by address and not by mnemonic.

```
toto : bool at %MW10;
titi : bool at %MW20;
...
titi := %MW20 OR toto;
```

- **hasref** : True if entity is referenced in the code. It means if it has at least one designator pointing on the entity somewhere in the whole application.
- **haswriteref** : True if entity has Write references. It means if it has at least one designator pointing on the entity and which is in write mode.
- **hs_distinctoperator** : Halstead metric : number of distinct operator
- **hs_distinctoperand** : Halstead metric : number of distinct operand
- **hs_totaloperator** : Halstead metric : total number of operator
- **hs_totaloperand** : Halstead metric : total number of operand
- **hs_vocabulary** : Halstead metric : vocabulary
- **hs_programlength** : Halstead metric : program length
- **hs_calcprogramlength** : Halstead metric : calculated program length
- **hs_volume** : Halstead metric : volume
- **hs_difficulty** : Halstead metric : difficulty
- **hs_effort** : Halstead metric : effort
- **hs_timerequired** : Halstead metric : time required for implementation
- **hs_numberdeliverybug** : Halstead metric : number of delivered bugs
- **initvalue** : return the initial value of the variable if any
- **instancecount** : return the number of instances of a given type
- **interval** : for PL7Pro application, value of the period (eg: "T#5 MS") of a task. For a cyclic task, interval attribute is "T#0 MS". (see also tmax attribute for the watch dog of a task)
- **intrarunggoto** : on GOTO instruction, this attribute indicates if the destination label is in the same rung/network than the GOTO instruction. 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** : attribute containing the IEC61131 language (ladder, litteral, grafcet, fbd or list)
- **library** : attribute to explicit that entity belongs to a library
- **locked** : True if POU or FB is locked
- **longcomment** : attribute specific to Step7 because a block (FB, FC, OB, DB, ...) can have two comments :

1. one is the only one line comment in the symbolic/mnemonic table (see *.asc file), it is stored in the "comment" attribute.
2. one other is the long description that can be maid for this block in the source file (see *.AWL, *.SCL files), it is stored in the "longcomment" attribute

- **loopvar** : returns Yes if a variable is used as index of a loop else returns nothing
- **macro** : attribute used to distinguish the macro graphs (when this attribute contains "yes") from the others SFC Graphs. It is useful when you want to check number of initial steps that must be 1 for all SFC except for the macro SFC for which it must be 0. In the below example, the macro attribute returns "no" for the graphe1 because it's an ordinary SFC graph. It returns "yes" for the graphe2 because it's a macro SFC graph.

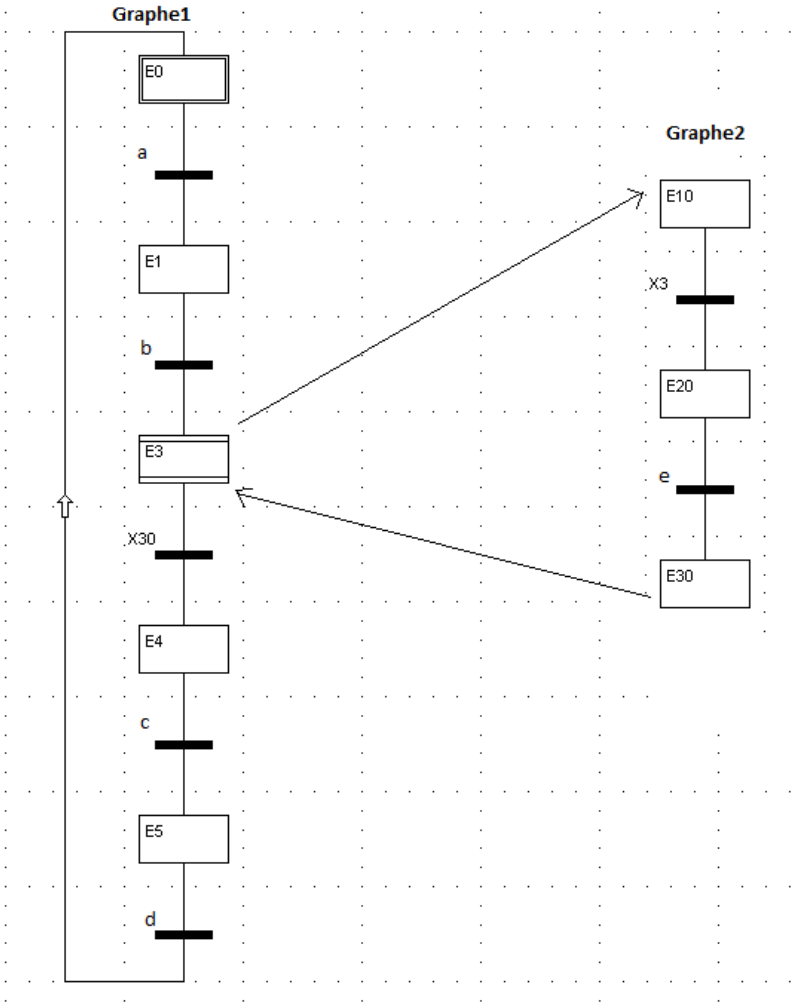


Figure 9.10: macro graphe

- **mnemonic** : On old version of plc checker, this attribute indicates that variables have mnemonics (now it should be the attribute "no_mnemonic"="true" or "false")
- **mode** : For Step7 and unity, returns variable mode (in, out, in_out, temp, stat for step7 and IN, in_out, OUT, public, private for unity). In the below example, the mode attribute returns "in" for var1 and var2. It returns "in_out" for out1.

```
procedure FB_1
    (out1 : in out int;
```

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

- **multiinstance** : On Step7, defines wether a FB is a multi-instance (multiinstance="yes") or not
- **mw_init_zero** : On PL7Pro, indicate the status of the project options to initialize to 0 the MW memory zone when starting the PLC
- **nbofbranches** : returns the number of branches by divergence in a Grafcet. In the exemple below, the nbofbranches attribute returns 4 for the first divergence and 11 for the second divergence.

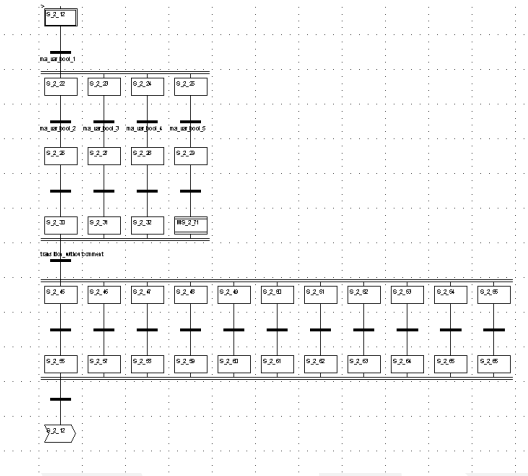


Figure 9.11:

- **networkcount** : returns the number of networks/rungs in a block of code. Note that for some languages (like literal) and/or some PLC workbenches (like UnityPro) the concept of network doesn't exist.
- **no_mnemonic** : Indicates that variables have no mnemonics (it was referenced by its topological address)
- notanattribute : this attribute doesn't exist. It is some times used in the IAS standard rules file just to create a fake rule that is always false and then raises always a message for each tested element.
- **originalname** : contains the name/symbol/mnemonic of an entity (variable, udt, function, ...) with the same case (capital and/or small letters) than in the source file. eg "My_LoCalVar". See also fullname and fullOriginalName attributes
- **parent_entity_type** : returns the type of the parent/container of an entity. Ex: in a prototype(FB, DFB, ADDON) the local variables
- **plc_address** : topological address, for example %MW12 on Unity
- **plc_family** : contains the PLC family of the given application (unity, step7, rockwell, pl7)
- **reference** : depends on type of node:
  - for an entity (variable, type, prototype, block of code, ...): returns its originalname (eg: my_Var),
  - for a reference (designator, selector): returns originalnames of the referenced entities (eg: MyFunct.my_Var),
  - for a system address (direct access to part of the memory): returns the system address (eg: "@DB[102].0,0:16"),
  - for an immediate value: returns the value (eg: "12", "false", "42.2", "hello world"),

  - for a range: returns the range (eg: "10..29"),
  - for other nodes: returns the type and the id of the node (eg: "N_INSTR_IF: 12345"). Note: the id may change frequently, so we advise not to use it for any test (filter or rule).

- **rename** : the entity renames another entity (like "alias" in Rockwell PLC)
- **run_stop** : for PL7Pro application, returns the address (eg: "%I2.0") of the input which controls the plc operating mode (run/stop). "0" means that this option is disabled.
- **rung** : return "true" for a list of instructions which is a rung/network and not for other instructions list (ex: instructions list in the THEN part of a IF)
- **signature** : returns a string characterizing a entity Glips
- **sloc** : Source Line Of Code : contains the number of instructions for each block
- **state** : returns the state of a Grafcet step ("Name_Initial" for an initial step of the grafcet, "Name_In" for a macro entry step, "Name_out" for a macro exit step, "No_Name" for a grafcet normal step). In the below example, the state attribute returns "Name_Initial" for the step E0 and "No_Name" for the others.

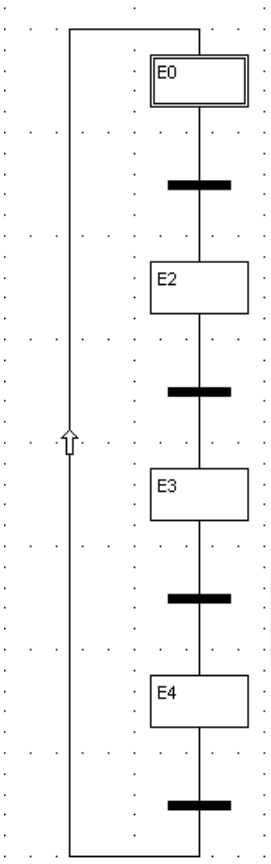


Figure 9.12: Grafcet example

- **temporary** : used to identify entities created by Glips analyzer (goal is to avoid findings on entities created by analyzer)
- **title** : For STEP7 and RSLOGIX5000, return the content of the title of rung/network

- **tmax** : for PL7Pro application, returns the delay (eg: "t#100 MS") before a watch dog is raised for a PLC cycle task duration too long. (see also interval attribute for cyclic or periodic task)
- **transition** : For unity, returns the state of a Grafcet transition
- **type** : returns the name of the type of a variable.

```
var1 : bool at %MW100;
var2 : int at %MW101;
...
if var1 then
   var2 := 10;
end if;
```

- **typeadress** : returns the address of the type of a variable (example on Step7: "FB 12", "UDT 100")
- **typeclass** : returns the class of the type of a variable (array_type, record_type, prototype_type, bool_type, . . .)
- **typecodechecksumstring** : On Unity pro, signature of DFB's code
- **typekind** : returns the kind of the type of a variable (array, record, access, enumerate. . .)
- **typesignaturechecksumstring** : On Unity pro, signature of the model of a DFB
- **unconstraint** : true if entity is unconstrained. For example a pointer in memory can be translated as an unconstrained array
- **var_mode** : returns variable mode (read, write, read_write for formal parameters; input, output for TOR, ANA, communication variables)
- **varlocation** : returns the location of a variable
- **version** : attribute that may be available on various node kinds depending on PLC kind (application, POU, FB, program)
- **vg** : returns the value of the cyclomatic complexity measured on an action/POU (procedure, function, FB, FC, Routines, Sub-Routine, . . .). It corresponds to the number of branches (eg: for a "IF THEN ELSE", number of branch is 2). It is an image of the testability of the block.

9.4 Result file

The results of the PLC Checker analyze, maid on the application file(s) using a rules file, are putted in a file with the **.gqre** extension (glips quality result).

In this file, the errors are ordered by two ways:

1. by rules order(using the checked rules tree of the rules file). Only the enabled rules are displayed.
2. and by application workshop order. Each error is attached to an element of the application workshop. In most of the cases, this element must be edited in order the error to be corrected.

9.4.1 Printing results file

To print or included the results file in a document, the results file must be open and then saved with **html** name extension.

The menu **File** > **Save as . . .** displays the "save as" dialog box.

Change name extension in the File name field, replacing **gqre** by **html** (respect lowercase) and then click **OK**. The **html** file is now in the Navigator view.

Then open the **html** file with your system editor (contextual menu **Open with** > **System editor**) and use his print (menu **File** > **Print. . .**) or copy (menu **Edit** > **Copy**) functions.

Chapter 10

The rules

1. Rules group (rulesGroup)
2. Read Check (readCheck)
3. Write Check (writeCheck)
4. Symbol check (symbolCheck)
5. Attribute check(attributeCheck)
6. Code blocks check (blockCheck)
7. Equation check (equationCheck)
8. Memory check (memoryCheck)
9. Copy/Past check (copyPastCheck)
10. FEF source file structure check (FEFStructureCheck)
11. FEF Source file attribute check (FEFAttributeCheck)
12. Label check (labelCheck)
13. Pattern matching check (patternMatchingCheck)
14. Automatic duplication of rules (duplicate)

10.1 Rules group (rulesGroup)

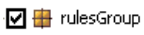


Figure 10.1: Rulesgroup

10.1.1 Usage

A rulesGroup element can be used:

1. to structure a rules file. It is only used to regroup the rules. For example a rulesGroup element can regroup all code filters, another one can regroup all rules related to written data.
2. as part of a filter created by intersection of its parents and its children.
3. to define a data or code filter that can be reused using its identifier (ID property).

10.1.2 Properties

Name (name)

See Concepts - Rules - Rule name

Identifier (id)

See Concepts - Rules - Rule identifier

Code filter (treatments)

See Concepts - Rules - Code filter

Data filter (variables)

See Concepts - Rules - Data filter

10.2 Read Check (readCheck)



Figure 10.2: Readcheck

10.2.1 Usage

A readCheck element allows to check that the number of readings made in the codes sub-set (defined by treatments properties) on the variables sub-set (defined by variables properties) is not to big (maxRead property) nor to small (minRead property).

10.2.2 Properties

Name (name)

See Concepts - Rules - Rule name

Identifier (id)

See Concepts - Rules - Rule identifier

Code filter (treatments)

See Concepts - Rules - Code filter

Data filter (variables)

See Concepts - Rules - Data filter

Minimal number of readings (minRead)

A readCheck rule is often used with a minRead of 1. In that case, the variables, from the data filter, that are NOT read in the code filter, are added to the result file. This can be used to check the application modularity.

Note : If this property is empty, then no error is generated.

Maximal number of readings (maxRead)

A readCheck rule is often used with a maxRead of 0. In that case, the variables, from the data filter, that are read in the code filter, are added to the result file. This can be used to check the application modularity.

Note : If this property is empty, then no error is generated.

Type of verification (checkType)

By default (defaultCheck), this rule checks that the number of write of the variable is in the given limits (see minRead and maxRead).

But it is also possible (readOnceVariableWrittenInOtherTaskCheck) to check that the variables written in others tasks are read 0 or 1 time (but no more than once) in the checked task. In this case the minRead and maxRead parameters are not used.

10.3 Write Check (writeCheck)

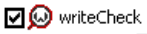


Figure 10.3: Writecheck

10.3.1 Usage

A writeCheck element allows to check that the writings, made in code sub-set (code filter) on variables sub-set (data filter), are done in the mode defined by the checkType property.

10.3.2 Properties

Name (name)

See Concepts - Rules - Rule name

Identifier (id)

See Concepts - Rules - Rule identifier

Code filter (treatments)

See Concepts - Rules - Code filter

Data filter (variables)

See Concepts - Rules - Data filter

Type of checking (checkType)

Written from one task only (writtenFromOneTaskOnly) This rule check that all writings of each variable of the variable sub-set, found in the code sub-set, can be executed from only one task. Example : a PL7-Pro application mays contain the MAST task, a FAST task and one task per events. This rule is useful to identify critical variables due to concurrents accesses.

Written only once (writtenOnce) This rule checks that each variable of the variables sub-set (variables properties) is written only once in the code filter (treatments properties). This is useful for example to check that the outputs are written from only one equation.

Set only once (setOnce) This rule checks that each variable of the variables sub-set (variables properties) is not written by something else than a SET and is not written more than once in the code filter (treatments properties).

Note: Only the SET and the ":= true;" are allowed. The variable can also not be written at all.

Reset only once (resetOnce) This rule checks that each variable of the variables sub-set (variables properties) is not written by something else than a RESET and is not written more than once in the code filter (treatments properties).

Note: Only the RESET and the ":= true;" are allowed. The variable can also not be written at all.

Written from one code block only (writtenFromOneBlockOnly) This rule checks that all writings of each variable of the variable sub-set (variables properties), found in the code sub-set (treatments properties), are in the same block of codes. Example : In a PL7-Pro application the blocks of codes are the sections, the SR and the events. This rule is useful to check application modularity.

Written from one instruction only (writtenFromOneInstrOnly) This rule checks that all writings of each variable of the variable sub-set (variables properties), found in the code sub-set (treatments properties), are in the same complex instruction. One variable written in different code blocks is excluded from this writtenFromOneInstrOnly verification. One message is emitted if no common instruction is found between the different writing of one variable.

Exemple:

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

On this piece of code, the rule writtenFromOneInstrOnly will emit a message because the two writings of the same variable var are done in two different instructions.

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

On this piece of code, the rule writtenFromOneInstrOnly won't emit any message because the different writings of the variable var are in the same IF instruction.

This rule is useful to check the application modularity.

Written before read (writtenBeforeRead) This rule check that each variable of the variable sub-set (variables properties) is written before read in the code sub-set (treatments properties). This rule can check application reactivity.

Written in block (writtenInBlock) This rule check that each variable of the variable sub-set (variables properties) is written at least once in the code sub-set (treatments properties). This rule allows to check that all application variables are used.

Not written in block (notWrittenInBlock) This rule check that each variable of the variable sub-set (variables properties) is NOT written in the code sub-set (treatments properties). This rule is useful to check application modularity.

Written a number of time between a minimum and a maximum (writtenCount) This rule checks that each variable of the variable sub-set (variables properties) are written a number of time greater than the Minimum and less than the maximum in the code sub-set (treatments properties). If the count is not in between, a message is emitted.

The following properties allows to assign the written occurrence number to check for this rule:

- minWrite (Minimum write count)
- maxWrite (Maximum write count)
- minSet (Minimum Set count)

- maxSet (Maximum Set count)
- minReset (Minimum Reset count)
- maxReset (Maximum Reset count)

Note : If a property is empty, then no error is generated.
This rule is useful to check application modularity.
Example : Check that each variable:

- is written by a SET only once
- And is written by a RESET only once
- and is never written by an other way (assignment).

for this, the properties are set as follows :

- minSet = 1
- maxSet = 1
- minReset = 1
- maxReset = 1
- minWrite = 2 (corresponding to the sum of the number of SET and RESET and assignment).
- maxWrite = 2 (corresponding to the sum of the number of SET and RESET and assignment).

10.4 Symbol check (symbolCheck)

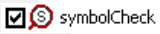


Figure 10.4: Symbolcheck

10.4.1 Usage

This rule allows to check:

1. that all application elements (data and code) have a symbol,
2. that the length of this symbol is not too short nor too long,
3. that the symbol does not contains reference to the physical address,
4. and that a group of elements (identify by the symbol) are in the given memory plan.

10.4.2 Properties

Name (name)

See Concepts - Rules - Rule name

Identifier (id)

See Concepts - Rules - Rule identifier

Code filter (treatments)

See Concepts - Rules - Code filter

Data filter (variables)

See Concepts - Rules - Data filter

Symbols minimal length (minLength)

This property defines for all application elements (code and data filters) the minimal length of its associated symbol.

Symbols maximal length (maxLength)

This property defines for all application elements (code and data filters) the maximal length of its associated symbol.

Symbol does not refer to absolute memory address (notAbsoluteCheck)

This boolean allows to select the application elements (code and data filters) whose symbol refers either to their absolute address or to their type.

- Examples :
  - W_10_MOTOR_SPEED at address %MW10 generates an error (reference to 10).
  - MW_MOTOR_SPEED at address %MW10 generates an error (reference to MW).
  - AMBIANCE at address %MB10 generates an error (reference to MB[1]).
  - AMBIANCE at address %M10 does not generate an error (no reference test for type represented by only one character)
  - PUMP_101_PRESSURE at address %M10 does not generate an error (10 is compared to 101)

Variables belong to memory zone (memFilter)

This rule allows to check that each variable of the variables sub-set (variables properties) belongs to the sub-set defined by the memFilter property.

This rules is used, for example, to check that all remote controls are in the defined memory range.

10.5 Attribute check (attributeCheck)



Figure 10.5: Attributecheck

10.5.1 Usage

This rule allows to check the specified attribute (attribute property) only on selected elements types (see elements types selection below). The list of available attributes depends on element type and on PLC programming tool (Unity-Pro, PL7-Pro, Step7, RSLogix5000).

10.5.2 Properties

Name (name)

See Concepts - Rules - Rule name

Identifier (id)

See Concepts - Rules - Rule identifier

[1] Megabyte

Code filter (treatments)

See Concepts - Rules - Code filter

Data filter (variables)

See Concepts - Rules - Data filter

Minimal length (minLength)

This property defines the minimal length of the value of the selected attribute.

Maximal length (maxLength)

This property defines the maximal length of the value of the selected attribute.

Attribute name (attribute)

This property defines the name of the attribute whose value has to be checked.

Attribute value expression (expression)

This property defines the expression that the attribute value has to match for no error to be generated.

Check only selected elements types

The following properties are use to select, when equal to true, the elements types whose selected attribute has to be checked by this rule:

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

10.6 Code blocks check (blockCheck)



Figure 10.6: Blockcheck

10.6.1 Usage

This set of rules work on the code blocks. The checking depends on the checkType property (see below).

For example, code blocks for PL7-Pro applications are the sections and SR.

10.6.2 Properties

Name (name)

See Concepts - Rules - Rule name

Identifier (id)

See Concepts - Rules - Rule identifier

Code filter (treatments)

See Concepts - Rules - Code filter

Data filter (variables)

See Concepts - Rules - Data filter

Check type (checkType)

The block contains dead code (containsCode) This check type insures that each of the code blocks defined by the treatments property contains at least one instruction.

The block is present (present) This check type insures that at least one of the code blocks defined by the treatments property is present in the application.

This is used to check that application follows a given conception plan.

The block is NOT present (notPresent) This check type insures that each of the code blocks defined by the treatments property is NOT present in the application.

This is used to check, for example, that temporary codes (adjustment, simulation, validation) are removed fro application.

The block contains dead code (containsDeadCode) This check type insures that each instruction of each of the code blocks defined by the treatments property can be executed in the application.

The set of the unreached instructions is added to the result file.

This allows to find unused code that can be source of lost of time during maintenance.

10.7 Equation check (equationCheck)

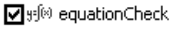


Figure 10.7: Equationcheck

10.7.1 Usage

This rule works on writing references of variables. It allows to check whether at least one read reference of the X variable (var filter) is used or not in the instruction that writes the Y variable (variables filter).

For example, it is possible to check that all outputs (S_*) writings depend on an authorization variable (aut_*). ie: s_* = f(aut_*,...)

For this we should have something like: variables="+(s_.+)" and var="+(aut_.+)"

10.7.2 Properties

Name (name)

See Concepts - Rules - Rule name

Identifier (id)

See Concepts - Rules - Rule identifier

Code filter (treatments)

See Concepts - Rules - Code filter

Data filter (variables)

See Concepts - Rules - Data filter

This is the selection of the variables whose writing references will be checked.

Type of checking (checkType)

Default check (defaultcheck) This is the default (and historic) verification done by equationCheck: it checks that when elaborating some element from a variable class, elements from an other class are used in the corresponding equation.

Immediate value usage check (noimmediatevaluecheck) When doing this check, no need to use other fields. It will raise findings when ever constant information is found in the code. Integer values -1, 0 and 1 are authorized. Boolean values are authorized.

No equality / difference between floats check (nofloatequalitycheck) This check detects when there is an equality (or difference) operator between floats. It generates findings for each node N_Expr_Egal or N_Expr_Diff when type on left or right are detected of Float_Type (Root_Type).

No equality / difference between integers check (nointegerequalitycheck) This check detects when there is an equality (or difference) operator between integers. It generates findings for each node N_Expr_Egal or N_Expr_Diff when type on left or right are detected of Integer_Type (Root_Type).

No comparison between pointers check (nopointercomparisoncheck) This check detects when there is a comparator operator between pointers. It generates findings for each node N_Expr_Inf, N_Expr_Infegal, N_Expr_Sup, N_Expr_Supegal when type on left or right are detected of Access_Type (Root_Type).

Used variable filter (var)

This filter allows to define the set of variables whose read references are searched in the writing references of the variables defined by the previous filter (variables filter).

10.8 Memory check (memoryCheck)

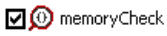


Figure 10.8: memoryCheck

10.8.1 Usage

This rule checks the memory location of variables in the different memory plans. It allows to check whether some variables declarations are overlapping each other (overlapStrict) or if written variables are overlapping each other (overlapLazzy), or if a given memory plan (defined by the memFilter filter) contains at minimum a given percentage of free locations (see free propertie).

10.8.2 Properties

Name (name)

See Concepts - Rules - Rule name

Identifier (id)

See Concepts - Rules - Rule identifier

Code filter (treatments)

See Concepts - Rules - Code filter

Data filter (variables)

See Concepts - Rules - Data filter

Memory plan filter (memFilter)

This filter (memFilter) allow to select the tested memory plan.

Minimum Free space (free)

This property defines the minimum ratio of the desired free space in each selected memory plans (see memFilter). ex: free="10%" if we want at least 10% of free space in each filtered memory plan.

Check type (checkType)

Variables Declarations overlap (overlapStrict) This type of checking will search if there are multiple variable declarations that concern the same memory location. The addresses, offsets and size of the variables, and also the overlapped memory plan (ex: MW and MD) are taken in account.

Written variables overlap (overlapLazzy) This type of checking will search if there are any written variable whose location in memory overlaps another written variable. The addresses, offsets and size of the variables, and also the overlapped memory plan (ex: MW and MD) are taken in account. The overlapped variable declaration is authorized only if only one of the variable is written and not the others. It allows for example to write a status word and then to read bits of this status word even if the status word variable and the bits variables have memory location that overlap each other.

Check minimum spare available (available) This type of checking will search if there is at least a given percentage (see Free property) of the memory plan that is free (meaning contains no declarations)

10.9 Copy/Past percentage check (copyPastCheck)

10.9.1 Usage

This rule measure a copy/past ration on the whole application export file.

10.9.2 Properties

Name (name)

See Concepts - Rules - Rule name

Identifier (id)

See Concepts - Rules - Rule identifier

Code filter (treatments)

See Concepts - Rules - Code filter

Data filter (variables)

See Concepts - Rules - Data filter

Size of copy/past (tokenLength)

This property allow to adjust the number of tokens in the sets of token to be checked for copy/past. A too small number of tokens will lead to detect a lots of copy/past, while a too big number of tokens will lead to no copy/past detection at all. A 100 tokens size is generally used. Example: in the following code "Total := a + rest ;" there are 6 tokens which are : "Total", ":=", "a", "+", "rest" and ";"

Check type (checkType)

Strict copy/past detection (cpd_symbols) This type of checking search if the structure AND the symbols of the two sets of tokens are identical. Example : "a := b + c ;" is a copy of "a := b + c ;", because the structure AND the symbols are the same.

Structure copy/past detection (cpd_no_symbols) This type of checking search if the structure AND the symbols of the two sets of tokens are identical. Example : "e := f + g ;" is a copy of "a := b + c ;", because the structure are the same.

10.10 Check FEF program structure in source file (FEFStructureCheck)

img /var/www/wikitris/data/media/doc-en/qualimetre/icon_fefstructurecheck.png

10.10.1 Usage

Obsolete rule. It was used for early PL7Pro application checking to verify the structure of the sections directly in the FEF export file, without any translation in the GLIPS tree.

10.10.2 Properties

Name (name)

See Concepts - Rules - Rule name

Identifier (id)

See Concepts - Rules - Rule identifier

Code filter (treatments)

See Concepts - Rules - Code filter

Data filter (variables)

See Concepts - Rules - Data filter

TODO

10.11 Check attributes value in export source file (FEFAttributeCheck)



Figure 10.9: FEFAttributeCheck

10.11.1 Usage

ONLY for PL7Pro or UnityPro application: This rule allow to search, directly in the export source file and without any translation in a GLIPS tree, for a given attribute (checkedAttributeName) of a given type of object (fefObjectName) and to verify the value of this attribute (see checkedAttributeValue and checkType properties). It is helpful when we need to test an attribute which is not already translated in GLIPS tree.

10.11.2 Properties

Name (name)

See Concepts - Rules - Rule name

Identifier (id)

See Concepts - Rules - Rule identifier

Code filter (treatments)

See Concepts - Rules - Code filter

Data filter (variables)

See Concepts - Rules - Data filter

Object type selection (fefObjectType)

This property define the type of object on which the attribute will be searched. (ex: "languageExtension" which is a xml node in some XEF export files)

Checked Attribute name (checkedAttributeName)

This property define the name of the attribute to search on each object whose type is the one defined by the fefObject-Type property. (ex: "multiToken" which is one of the attribute of the "languageExtension" object, see above).

Checked Attribute value (checkedAttributeValue)

This property define the value that has to be compared with the value of the founded attribute (see checkedAttributeName), if any.

Check type (checkType)

This property define if the value of the attribute must by equal of notEqual to the value defined in the rule (checkedAttributeValue).

Filtered Attribute name (filterAttributeName)

This property define the name of the attribute to be verified on each object whose type is the one defined by the fefObjectType property, to see if the object has to be checked or not. It helps to filter a sub-set of object to be checked when object type is not precised enough.

Filtered Attribute value (filterAttributeValue)

This property define the value that has to be compared with the value of the founded filter attribute (see filterAttributeName), if any.

Filter type (filterType)

This property define if the value of the filter attribute must by equal of notEqual to the filter value defined in the rule (filterAttributeValue).

10.12 Label check (labelCheck)

Figure 10.10: labelCheck

10.12.1 Usage

This rule allows to check different properties (see checkType property) of labels of a group of blocs of codes.
The checking of the continuity, increasing, boundaries of the labels concern only labels that end with a number (ex: L251).

10.12.2 Properties

Name (name)

See Concepts - Rules - Rule name

Identifier (id)

See Concepts - Rules - Rule identifier

Code filter (treatments)

See Concepts - Rules - Code filter

Data filter (variables)

See Concepts - Rules - Data filter

Type of checking (checkType)

Label is absent (absence) This case checks that all labels described in the variables property, are NOT present in any of the blocs of code defined in the treatments property.

Label is present (presence) This case checks that all labels described in the variables property, are present in at least one of the blocs of code defined in the treatments property.

Labels are continuous (continuity) This case checks that all labels are continuous (difference with previous = 0 or 1) in the blocs of codes defined in the treatments property.

Labels are increasing (increasing) This case checks that all labels are increasing (difference with previous >= 1) in the blocs of codes defined in the treatments property.

Labels are bounded (boundary) This case checks that all labels are bounded (boundaries excluded) between a minimum (minAddr property) and a maximum (maxAddr property) in the blocs of codes defined in the treatments property.

Labels are bounded (boundaryIncluded) This case checks that all labels are bounded (boundaries included) between a minimum (minAddr property) and a maximum (maxAddr property) in the blocs of codes defined in the treatments property.

Labels are unique (unicity) This case checks that all labels are unique, in the blocs of codes defined in the treatments property.

10.13 Pattern matching check (patternMatchingCheck)

Figure 10.11: patternMatchingCheck

10.13.1 Usage

This rule allows to check the presence or the absence of a pattern of code in each bloc of code of the set of blocs of code defined in the treatments property.
The pattern is described in a text field as son of the patternMatchingCheck rule node. This pattern can be edited if the isTextEditable attribute is true (see isTextEditable property).

10.13.2 Properties

Name (name)

See Concepts - Rules - Rule name

Identifier (id)

See Concepts - Rules - Rule identifier

Code filter (treatments)

See Concepts - Rules - Code filter

Data filter (variables)

See Concepts - Rules - Data filter

Pattern (text field son)

This field contains, in Glips language (structured language), the code describing the pattern that needs to be checked. This pattern mays contain some jokers to let some latitude to the checked code.

Joker for the expressions The joker for the expressions is:

```
$...$
```

- between the two dollars ('$'), the PLC Checker filters are used to describe what is allowed.
- use two dollars ("$$") to signify that every expression is allowed.

Examples:

- `$$ <=> Any expression`
- `$+toto$ <=> Any entity (or designator of an entity) named "toto".`
- `$+@M:800$ <=> Any variables located in the memory plan M at offset 800.`
- `$+(test.+)-!comment=(-(demo.*))$ <=> Any entity (or designator of an entity) named "test*" and whose comment starts by "demo".`

Joker for the instructions The joker for the instructions is:

```
[ ]{}
```

- brackets ("[ ]") contains the description of the allowed instruction(s)
- braces ("") contains the description of how many times the previous allowed instructions must be present. The syntax is the same than for the regular expressions:
  - `{n} exactly n times`
  - `{n,} at least n times`
  - `{n,m} between n and m times`

Examples:

- `[ $+out_*$ := $$ ; ]{3} <=> 3 consecutive affectations of variables named out_*.`
- `[$$]{5,9} <=> between 5 and 9 instructions.`

10.14 Automatic duplication of rules (duplicate)

Figure 10.12: duplicate

This mechanism uses the <duplicate/> node in the rules file to duplicate automatically some rules or groups of rules.

10.14.1 Execution of the duplication

The duplicate nodes are interpreted by the PLC Checker during the analyze (and not by the rules editor of the Eclipse platform before to launch the analyze). So the rules file (*.gqr) will not be modified when it contains one or more <duplicate/> nodes.

10.14.2 Definitions

A duplicate node is used to define a list of values to be used (ex: A, B, C). The size of this list gives the number of duplications (ex : here 3 duplication: the first with A, the second with B and the third with C). A value to be used can contain multiple fields (ex: (A,titi), (B,toto), (C,tata)).

The different ways to create those lists and the associated number of duplication are listed in the duplication types.

A duplicate node must be defined before to be used.

The same duplicate node can be used to do different things (see Duplications usages).

In order to use a duplicate node, a PATTERN must be used in a rule (or rulesGroup) and will be replaced by the different values (defined by the duplicate node) when the rule (or rulesGroup) is duplicated.

10.14.3 Syntax

The duplicate node may contains some specific attributes (id, filter, catch, initial, increment, count, alphanumeric, values, file, conditions).

The "id" attribute contains the identifier (by convention a string like [A-Z][A-Z_0-9]*) of the duplicate node. ex: id="ID_VALUE"

The other attributes are used depending on the duplication types.

Each time a node (ex: a rule or a rulesGroup) of the rules file needs to be duplicated or modified according to a duplicate node, the following conditions must be fulfilled:

1. an attribute dynamic="true" needs to be added to this node (the rule or the rulesGroup to be duplicated or modified),
2. the PATTERN to be replaced must be defined. ex: %ID_VALUE%, or %ID_VALUE.2%, ... See other examples,
3. the PATTERN must be added to the value of one (or more) attribute(s) of this node (the rule or the rulesGroup to be duplicated or modified).

The PATTERN syntax defines:

1. wether the node must be duplicated or modified. See Duplication usages.
2. and the content of the string that will replace the PATTERN.

In case when a value contains multiple fields, the number of the field to be used is indicated in the PATTERN after the duplicate node id. ex: the PATTERN %ID_VALUE.2% will use the 2nd field of the different values defined by the duplicate node whose id="ID_VALUE".

10.14.4 Behavior

Duplication types

A duplication node defines the list of values used to replace the PATTERN and then the corresponding number of duplication (or modification) of each node that use the associated PATTERN. The list of values is maid:

- EITHER according to the detection of items in the program: The number of duplication and the different values used for duplication are defined by:
  - the regular expression contains in the **filter** attribute of the duplicate node (ex: filter="+[(motor|valve)([0-9]+)]"),
  - the catch group (ex: "N" for the N-th catch group), or combination ("N,M" or "NxM") of catch groups identified in the regular expression (attribute **catch**) and
  - the different values of those catch groups founded in the different items of the program following the filter.
- OR according to a predefined list:
  - The values are listed in the value attribute and splatted by a space (ex: value="val1 val2 val3 ... valn")

- OR according an increment: The values are calculated
  - starting, for the first value, by an integer initial value defined by the **initial** attribute (ex: initial="10")
  - and creating each time a new value by adding to the previous one the value of the **increment** attribute (ex increment="5")
  - until the amount of generated values reaches the value of the **count** attribute (ex: count="6")
  - and counting in:
    - decimal base (attribute **alphanumeric**="false") (ex: initial="10" increment="5" count="6" alphanumeric="false", will generate values 10, 15, 20, 25, 30, 35)
    - or hexadecimal base (attribute **alphanumeric**="true") (ex: initial="10" increment="5" count="6" alphanumeric="true", will generate values 10, 15, 1A, 1F, 24, 29)
- OR according the content of an external file:
  - The external file to be used is defined by the **file** attribute (ex: file="my_file.csv")
  - The number of duplication correspond to the number of lines, in this file, following the conditions defined by the **conditions** attribute
    - A condition contain the column number (eg: 1, 2, 3, ...) and the text filter (simple or regular expression) to be applied in order to select or not a raw of the file
    - multiple conditions can be combined together using the operators AND, OR and NOT and using the polish inverted notation
    - example 1 : conditions="(4,+X) (5,-Y*) AND", will select only raws whose column 4 contains the string X (or x because no case sensitivity) AND whose column 5 contains a string that does NOT start by Y (or y because no case sensitivity)
    - example 2 : conditions="(4,+(F(B|C)_.+)) (5,-(STD[0-9]+_.+)) OR NOT", will NOT select the raws whose column 4 contains a string starting with FB_ or FC_ OR whose column 5 does NOT start by STDxx_ (with xx equal one or more digits (0-9))
  - The values used for the duplication depend on the content of the selected lines. For the replacement of the PATTERN, depending on the file type (CSV or TXT), it is possible to use the content of a given column only (ex: PATTERN.3 for the 3rd column of a CSV file) or of the complete line (ex: PATTERN or PATTERN.1 for the complete line of a TXT file).

Duplication type choice

For a given duplicate node, the choice of a type of duplication is made according the presence of some attributes in this duplicate node. The priority is given:

1. to the duplication according to the detection of items in the program (if **filter** attribute is present)
2. else to the duplication according to a predefined list (if **value** attribute is present)
3. else to the duplication according to an increment (if **increment** attribute is present)
4. else to the duplication according to the content of an external file (if **filepath** attribute is present)
5. else no duplication is executed.

Duplication usages

The same duplication type (defining the list of values) can be used:

- to **duplicate some XML nodes** (rule or rulesGroup) of the rules file tree. So for each value of the duplication list, the node (rule or rulesGroup) will be duplicated and each PATTERN related to this duplication will be replaced by one of the field of the value to be used. See the XML node duplication examples.
- **AND/OR** to replace the PATTERN in a string by a **string concatenation** of the different values of the duplication list using optional prefix and/or suffix. See the string concatenation examples.

The distinction between those two usages is maid by the format of the PATTERN:

- %ID% or %ID.12% for the node duplication
- %ID$pref$suf% or %ID.12$pref$suf% for the string concatenation

OPERATING PRINCIPLE The duplicate nodes are visited in the apparition order in the rules file. For each duplicate node containing an id attribute (ex: id="PATTERN"), the nodes of the rules file are visited in the apparition order in the rules file (father is visited before its sons).

- **NODE DUPLICATION CASE**:,
  IF a node contains an attribute named dynamic and equal to "true" AND IF one of its attributes contains the %PATTERN% pattern (considering PATTERN is the id of the duplicate node),
  THEN, for each value in the duplication list of value, this node and its sons are duplicated and each string "%PATTERN%" in this duplicated sub-tree is replaced by the current value of the list.
  The number of duplication of the sub-tree depends on the length of the list of values of the duplicate node.
- **STRING CONCATENATION CASE**:
  IF a node contains an attribute named dynamic and equal to "true" AND IF one of its attributes contains the %PATTERNprsu% pattern (considering PATTERN is the id of the duplicate node),
  THEN, each "%PATTERN%" string will be replaced by the concatenation of all values in the duplication list, each value is preceded by the "pr" prefix and followed by the "su" suffix.
  The number of concatenated values depends on the length of the list of values of the duplicate node.

IF a combination of catch groups ("1,3" or "1×3") is used, instead of a unique given catch group ("1" or "3"), in the catch attribute of a duplication according to the detection of items in the program, THEN the list of values is in fact a list of sets of values. Indeed, to one duplication corresponds a set made with the value of each catch group. In this case, the catch group to be used for replacement has to be chosen between the available catch groups. This is maid by adding in the pattern, after the id of the duplication node, the rank of the desired catch group (ex: %PATTERN.2% to use the 2nd value in the set of values of each duplication of the duplication node whose id="PATTERN"). The string %PATTERN.x% is replaced by the content of the x-th value of the set of values.

For example, in order to use the 3rd column of a CSV file, we need:

- a duplication node whose duplication type is according to an external file (meaning containing the attribute filepath="...") and which contain an id attribute (ex: id="My_ID")
- AND a pattern:
  - %My_ID.3% for a node duplication
  - Or %My_ID.3$$% for a string concatenation with no prefix and no suffix

In all cases, %PATTERN% and %PATTERN.1% are equivalent and replaced by the same value.

10.14.5 Examples

XML node duplication

- Incremental duplication:

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

will give after duplication:

```
<duplicate id="PATTERN1" initial="1" increment="2" count="8"
alphanumeric="true"/>
<element dynamic="true" variables="+engine15" />
<element dynamic="true" variables="+engine17" />
```

```
<element dynamic="true" variables="+engine19" />
<element dynamic="true" variables="+engine1B" />
<element dynamic="true" variables="+engine1D" />
<element dynamic="true" variables="+engine1F" />
<element dynamic="true" variables="+engine21" />
<element dynamic="true" variables="+engine23" />
```

- Incremental duplication:

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

will give after duplication:

```
<duplicate id="PATTERN1" initial="1" increment="2" count="8"
alphanumeric="false"/>
<element dynamic="true" variables="+engine15" />
<element dynamic="true" variables="+engine17" />
<element dynamic="true" variables="+engine19" />
<element dynamic="true" variables="+engine21" />
<element dynamic="true" variables="+engine23" />
<element dynamic="true" variables="+engine25" />
<element dynamic="true" variables="+engine27" />
<element dynamic="true" variables="+engine29" />
```

- Duplication according to a list of value

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

will give after duplication:

```
<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 according to detection of items in the program, **WITHOUT** combination of catch groups:

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

will give the following after duplication, if these items, motor2, motor34 and motorGT, exist in the program:

```
<rulesGroup variables="+motor*">
  <duplicate id="PATTERN3" filter="+(motor([0-9]+))" catch="1"/>
</rulesroup>
<element dynamic="true" variables="+motor2" name="test mot. 2" />
<element dynamic="true" variables="+motor34" name="test mot. 34" />
```

Note that there is no duplication for motorGT because it doesn't correspond to the filter motor([0-9]+) of the duplicate node.

- Duplication according to detection of items in the program, **WITH** combination of catch groups: Considering, the rules group named IG with its duplicate nodes:

```
<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" />
</rulesroup>
```

suppose the program contains the following variables: i1_g4, i1_g3, i2_g4, then each duplicate node will use the same list of entities (the variables i1_g4, i1_g3, i2_g4 mentioned previously). Nevertheless, the lists of sets of values will be different for each duplicate node:

1. for the duplicate IGN: ("1","4"), ("1","3"), ("2","4") - there will be 3 duplications for which the strings %IGN.1% and %IGN.2% can be replaced by "1" and "4", then by "1" and "3", and finally by "2" and "4":

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

   will be replaced by

   ```
   <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. for the duplicate IN: "1", "2" - there will be 2 duplications for which the strings %IN% (or %IN.1%) can be replaced by "1" and then "2":

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

   will be replaced by

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

3. for the duplicate GN: "3", "4" - there will be 2 duplications for which the strings %GN% (or %GN.1%) can be replaced by "3" and then "4":

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

   will be replaced by

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

4. for the duplicate IGXN: ("1","3"), ("1","4"),("2","3"),("2","4") - there will be 4 duplications for which the strings %IGXN.1% and %IGXN.2% can be replaced by "1" and "3", then by "1" and "4", then by "2" and "3", and finally by "2" and "4":

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

   will be replaced by

   ```
   <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"/>
   ```

String concatenation

- Concatenation of values

  1. for the values (A, B, C),
     (a) the PATTERN %ID$$% will be replaced by ABC
     (b) the PATTERN %ID.1$$% will be replaced by ABC
  2. for the values with multiple fields ((A,titi), (B,toto), (C,tata)),
     (a) the PATTERN %ID$$% will be replaced by ABC

(b) the PATTERN %ID.1$$% will be replaced by ABC
(c) the PATTERN %ID.2$$% will be replaced by titititototata

- Concatenation of values with prefix

1. for the values (A, B, C),
   (a) the PATTERN %ID$Pref$% will be replaced by PrefAPrefBPrefC
   (b) the PATTERN %ID.1$Pref$% will be replaced by PrefAPrefBPrefC
2. for the values with multiple fields ((A,titi), (B,toto), (C,tata)),
   (a) the PATTERN %ID$Pref$% will be replaced by PrefAPrefBPrefC
   (b) the PATTERN %ID.1$Pref$% will be replaced by PrefAPrefBPrefC
   (c) the PATTERN %ID.2$Pref$% will be replaced by PreftitiPreftotoPreftata

- Concatenation of values with suffix

1. for the values (A, B, C),
   (a) the PATTERN %ID$$Suf% will be replaced by ASufBSufCSuf
   (b) the PATTERN %ID.1$$Suf% will be replaced by ASufBSufCSuf
2. for the values with multiple fields ((A,titi), (B,toto), (C,tata)),
   (a) the PATTERN %ID$$Suf% will be replaced by ASufBSufCSuf
   (b) the PATTERN %ID.1$$Suf% will be replaced by ASufBSufCSuf
   (c) the PATTERN %ID.2$$Suf% will be replaced by titiSuftotoSuftataSuf

- Concatenation of values with prefix and suffix

1. for the values (A, B, C),
   (a) the PATTERN %ID$Pref$Suf% will be replaced by PrefASufPrefBSufPrefCSuf
   (b) the PATTERN %ID.1$Pref$Suf% will be replaced by PrefASufPrefBSufPrefCSuf
2. for the values with multiple fields ((A,titi), (B,toto), (C,tata)),
   (a) the PATTERN %ID$Pref$Suf% will be replaced by PrefASufPrefBSufPrefCSuf
   (b) the PATTERN %ID.1$Pref$Suf% will be replaced by PrefASufPrefBSufPrefCSuf
   (c) the PATTERN %ID.2$Pref$Suf% will be replaced by PreftitiSufPreftotoSufPreftataSuf

Chapter 11

Rules file creation and edition

11.1 Rules file creation

To create a rules files, follow this steps:

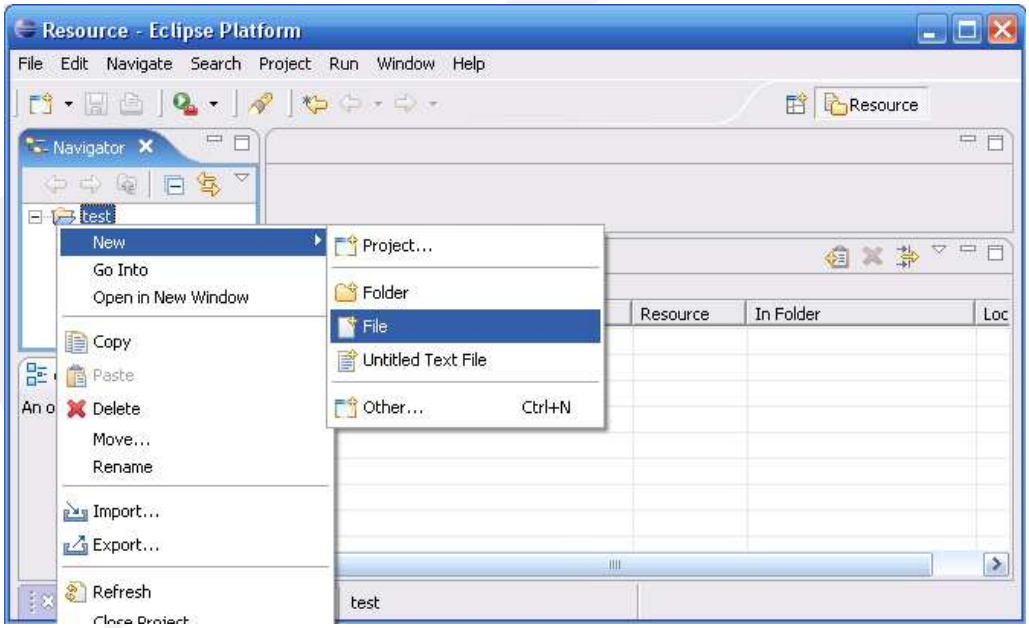


Figure 11.1: New file creation

1. In the Eclipse Navigator view, use **right click** on the project or folder were you want to create the rules files.

2. In the contextual menu, select **New** > **File**.

3. Enter the file name with extension "**.gqr**" (for example, My_rules_file.gqr). The extension must be **gqr** (glips quality rules) and must respect the lowercase.

4. Click on **Finish**.

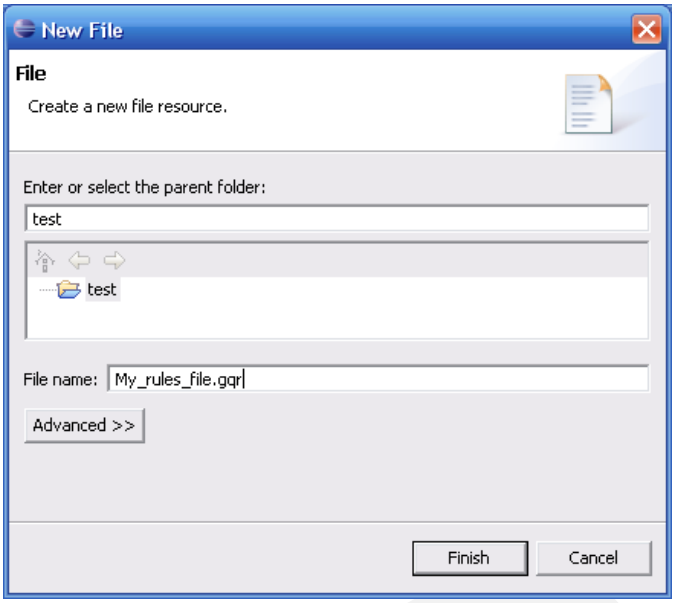


Figure 11.2: New rules file creation

A complete rules file is created. This file already contains basics rules.

The file is opened in the PLC Checker rules editor. You have to define the code sections (initialization, inputs acquisition, outputs emission, etc...), and also the variables types (inputs, outputs, alarms, etc...). If necessary, you'll have to define the addresses of the supervision tables.

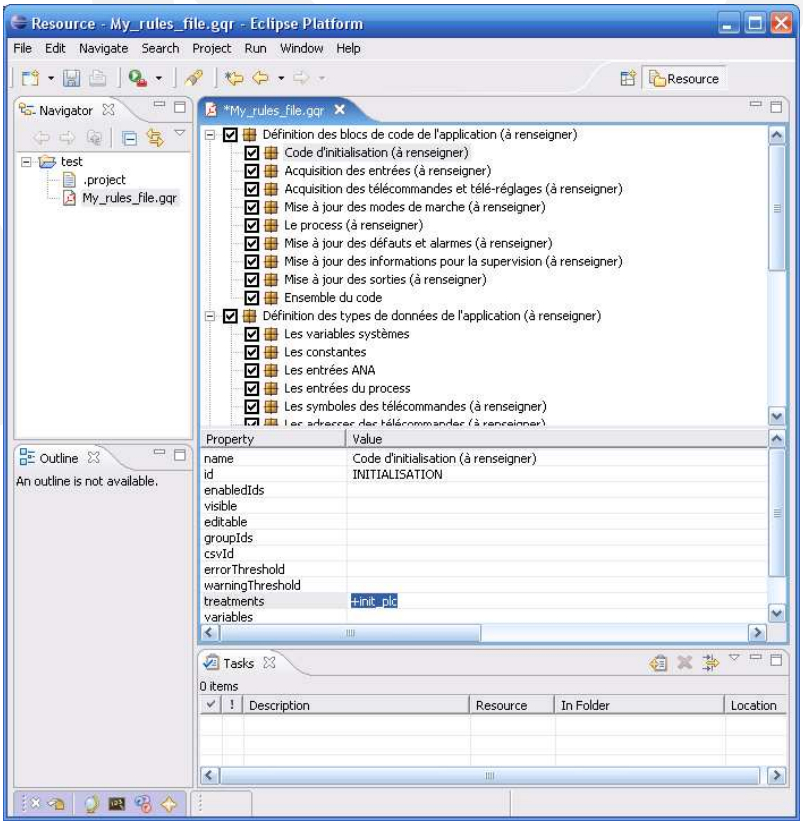


Figure 11.3: The new rules file

IMPORTANT Before to modify a treatments property (code sections) or a variables property (variables types), you'll have to read the Concepts - Filters section, in order to take account of the formalism used for those properties.

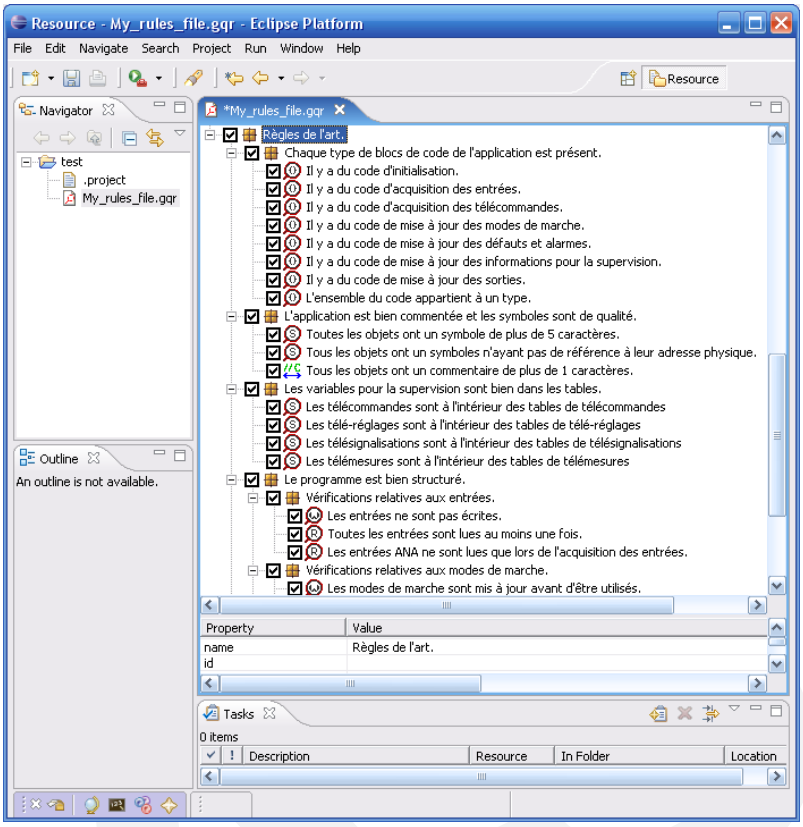


Figure 11.4: Basic rules included in the new rules files

11.2 Add a rule

All operations on rules are available in the contextual menu obtained by a **right click** in the rules editor.

Use **New** to add a new rule.

- **New** > **Sibling** > Before: add a rule before the selected one.
- **New** > **Sibling** > After: add a rule after the selected one.

If the selected rule is a rulesGroup, then you can also add a child rule.

- **New** > **Child** > First: add a Child in head of the list of children of the selected rulesGroup.
- **New** > **Child** > Last: add a Child in tail of the list of children of the selected rulesGroup.

Then choose the rule type to be added (rulesGroup, writeCheck, symbolCheck, commentCheck, blockCheck, readCheck, . . .).

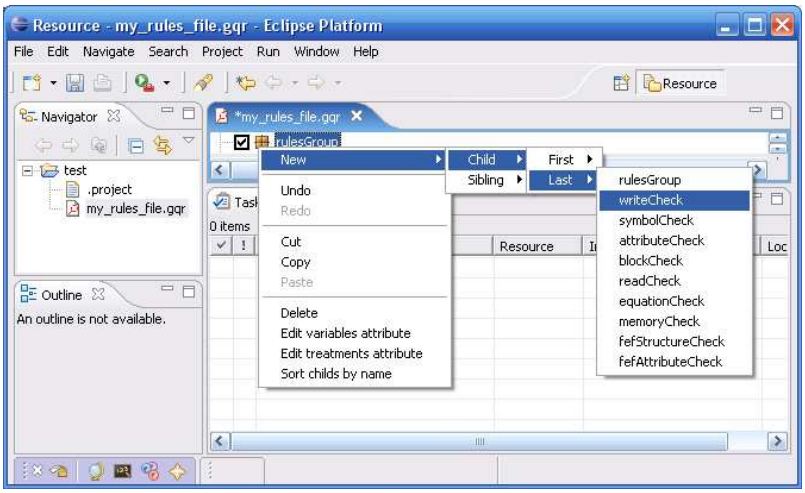


Figure 11.5: New rule creation

11.3 Cancel modifications

Use **Undo** in contextual menu, to go back and cancel modifications. You can redo modifications using the **Redo** in the contextual menu.

Note : When you come back up to the state saved on disk, the contextual menu **Undo** is disabled.

11.4 Copy and past rules

A set of contiguous rules with the same father can be copy using the **copy** command in the contextual menu.

The **Cut** command enable to copy and cut the selection. you can **Past** one or more times the previously copied selection. The rules selection is pasted after the rule selected.

Note : If the selected rule is a rulesGroup, and if it does not contain any child, then the rules selection is pasted as child of the selected ruleGroup.

11.5 Delete rules

use the **Delete** command in the contextual menu, in order to remove the rules selection. The rules selection is not copied.

11.6 Operations on rules properties

You can **copy**, **cut**, **past** properties contains, using the contextual menu displayed by a right-click on property field when it is edited (cursor displayed).

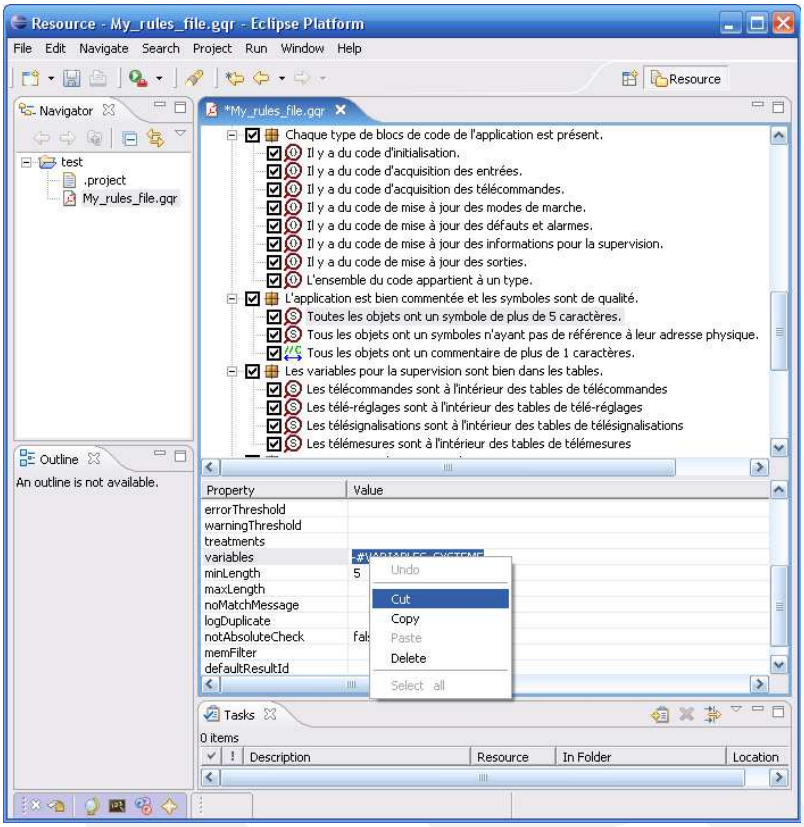


Figure 11.6: Copy-Past rules properties

11.7 Printing rules file

Website

On the website, the rules file locate in a program directory can be converted in pdf format using the menu **PLC Checker** > **Generate**.

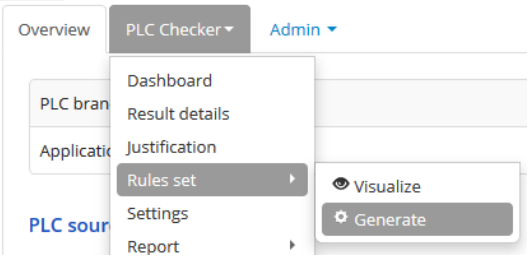


Figure 11.7: Generate PDF for rules file

Eclipse platform

In the Eclipse platform, to print or included the rules file in a document, the rules file must be saved with name extension **html**.

The menu **File** > **Save as ...** displays the "save as" dialog box.

Change name extension in the File name field, replacing **gqr** by **html** (respect lowercase) and then click **OK**. The **html** file is now in the Navigator view.

Then open the **html** file with your system editor (contextual menu **Open with** > **System editor**) and use his print (menu **File** > **Print...**) or copy (menu **Edit** > **Copy**) functions.

Chapter 12

Final license contract for PLC Checker

The license contract can be found at this address: http://www.automationsquare.com/licenses.html

IAS