

ST tricks and tips

straton user guide – Rev.10

sales@straton-plc.com



straton



STRATON AUTOMATION, All Rights Reserved

The information contained in this document is the property of STRATON AUTOMATION. The distribution and/or reproduction of all or part of this document in any form whatsoever is authorized only with the written authorization of STRATON AUTOMATION. The technical data are used only for the description of the product and do not constitute a guarantee of quality in the legal sense of the term. We reserve the right to make technical changes.

Content

1. TRICKS AND TIPS.....	4
1.1. Recommended settings	4
1.2. Tool bar	4
1.3. Auto-declaration of missing symbols	4
1.4. Auto-declaration with prefix.....	5
1.5. Conditional compiling coloring.....	5
1.6. ST line numbers.....	5
1.7. Auto-completion of ST statements.....	5
1.8. Auto-indentation	6
1.9. Tool tips.....	6
1.10. Auto-completion of function calls	6
1.11. Auto-completion of a variable name.....	6
1.12. Selection of FB member.....	7
1.13. Selection of Structure or Bitfield member.....	7
1.14. Other syntax related commands.....	7
1.15. Track changes	7
1.16. ST Editor Outlining.....	7
1.17. Find command highlight all occurrences	8
1.18. Variable display	8
 2. CODE OPTIMISATION	 9
2.1. Major optimization.....	9
2.2. Recommendations	9
2.2.1. Repetition of expressions	9
2.2.2. Useless type conversions	10
2.2.3. Boolean calculation vs comparisons	10
2.2.4. Boolean expressions.....	10
2.2.5. Array indexes.....	10
2.2.6. The “FOR” statement	11

1. Tricks and tips

1.1. Recommended settings

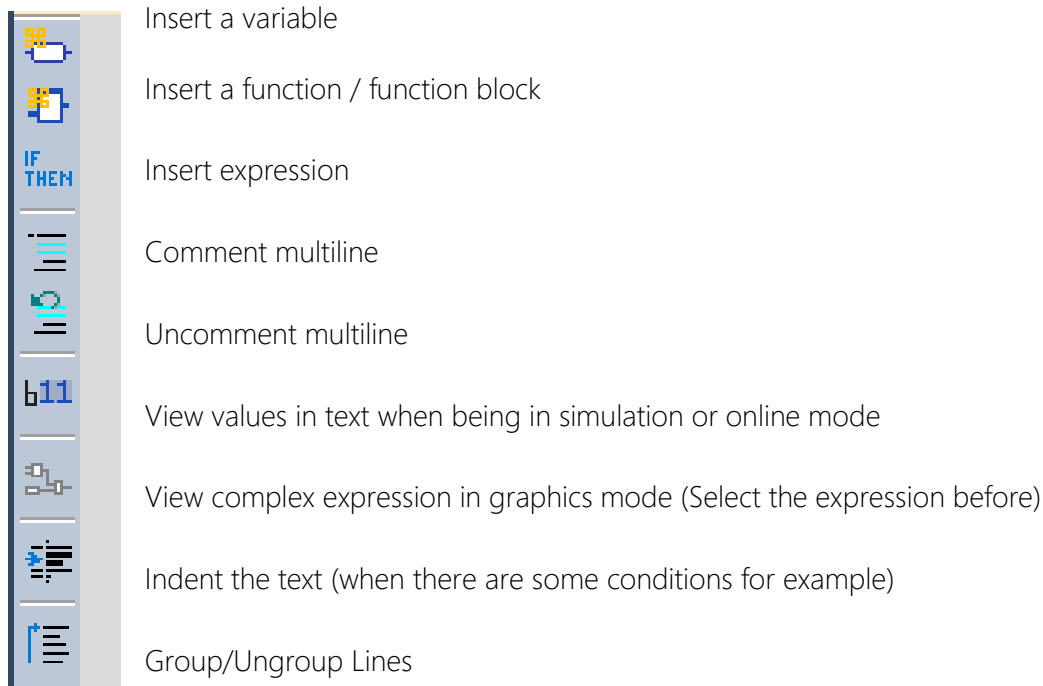
Some additional features are available for smart editing and are referred to as "Intellisense". Intellisense can be memory consuming and can be activated or deactivated from:

- Tools > Options > Editing > Activate intellisense

After activating or de-activating the Intellisense, you must close and reopen your project list.

The features described in this document are available when the Intellisense is activated.

1.2. Tool bar

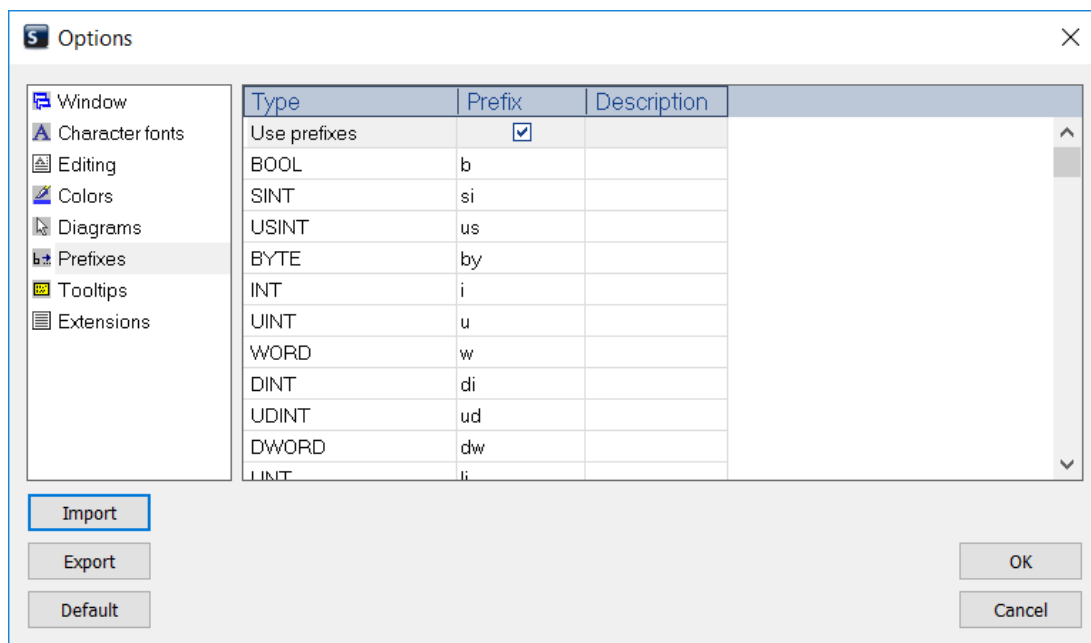


1.3. Auto-declaration of missing symbols

When pressing ENTER at the end of a line containing an unknown variable symbol, you will be prompted for declaring it immediately.

1.4. Auto-declaration with prefix

The option Prefixes from the menu Tools > Options > Prefixes, allows to declare variables with the corresponding type when enabled.



1.5. Conditional compiling coloring

Parts of code conditioned by `#ifdef` pragmas and which are not validated are grayed.

1.6. ST line numbers

ST line numbers can be displayed:

Tools > Options > Editing > Show ST line numbers

1.7. Auto-completion of ST statements

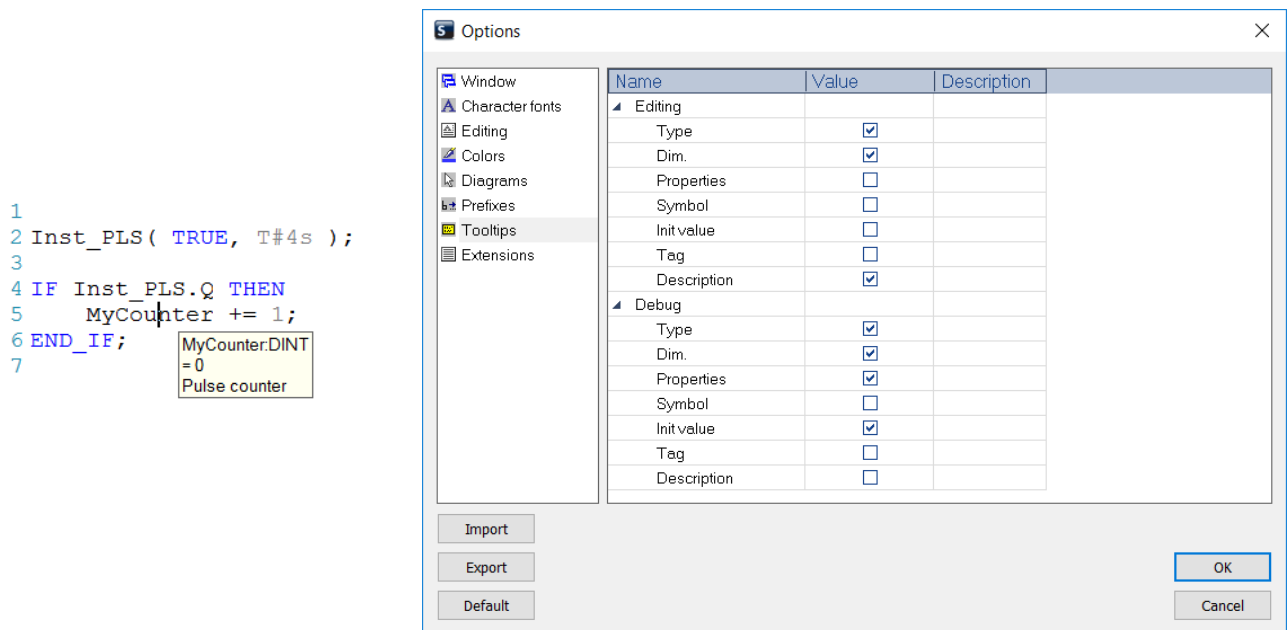
On an empty line, just enter the main keyword of a ST statement such as "FOR", "IF"... and immediately press the ENTER key. The whole statement will be completed including comments that will guide you through the syntax. The caret is automatically placed where you must enter the first required term or condition.

1.8. Auto-indentation

Lines are automatically formatted/indented on the left when entering structured ST statements.

1.9. Tool tips

When the mouse cursor is over a word, a tooltip is displayed. The contents of the tooltips can be set in the menu Tools > Options > Tooltips.



1.10. Auto-completion of function calls

Enter the name of a function simply followed by an opening parenthesis and immediately press the ENTER key. The call will be completed with the appropriate argument list including comments and possibly default values so that you are guided through the list of values to be passed to the called function.

It is also possible to drag and drop blocks in the ST window. The input parameters will be automatically displayed.

1.11. Auto-completion of a variable name

If entering the first letters of a variable name, you can hit the CTRL+SPACE key for automatically completing the name. A popup list is displayed with possible choices if several declared variable names match the typed characters.

1.12. Selection of FB member

When typing the name of a function block instance, pressing the point "." after the name of the instance opens a popup list with the names of possible output members.

1.13. Selection of Structure or Bitfield member

When typing the name of a variable having a structure or bitfield type, pressing the point "." after the name of the variable opens a popup list with the names of possible members.

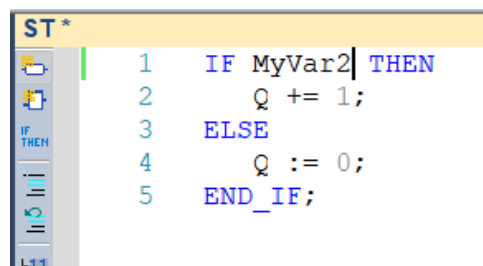
1.14. Other syntax related commands

When several lines are selected, you can automatically indent them. Press TAB or SHIFT+TAB keys to shift the lines to the right or to the left.

1.15. Track changes

The Track changes option allows the users to see all the changes that has been made in a ST program.

Changes are identified by a specific color on the left of the concerned line:




The Track changes option is activated by default, in order to activate/deactivate this option:

Tools > Options > Editing > Track changes

1.16. ST Editor Outlining

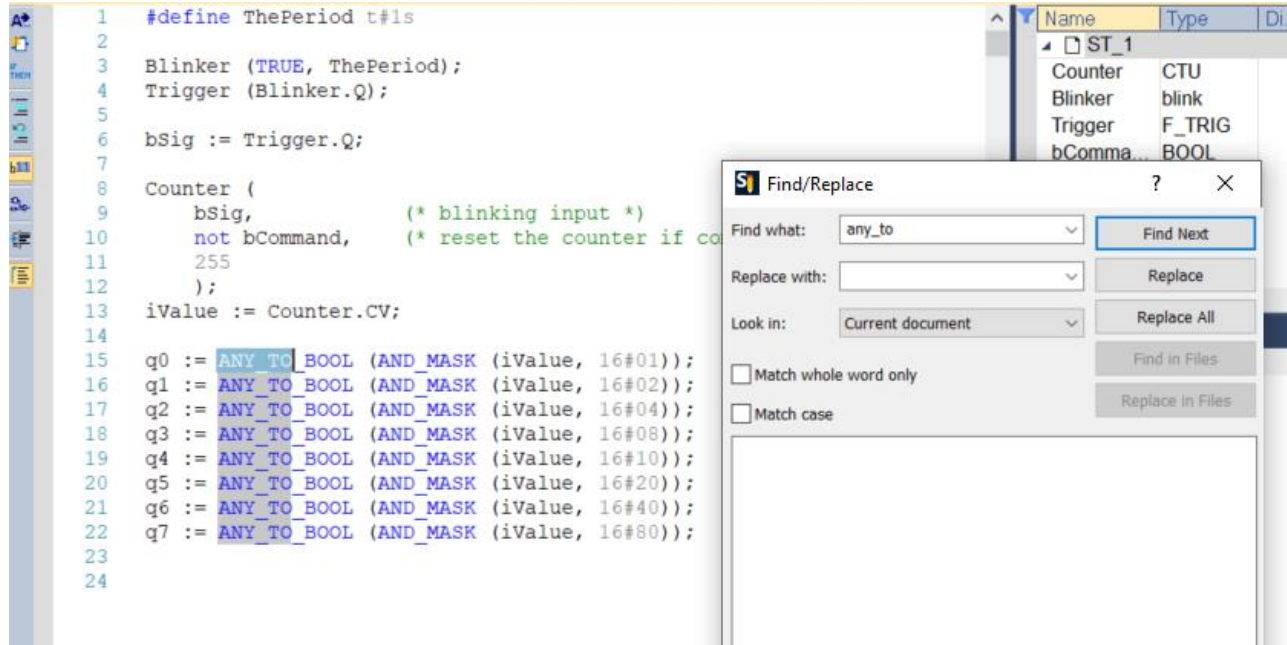
ST Editor Outlining is a way to group ST conditions (IF, CASE, WHILE...), using the button "—" to collapse the structure and button "+" to expand it.



This option can be set/unset directly from the Tool bar thanks to the  button.

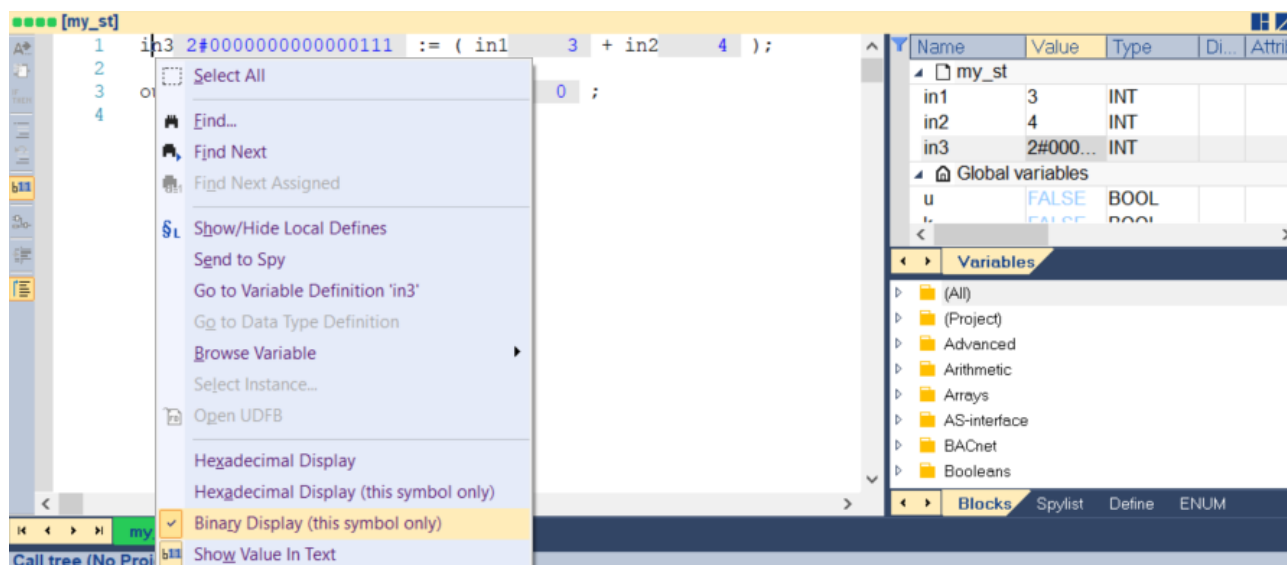
1.17. Find command highlight all occurrences

When using the Find/Replace command, all occurrences of the searched text are highlighted.



1.18. Variable display

The user can choose the binary display for a variable. When clicking on "Binary Display", the symbol is binary displayed in all debuggers (programs, variable list, spy lists...):



The user has then several options for the variable display:

- ▶ Binary display for one specific symbol
- ▶ Hexadecimal display for one specific symbol
- ▶ Hexadecimal display for all symbols
- ▶ Hexadecimal display for all symbols + binary display for specific symbols
- ▶ Normal displaying (depending on the symbol type)

2. Code optimisation

This part is a guide to write or simplify ST programs in the most optimized way. Generally speaking, the ST compiler integrates few optimization features so the way the ST code is written may have an impact on the runtime performances.

2.1. Major optimization

Before anything, you should consider the project options and their impact on the execution speed:

- ▶ Compiling the project in DEBUG mode is very time consuming. Optimized applications should be compiled in RELEASE mode.
- ▶ The option "Store complex variables in a separate segment" is time consuming and should be avoided if possible.
- ▶ The support of "Locked" variables is time consuming and should be disabled when the application is compiled in RELEASE mode.

2.2. Recommendations

2.2.1. Repetition of expressions

If a complex expression is used several times in a program, it is better to evaluate it only once and put the result in a temporary variable.

Not optimized:

```
r1 := any_to_real (di1 / 100) + r2 + 1.0;
r2 := any_to_real (di1 / 100) + r2 + 1.0;
r3 := any_to_real (di1 / 100) + r2 + 1.0;
```

Optimized:

```
temp := any_to_real (di1 / 100) + r2 + 1.0;
r1 := temp;
r2 := temp;
r3 := temp;
```

2.2.2. Useless type conversions

Instead of using ANY_TO_... functions to cast a constant expression, directly specify the constant with the appropriate type:

Not optimized:

```
lr1 := any_to_lreal (1000);
```

Optimized:

```
lr1 := LREAL#1000.0;
```

2.2.3. Boolean calculation vs comparisons

For boolean tests, it is better to use Boolean operations rather than IF/THEN/ELSE structure.

Not optimized:

```
if B1 = true then
  B2 := true;
else
  B2 := false;
end_if;
```

Optimized:

```
B2 := B1;
```

2.2.4. Boolean expressions

straton proposes the following compiling option regarding boolean expressions:

This option should be checked in case of complex Boolean expressions such as:

```
B := B1 and B2 and not (B3 or B4);
```

If the option is set, then the evaluation is performed from the left to the right and stops as soon as the result is known. E.g. in this case only B1 is evaluated if it is FALSE.

Warning: this may lead to some side effects if function or sub-program calls are in the expression, such as:

```
B := B1 and MyFunction ();
```

In this case, if the option is set and B1 is FALSE, then MyFunction() is NOT called.

2.2.5. Array indexes

All arrays in straton begin with an index of 0. If you are used to work from 1 to N, then declare an array of one more element and do not use the [0] item. Avoid offsetting the index in your code.

Not optimized:

```
// ARR is an array of 10 (from 0 to 9)
for I := 1 to 10 do
```

```
ARR[i-1] := ...  
end_for;
```

Optimized:

```
// ARR is an array of 11 (from 0 to 10)  
for I := 1 to 10 do  
  ARR[i] := ...  
end_for;
```

2.2.6. The “FOR” statement

The “FOR” statement performs a lot of checks about the index at runtime and thus is less performant than other loop statements such as WHILE or REPEAT.

Not optimized:

```
for index := 1 to 10 do  
  ...  
end_for;
```

Optimized:

```
Index := 1;  
while index <= 10 do  
  ...  
  Index := index + 1;  
end_while;
```