APPLICATION EXAMPLE

AC500 V3 PRO STATIC ANALYSIS

USAGE AND BENEFITS FOR CODE OPTIMAZIATION

Contents

1 Disclaimer

A. For customers domiciled outside Germany /

Für Kunden mit Sitz außerhalb Deutschlands

„Warranty, Liability:

The user shall be solely responsible for the use of this products described within this file. ABB shall be under no warranty whatsoever. ABB's liability in connection with application of the products or examples provided or the files included within this products, irrespective of the legal ground, shall be excluded. The exclusion of liability shall not apply in the case of intention or gross negligence. The present declaration shall be governed by and construed in accordance with the laws of Switzerland under exclusion of its conflict of laws rules and of the Vienna Convention on the International Sale of Goods (CISG)."

„Gewährleistung und Haftung:

Der Nutzer ist allein für die Verwendung des in diesem Dokument beschriebenen Produkte und beschriebenen Anwendungsbeispiele verantwortlich.

ABB unterliegt keiner Gewährleistung. Die Haftung von ABB im Zusammenhang mit diesem Anwendungsbeispiel oder den in dieser Datei enthaltenen Dateien - gleich aus welchem Rechtsgrund - ist ausgeschlossen. Dieser Ausschluss gilt nicht im Falle von Vorsatz oder grober Fahrlässigkeit. Diese Erklärung unterliegt Schweizer Recht unter Ausschluss der Verweisungsnormen und des UN-Kaufrechts (CISG)."

B. Nur für Kunden mit Sitz in Deutschland

„Gewährleistung und Haftung:

Die in diesem Dokument beschriebenen Anwendungsbeispiele oder enthaltenen Dateien beschreiben eine mögliche Anwendung der AC500 bzw. zeigen eine mögliche Einsatzart. Sie stellen nur Beispiele für Programmierungen dar, sind aber keine fertigen Lösungen. Eine Gewähr kann nicht übernommen werden.

Der Nutzer ist für die ordnungsgemäße, insbesondere vollständige und fehlerfreie Programmierung der Steuerungen selbst verantwortlich. Im Falle der teilweisen oder ganzen Übernahme der Programmierbeispiele können gegen ABB keine Ansprüche geltend gemacht werden.

Die Haftung von ABB, gleich aus welchem Rechtsgrund, im Zusammenhang mit den Anwendungsbeispielen oder den in dieser Datei enthaltenen Beschreibung wird ausgeschlossen. Der Haftungsausschluss gilt jedoch nicht in Fällen des Vorsatzes, der groben Fahrlässigkeit, bei Ansprüchen nach dem Produkthaftungsgesetz, im Falle der Verletzung des Lebens, des Körpers oder der Gesundheit oder bei schuldhafter Verletzung einer wesentlichen Vertragspflicht. Im Falle der Verletzung einer wesentlichen Vertragspflicht ist die Haftung jedoch auf den vertragstypischen, vorhersehbaren Schaden begrenzt, soweit nicht zugleich ein anderer der in Satz 2 dieses Unterabsatzes erwähnten Fälle gegeben ist. Eine Änderung der Beweislast zum Nachteil des Nutzers ist hiermit nicht verbunden.

Es gilt materielles deutsches Recht unter Ausschluss des UN-Kaufrechts."

2 Introduction

2.1 Scope of the document

This document describes the usage and benefits of the pro static analysis tool. This includes the settings as well as results of the analysis. The Application example provides some faults in the code which is throwing warning and errors.

2.2 Compatibility

The application example explained in this document have been used with the below engineering system versions. They should also work with other versions, nevertheless some small adaptations may be necessary, for future versions.

- Automation Builder 2.3.0 or newer
- Pro Static Analysis license
- Usable only for V3 PLCs

2.3 Overview

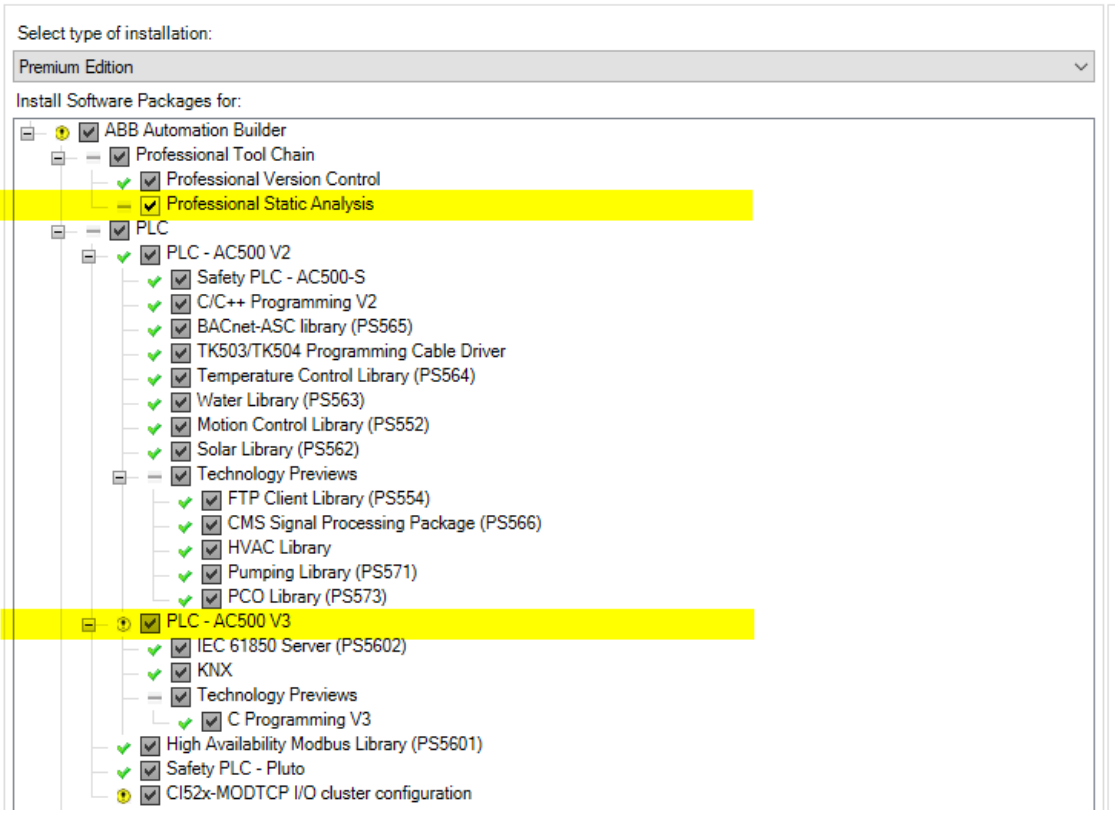
The Pro Static Analysis tool can be used to check the source code based on rules and naming conventions. The Pro Static Analysis tool should be used during development. There possible weaknesses of the code can be found and fixed before the application is used in the field.

The static analysis can automatically run each time when the code is generated. This way weaknesses can be found even in the beginning of programming. The issues which can be found here save a lot of time which otherwise is needed for debugging.

3 Installation and licensing and location

3.1 Installation

During the installation of Automation Builder 2.3.0 or later you are asked to select the components you want to install. Later on, you can also install the Professional Static Analysis with the installation manager. Please make sure that the Professional Static Analysis as well as the PLC – AC500 V3 are selected.



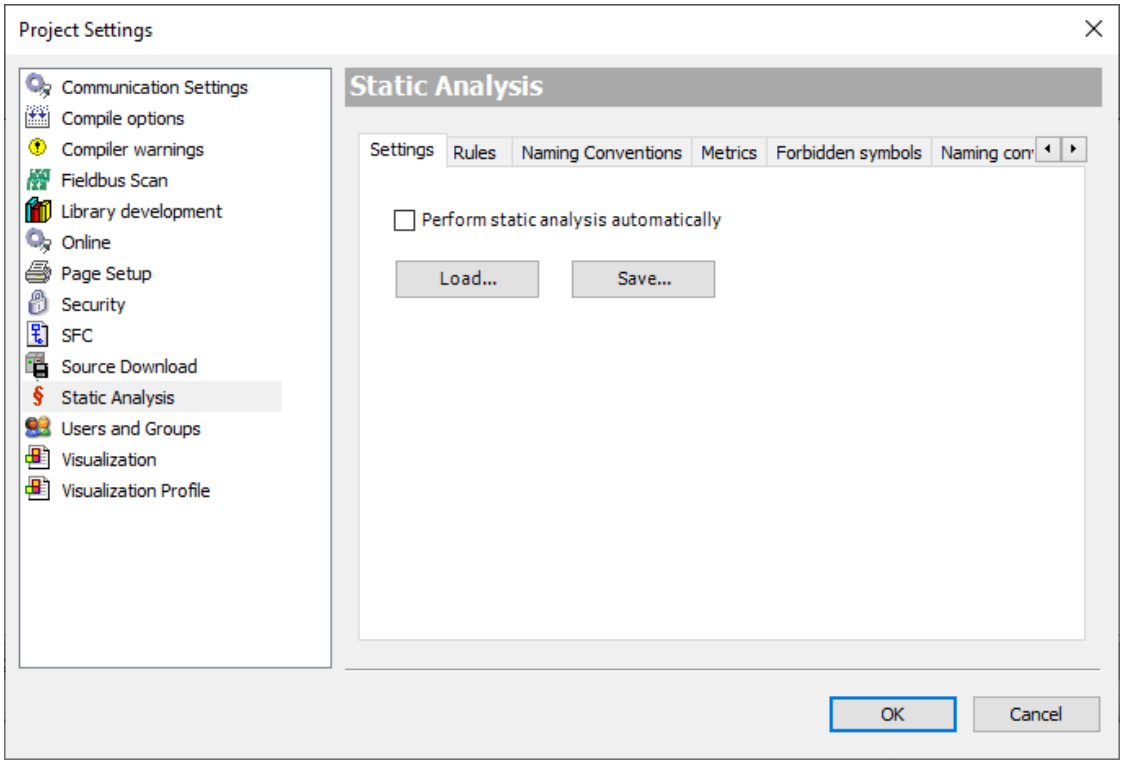
3.2 Licensing

The Pro Static Analysis is not included in the Premium license. In addition, the DM210-PSA or DM217-PSA-NW license is required. The license can be installed similar to the normal licenses by closing all applications and selecting Tools, Install additional License. The license manager will guide you through the activation.

3.3 Location

3.3.1 Configuration

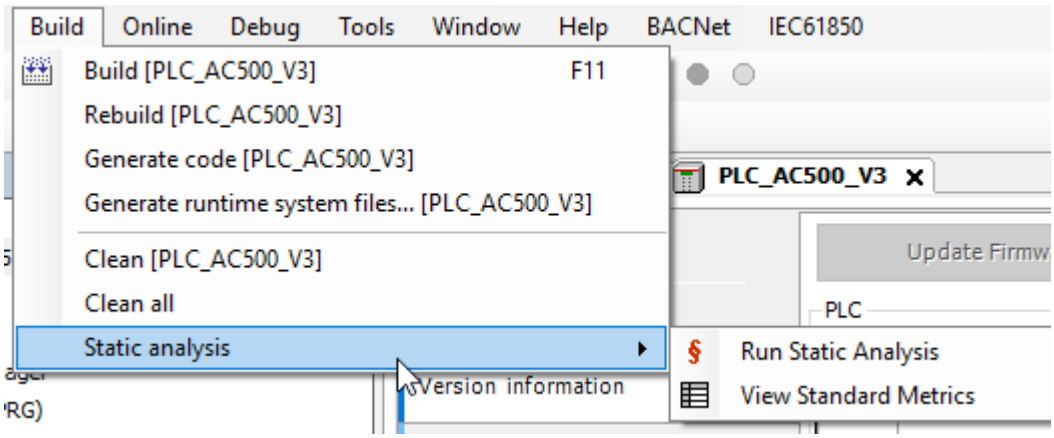
The Configuration of the Static Analysis is in the Project settings.



In case only “Static Analysis Light” is listed here the package is either not installed or not licensed.

3.3.2 Execution

Below Build “Static Analysis” is added. Here the user can run the static analysis or view the standard metrics of the project.



4 Pro Static Analysis Settings

This chapter gives an overview about the settings which can be made. Not all settings are listed in this chapter. The complete list can be found in the online help in the section “Engineering Interfaces and Tools”

4.1 Settings

In the tab Setting the user can select the checkbox “Perform static analysis automatically”. If this check box is selected the static analysis will be performed before downloading or by running generate code. Furthermore, the user has the possibility to save the current Static Analysis settings as file or load already saved settings. The settings for this project are also attached to this application example.

4.2 Rules

The tab rules are the biggest one. Here the user has more than 100 rules which can be checked by the Static Analysis.

For each rule the user has three possibilities.  ,  and  . A not checked box means this rule is not used. Rules checked red throw errors in the message view which also prevent a download. By checking the rule again, it becomes yellow. If this rule is violated in the program a warning is thrown.

The rules which are checked in this project are not explained in detail. The description for each rule as well as a short example can be found in the online help.

4.3 Naming Conventions

Depending on the variable type the user can select a Prefix. For example, each BOOL or Bit variable has to start with *x*, each byte with *b*, each word with *w* and each INT with *i*. The Prefix for all types can be specified.

This can not only be done for types but also for scopes, POUs and DUTs.

4.4 Metrics

Standard metrics can be defined which are reported as error. For each limit a lower and upper limit can be defined. Each Metric which has an upper and lower limit can later be used to display a Kiviot Diagram of the POUs

4.5 Forbidden symbols

In this section it is possible to define keywords and symbols that must not be used in the project code. The forbidden code is not used in this example.

4.6 Naming Conventions (2)

In the second part of the naming conventions the user can specify that the first character after the prefix must be upper letter case. Furthermore, it is possible to expect recursive or combined prefixes for mixed types.

5 Automation Builder Examples

5.1 Static Analysis

In the PLC_PRG a short program is written which can be compiled without any errors or warnings.

PLC_PRG

```
1   //This is the main program called by the task
2   PROGRAM PLC_PRG
3   VAR
4       iLoop : INT;            //This is the variable for loops
5       iLoopEnd : INT := 10;   //This is the end of the loop but not defined as constant
6       iArray: ARRAY [0..5] OF INT; //This is an array;
7       xTest : INT;            //This variable has the wrong prefix
8       xNoComment : BOOL;
9       xNotUsed : BOOL;        //This is a comment //This is a nestet comment
10  END_VAR
11  VAR CONSTANT
12
13  END_VAR
```

```
1   FOR iLoop := 0 TO iLoopEnd BY 1 DO
2       //This loop is counting to 10 but the maximum index of the array is 5
3       iArray[iLoop] := iLoop;
4   END_FOR
5
6   WHILE xTest < iLoopEnd DO
7       iArray[xTest] := xTest;
8   END_WHILE
9
10  IF xNoComment = TRUE THEN
11      xNoComment := FALSE;
12  END_IF
```

Messages - Total 0 error(s), 0 warning(s), 0 message(s)

Build | 0 error(s) | 0 warning(s) | 0 message(s)

Description
------ Build started: Application: PLC_AC500_V3.Application -------
The application is up to date
Compile complete -- 0 errors, 0 warnings

But when performing the static analysis seven errors and one warning are thrown.

Description		Project	Object	Position
Static analysis...				
§ SA0033: Unused Variable 'xNotUsed'		V3_St...	PLC_PRG [PLC_AC500_V3: PLC Logic: Application]	Line 9 (Decl)
§ NC0014: Invalid variable name 'xTest'. Expect prefix 'i'		V3_St...	PLC_PRG [PLC_AC500_V3: PLC Logic: Application]	Line 7 (Decl)
§ SA0012: Variable 'iLoopEnd' could be declared as constant		V3_St...	PLC_PRG [PLC_AC500_V3: PLC Logic: Application]	Line 5 (Decl)
§ SA0162: Missing comment for 'xNoComment'		V3_St...	PLC_PRG [PLC_AC500_V3: PLC Logic: Application]	Line 8 (Decl)
§ SA0163: Nested comment 'This is a nestet comment'		V3_St...	PLC_PRG [PLC_AC500_V3: PLC Logic: Application]	Line 9 (Decl)
§ SA0081: Upper border of a for loop must be a constant value		V3_St...	PLC_PRG [PLC_AC500_V3: PLC Logic: Application]	Line 1, Column 1 (Impl)
§ SA0062: Uses of TRUE or FALSE in expressions		V3_St...	PLC_PRG [PLC_AC500_V3: PLC Logic: Application]	Line 10, Column 1 (Impl)
§ SA0041: Possible loop invariant code 'xTest < iLoopEnd'		V3_St...	PLC_PRG [PLC_AC500_V3: PLC Logic: Application]	Line 6, Column 1 (Impl)
Static analysis complete -- 7 errors, 1 warning				

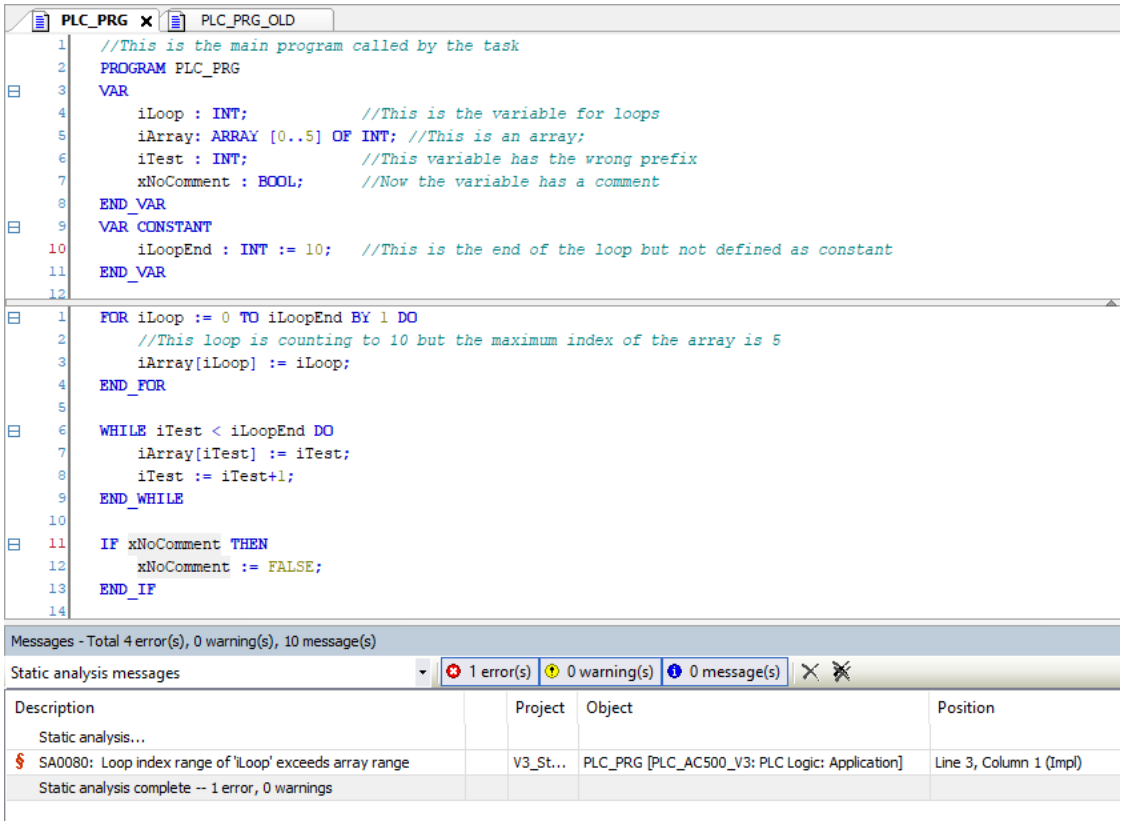
The first message is related to the variable xNotUsed which is declared but never used in the program. Depending on the program the variable should either be used or deleted. The second message is related to the integer variable xTest which has x instead of i as prefix.

The third message is related to iLoopEnd which is only read and never written. So it can be constant. The variable xNoComment is not commented. Furthermore, xNotUsed has a nested comment. This is shown as warning.

In addition to the messages related to the declaration part three errors are thrown in the Implementation. In the very first line the variable iLoopEnd is used. As this variable could change during runtime it is recommended to use either a constant value or a hard coded number. In line ten a Boolean variable is compared with TRUE. This is not needed and can be simplified.

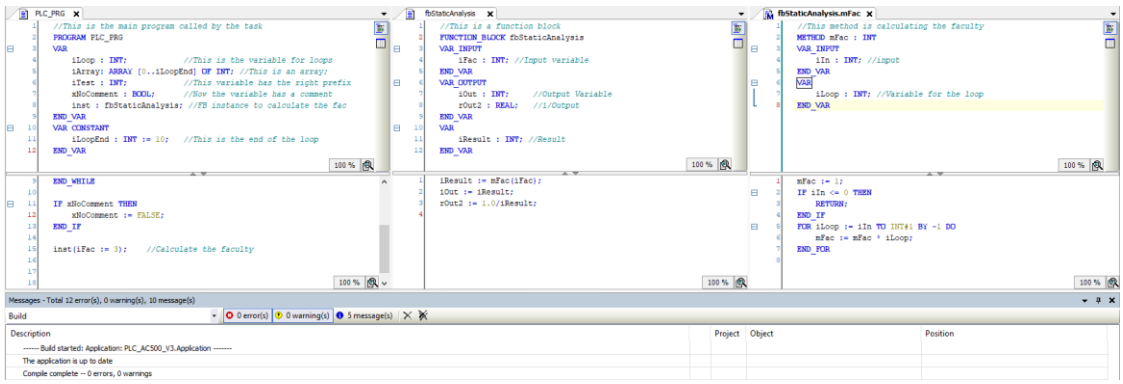
The last error message is related to the While loop. The message is loop invariant code ‘xTest < iLoopEnd’. This means the variables xTest is never written inside the loop. That indicates that the while loop could be endless. In this example xTest has to be incremented inside the loop.

After changing these errors, the code can be compiled and the analysis executed again. The static analysis is messaging a new error message.

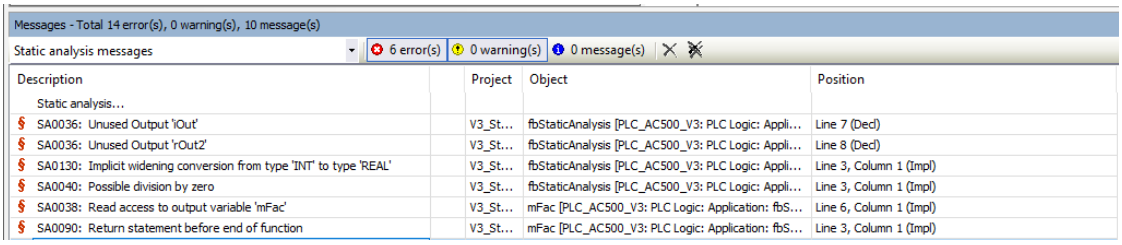


To solve this error message the upper border of the array is exceeded to *iLoopEnd*. Afterwards the static analysis is giving no errors or warnings.

In addition, the main program a function block is written to calculate the faculty of an input variable. As visible in the screenshot below on the left side the function block is instanciated and called. In the middle the Function Block implementation and on the right the Method implementation is shown. This program can be compiled without error or warning.



After running the static analysis some errors are thrown.



The first two messages show that the outputs of the function block are never used. So they should be assigned in the PLC_PRG. The third message shows the implicit conversation to REAL in the function block. This can be avoided by doing an explicate type conversion with INT_TO_REAL.

The fourth error advises that a division by 0 is possible which could cause an exception. As the return of the method could be zero, an if condition should be used to check the result against 0.

The second to last message is showing that an output variable is read in the program. This can be avoided by using a local variable. The last message is related to the return. As this is return is intended to have an output when the input is <0 either an if clause has to be added or the error should be ignored by the analysis. To avoid having an error message at this position the rule can be disabled for this line. This is described in the next chapter.

5.2 Pragmas

In case warning and errors are thrown and the user checked them and decided that this is intended because of interacting with other components or intended behavior it is possible to deactivate them. Therefore, the pragma <analysis> is used.

With <analysis -99> the rule Return statement before end of function can be disabled. Please use <analysis +99> to enable the rule again.

It is also possible to enable and disable multiple rules at once. The pragma

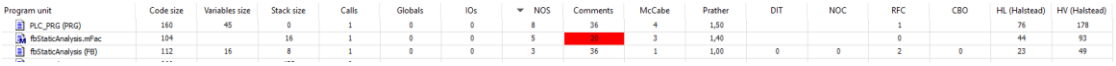
<analysis -10 -18> is disabling SA0010 (Array with only one component) and SA0018 (Unusual bit access).

With the attribute naming the naming conventions can be turned of and on.

{attribute ‘naming’ := ‘off’} switches the naming conventions off.
{attribute ‘naming’ := ‘on’} switches the naming conventions on.
{attribute ‘naming’ := ‘omit} disables the naming conventions only for the next line.

5.3 Metrics

The standard metrics for all POUs, GVL and DUTs can be shown. Therefore, run Build → Static analysis → View Standard Metrics.



Program unit	Code size	Variables size	Stack size	Calls	Globals	IOs	NOS	Comments	McCabe	Prather	DIT	NOC	RFC	CBO	HL (Halstead)	HV (Halstead)
PLC_PRG (PRG)	160	45	0	1	0	0	8	36	4	1,50			1		76	178
fbStaticAnalysis.mFac	104		16	1	0	0	5	20	3	1,40			0		44	93
fbStaticAnalysis (FB)	112	16	8	1	0	0	3	36	1	1,00	0	0	2	0	23	49

A window is outputted where each POU is listed which is used in the program. POUs which are not opened like compiled libraries have only the Code size, Variable Size, Stack Size and Calls value.

By right clicking the table can be calculated again, copied printed or exported, configured or shown as Kiviat-Diagram. To display the Kiviat-Diagram at least three Metrics have to be configured with an upper and lower limit. It is also possible to open the POU from the Metrics view.

The Code size, variable size and stack size is given in bytes. Calls is the number how often this POU is called in the whole program.

Global is the number of used Global variables and IOs the number of direct addressed variables. It is also possible to show Local, Input and Output Variables of each POU.

NOS is Number of statements inside the POU.

Comments is giving the percentage of comments in the POU. As visible in the screenshot above comments for the method mFac is highlighted in red. In the Metric settings the minimum percentage of comments is defined as 25 %. As this is not reached cell is highlighted.

McCabe is the complexity. So the number of possible independent branches for program. It is the number of tests which are necessary that at least each line of code is executed once. The number increases by using IF and CASE statements.

Prather is the complexity of nesting. Which is influenced by the nesting weight and the number of statements.

DIT is the Deepth of Inheritance. The path length from the root to this object. NOC is the number of children of this class. RFC means Response For Class. This are the number of possible methods which can be executed.

CBO is the Coupling between objects. So the number of classes coupled with this class.

Halstead complexity measure has the outputs Length and Volume in the program. Also the Halstead Difficulty could be configured. The three KPIs are related to the number of unique operators and operands in contrast to the number of total operators and operands which can also be displayed if reqired.

The Halstead complexity calculation can be found in the internet. For example, on Wikipedia.[1]

[1] Source: https://en.wikipedia.org/wiki/Halstead_complexity_measures accessed 18.08.2020

6 Summary

The Pro Static Analysis tool can be configured individually depending on the project requirements. By using the Pro Static Analysis tool during the code development process issues can be found easily and the coding style can be improved.

The definition of the rules as well as the code changes and ignores during the development process need some time. This time can later be saved during commissioning and operation as the fault rate of the code can be reduced.

The standard metrics tool can give an overview of all POUs. With the usage of the standard metrics long or difficult functions can be detected. Metrics can be compared with an earlier state to see improvements or weaknesses

Furthermore, the Pro Static Analysis Tool can be used to show and compare the metrics of the project.

The ruleset to check the code has to be defined before the program is written. Checking the ready code before commissioning will lead to many issues as the code might not be written according the guidelines.

Using the Pro Static Analysis Tool can improve on the one hand the code readability and one the other hand the code robustness.

ABB Automation Products GmbH
Eppelheimer Straße 82
69123 Heidelberg, Germany
Phone: +49 62 21 701 1444
Fax: +49 62 21 701 1382
E-Mail: plc.support@de.abb.com
www.abb.com/plc