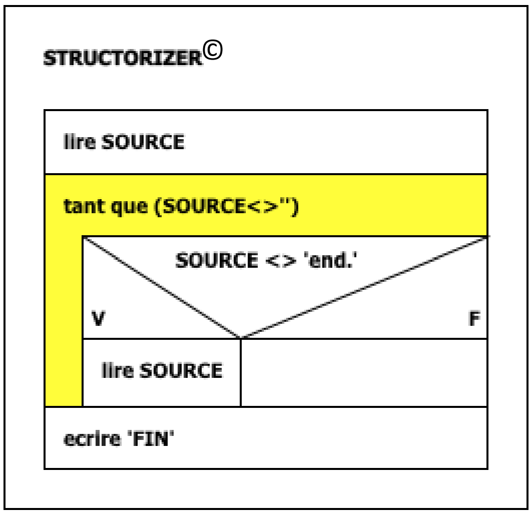


Structorizer© is a free open-source editor for Nassi-Shneiderman diagrams, written by Robert Fisch (https://structorizer.fisch.lu)

Quick Start Guide[1]

Author:	Praveen Kumar
Revised:	Kay Gürtzig
Edition:	2020-03-29

Contents

Introduction 2
Quick Start 3
Heading 3
Comments 4
Element Selection 4
Element Insertion 4
Moving Elements 5
Preferences 5
- Structure Preferences 5
- Parser Preferences 6

Diagram Size 6
Colouring 7
Algorithm Export 7
- Picture export 7
- Code export 7

Code Import 8
Advanced Features (Examples) 8

[1] Please consult the User Guide (https://help.structorizer.fisch.lu) for a detailed description of the product.

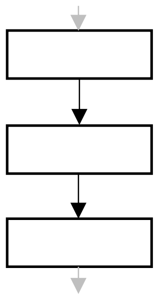
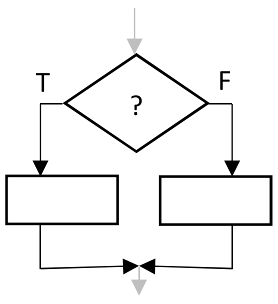
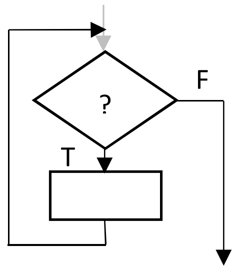
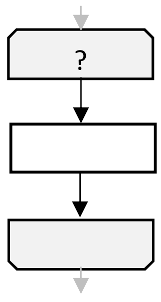
Introduction

Many programmers first design their algorithms as a graphical flow chart, which helps visualizing the code flow, before actually starting to write the code, particularly for new and complex tasks.

All algorithms can be composed using the following basic constructs:

1. Operations (Activities)
2. Decisions
3. Iterations (Loops)

The traditional “unstructured” flow charts (in Germany named “Programmablaufplan” or “PAP”, as standardized by DIN 66001) form a graph of nodes and directed links (arrows). The above mentioned basic components are expressed as follows:

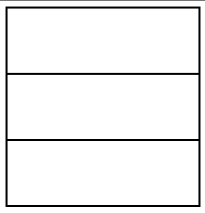
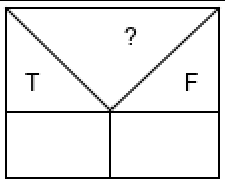
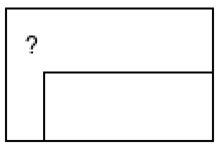
Operations	Decision	Loop (example < 1982)	Loop (example ≥ 1982)
			

The major disadvantages of these flowcharts are:

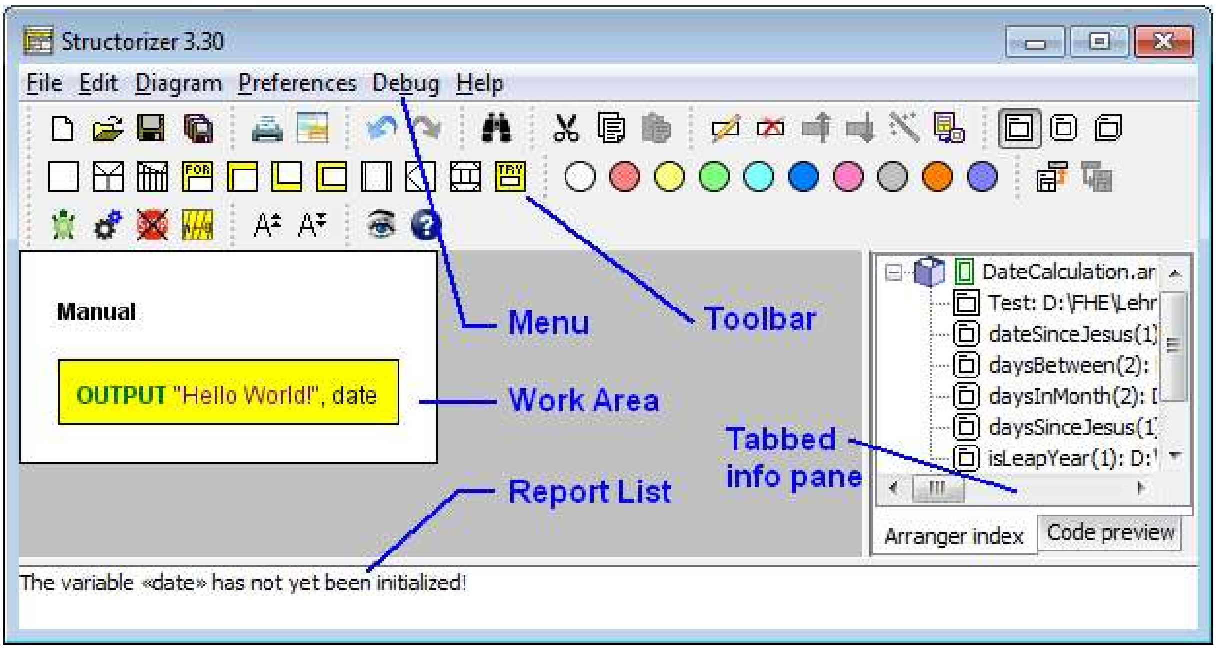
1. The arrows take a lot of space
2. Some of the basic algorithm structures are composed of several nodes and links that may be distorted and displaced in many ways, making a recognition of structures difficult.
3. It is very difficult to convert flowcharts of this kind into programming code.
4. Structural consistency is hard to check.
5. Nested loops with the symbolism of DIN 66002/1982 give poor overview as their corresponding pairs of start and end nodes must be matched by counting along the apparently linear flow.

Structured flowcharts as introduced by Isaac NASSI and Ben SHNEIDERMAN (so called structograms, standardized in Germany with DIN 66261) in contrast are very compact, unambiguous and easy to convert into code. Their characteristics are:

- The control flow passes through the elements from top to bottom (no need for arrows).
- The branches of a decision re-join necessarily at its end.
- Every loop has a single exit point.

Operations	Decision	Iteration (Loop)
		

Quick Start

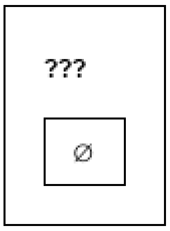


The variable «date» has not yet been initialized!

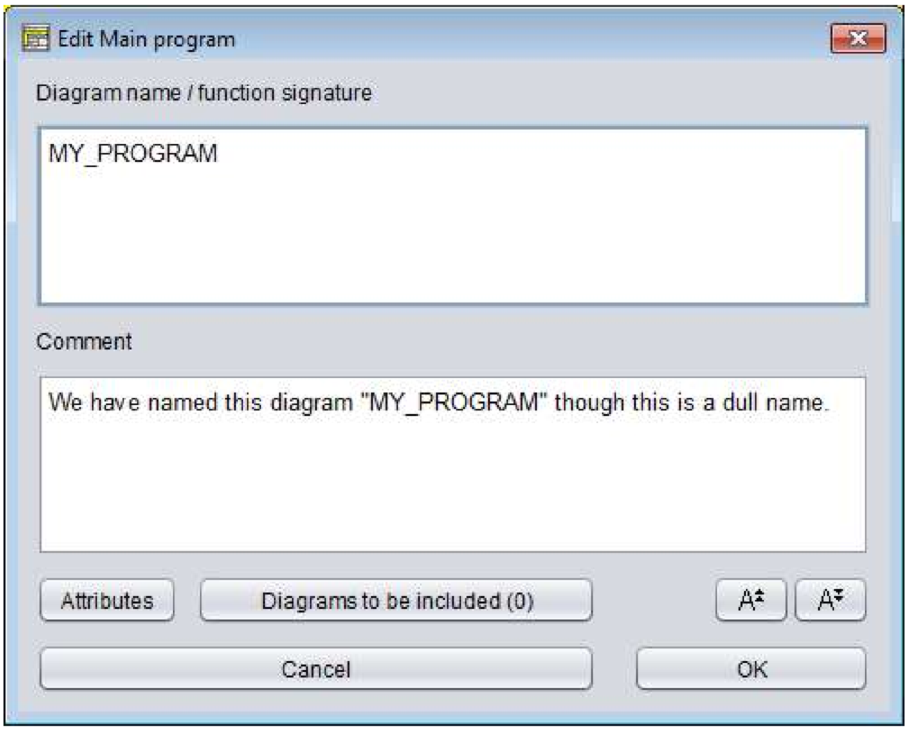
The dialog language can be chosen among a broad set of idioms:

Heading

When you start Structorizer, it shows an empty diagram (see right-hand side).



Double-click its outer rectangle (containing the "???" label) in order to give your diagram a descriptive name. This will open the diagram editor:


Edit Main program
Diagram name / function signature
Comment
Attributes
Diagrams to be included (0)
Cancel
OK

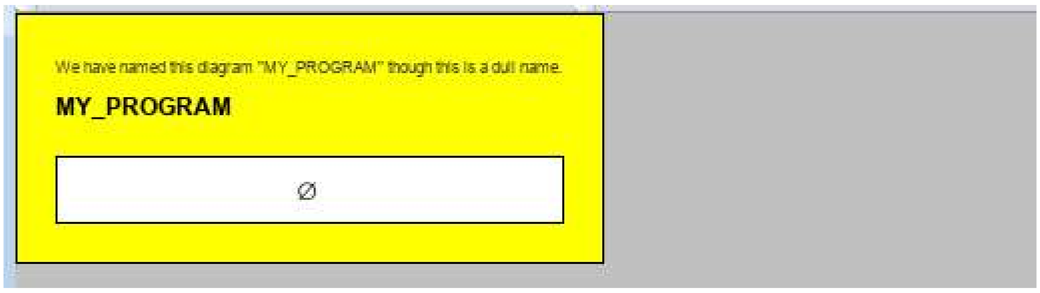
MY_PROGRAM

We have named this diagram "MY_PROGRAM" though this is a dull name.

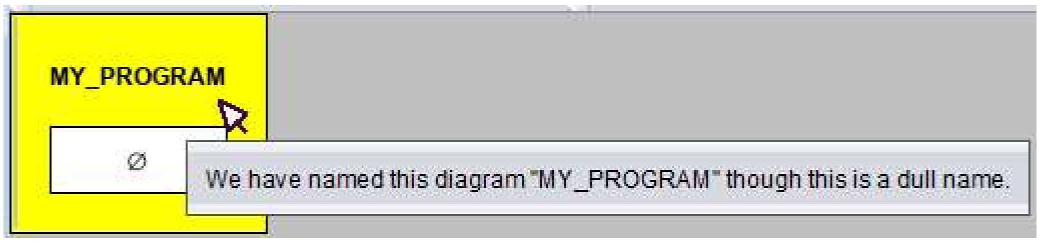
Comments

The comment for the diagram (or its elements) can be shown in different ways:

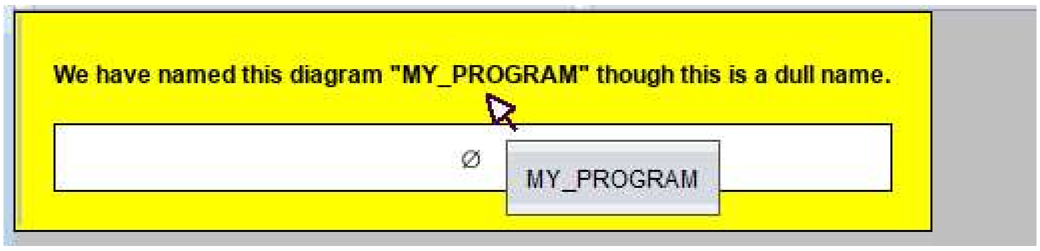
1. Together with the text but in smaller font (mode “Text + Comments”, see menu “Diagram”)


We have named this diagram "MY_PROGRAM" though this is a dull name.
MY_PROGRAM
∅

2. Its existence marked by a grey bar at the left edge and popping up while the Mouse hovers over the element (mode “Show comments?”, see menu “Diagram”):

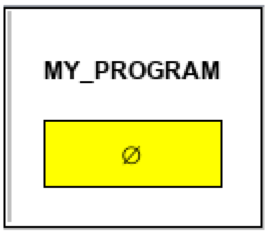

MY_PROGRAM
∅
We have named this diagram "MY_PROGRAM" though this is a dull name.

3. Shown *instead* of the element text (conversely, the text may be popped up on demand) with mode “Switch texts/comments?”, see menu “Diagram”:


We have named this diagram "MY_PROGRAM" though this is a dull name.
∅
MY_PROGRAM

4. Completely hidden (none of the modes mentioned above).

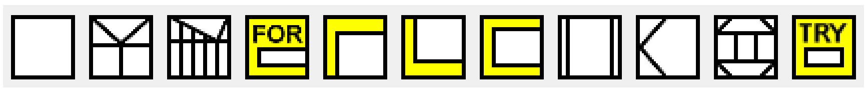
Element Selection



Before inserting elements, we first select the place where the insertion is intended. This is done by clicking on the closest element. At the beginning there is only the empty box with the symbol ∅ in it. Select it by clicking once on it.
It will get highlighted (in bright yellow, see screenshot to the right):

Element Insertion

Ten standard element types (and a non-standard extra type) are now available for insertion:



1	2	3	4	5	6	7	8	9	10	11
F5	F6	F10	F7	F8	F9	Ctrl F7	F11	F12	Ctrl F6	Ctrl F5

The symbols show the approximate shape of the respective element types. Their meaning is:

1. Instruction (including Input and Output)

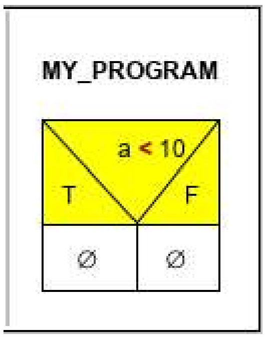
2. IF THEN ELSE Statement (Alternative) 3. CASE Statement (Case Selection)	Decision (Branching)
4. FOR Loop (Count-controlled or Collection-controlled) 5. WHILE Loop (Condition-controlled at start, entry-controlled) 6. REPEAT UNTIL Loop (Condition-controlled at end, exit-controlled) 7. Endless Loop (neither entry nor exit control)	Iteration (Loops)

8. CALL a subroutine (function or procedure, i.e. another diagram)
9. EXIT Statement (break/leave, return, exit)
10. PARALLEL section
11. TRY CATCH FINALLY block (exception control, non-standard)

Clicking on one of the symbols (or pressing the respective function key combination shown in the table above) will insert an element of the chosen type **after** the selected element. If you want it to be inserted **before** the selected element then hold the Shift key pressed while you click on the toolbar button or press the function key combination.

When the element editor opens (it looks similar to that shown in section "Heading"), fill in the requested text content and possibly some helpful comment.

After having selecting e.g. symbol 2 and filled in a condition as requested, say **`a < 10`**, the diagram might look like this:



Moving Elements

To move a misplaced element to another position you may click on the element and drag it to the target position. A valid move is highlighted in green, whereas an invalid position is highlighted in red. Note that the dragged element will always plop in *after* the highlighted target position.

Another way to move an element to neighbouring positions is to use the cursor keys while you keep the "Ctrl" key pressed.

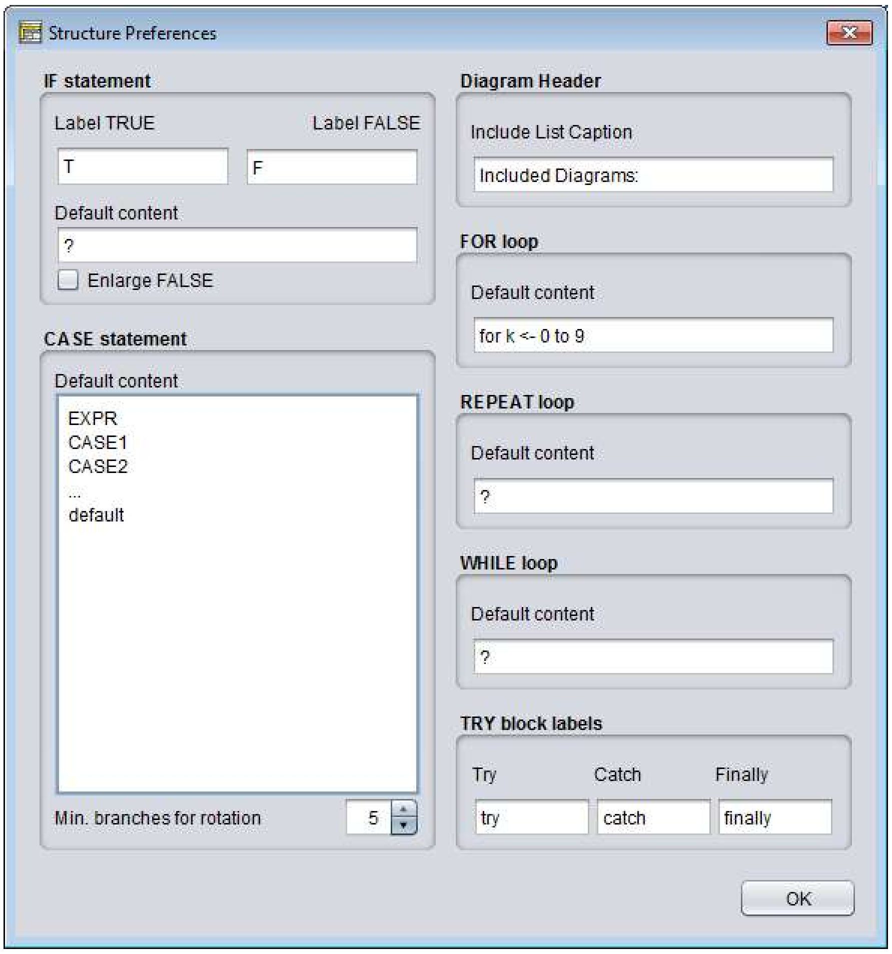
Or you may cut and paste elements or contiguous element sequences via the usual key combinations.

Preferences

Structure Preferences

The labels for the branches of an Alternative (IF THEN ELSE statement, see previous screenshot) may be configured as required via menu item "Preferences"→"Structures…". You may also specify the editor default texts for the respective kinds of element to be filled in when you insert a new element.

If you choose to place some "decorative" sort of keyword in the default text, e.g. "`if( )`" for an IF THEN ELSE statement (briefly IF statement), then it might be a good idea to specify this decorative keyword as to be ignored by Structorizer for the respective element type in the Parser Preferences (see further below).


Structure Preferences

IF statement

Label TRUE	T
Label FALSE	F
Default content	?

☐ Enlarge FALSE

CASE statement

Default content:

```
EXPR
CASE1
CASE2
...
default
```

Min. branches for rotation: 5

Diagram Header

Include List Caption: Included Diagrams:

FOR loop

Default content: for k <- 0 to 9

REPEAT loop

Default content: ?

WHILE loop

Default content: ?

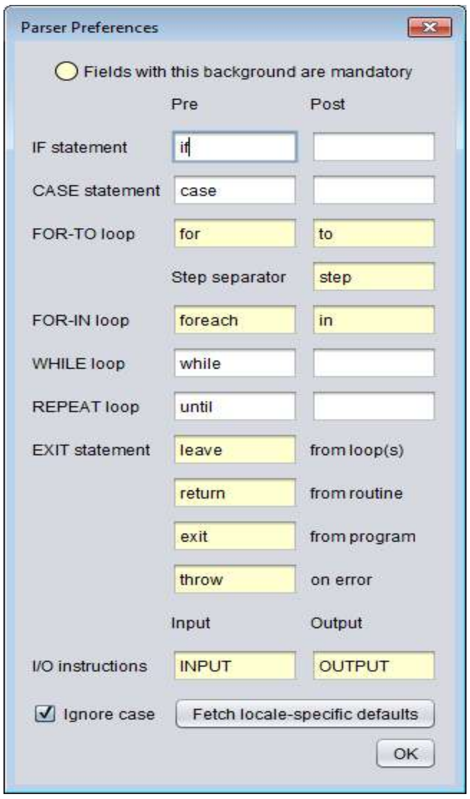
TRY block labels

Try	Catch	Finally
try	catch	finally

Parser Preferences

There are no particular element types for input and output. Instead you are to use ordinary Instruction elements. If Structorizer shall identify (and interpret) input and output instructions, however, then you ought to specify suited key-words according to your personal preference. The same holds for EXIT elements (which may have different meaning) and with respect to a correct interpretation of FOR loops.

This configuration can be done via menu item “Preferences”→”Parser…” (see following screenshot). The keywords necessary for correct interpretation of elements on execution in the debugger and on code export and code preview have a yellowish field background. The remaining fields are optional and simply allow to specify, which “decorative” keywords (as mentioned in section “Structure Preferences” above) are to be tolerated and ignored on execution and code export.


Parser Preferences

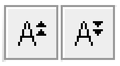
◯ Fields with this background are mandatory

	Pre	Post
IF statement	if	
CASE statement	case	
FOR-TO loop	for	to
	Step separator	step
FOR-IN loop	foreach	in
WHILE loop	while	
REPEAT loop	until	
EXIT statement	leave	from loop(s)
	return	from routine
	exit	from program
	throw	on error
	Input	**Output**
I/O instructions	INPUT	OUTPUT

☑ Ignore case

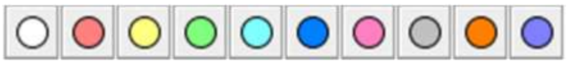
Diagram Size

The size of the flowchart can be changed via the font size. You may enlarge or diminish the diagram font by clicking on the following toolbar buttons:



Colouring

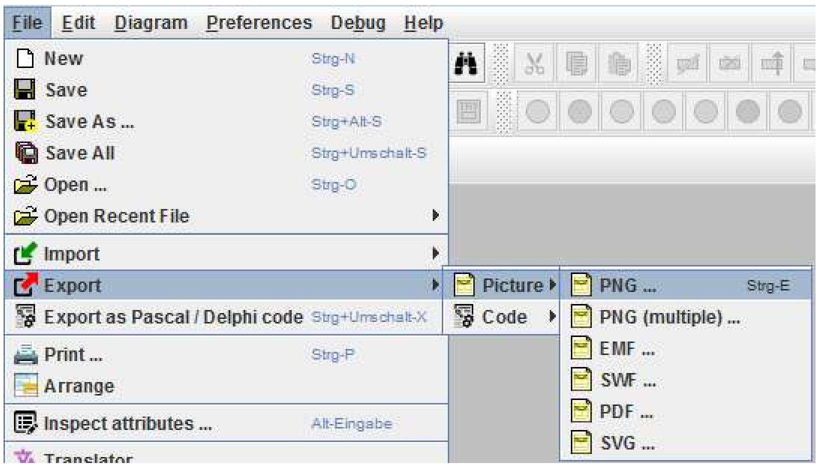
The element background (while not selected) is by default white. In order to emphasize or distinguish certain parts of the algorithm you can change the filling colour of the selected elements by clicking on the appropriate paint box button in the toolbar:



Algorithm Export

Picture export

The diagram may be exported as picture to the following formats:

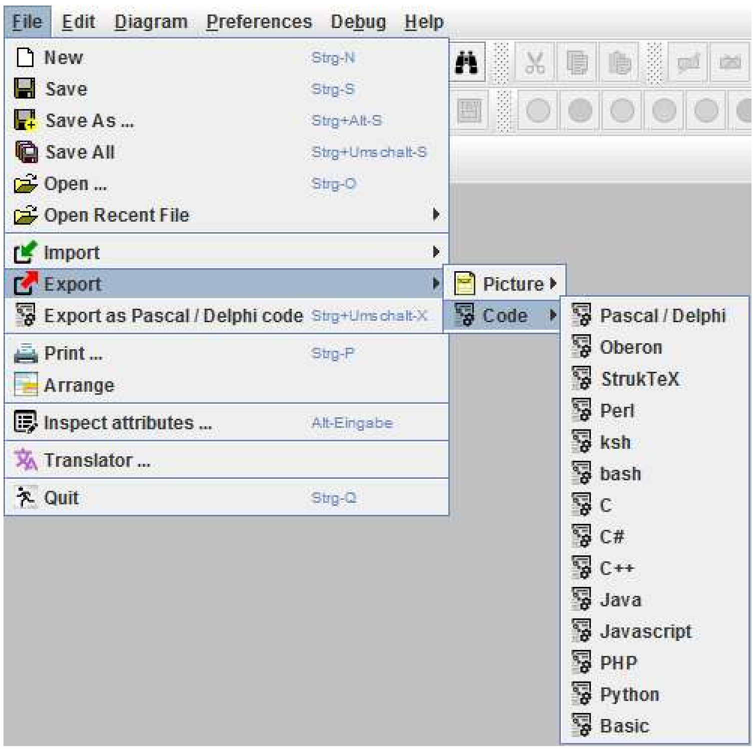


You may also copy the current diagram as PNG or JPG picture (system-dependent) to the clipboard by pressing Ctrl-D.

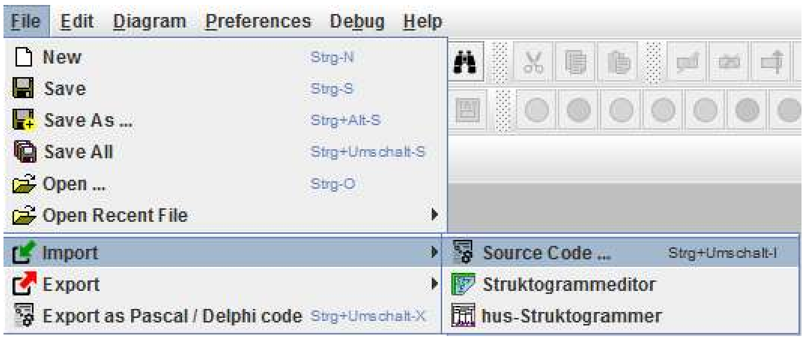
Code export

Already while you work on a diagram, you will see a code preview in the right part of the window. You may easily change the target language via the context menu of the code preview pane.

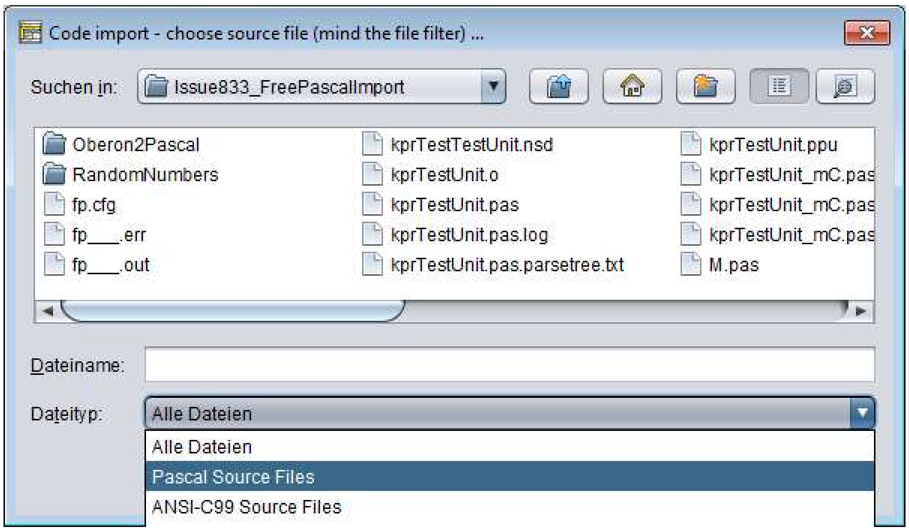
Via the menu you have the opportunity to export your algorithm to a source file:



Code Import



For certain programming languages (by now Pascal, C, and COBOL) code can directly be imported to obtain structured flowcharts (i.e. Nassi-Shneiderman diagrams). Structorizer will decide automatically what Parser to use by the file name extension. In case of doubt Structorizer will ask you. Alternatively, you may choose among the file filters in the file chooser dialog:



Advanced Features (Examples)

A built-in debugger allows you to test your algorithm by interpreting the diagram directly. You can do a step test and may place breakpoints. Current variable contents are shown. You may perform a qualitative runtime analysis having Structorizer count the execution passages of each element and display the diagram in an execution-count-related colour spectrum.

"Turtleizer": Structorizer provides some built-in turtle drawing commands useful for teaching and learning how to program. A special debug environment allows to interpret them and draw the resulting turtle images.

Structorizer facilitates the decomposition of complex algorithms into several subroutines by offering to extract a selected instruction sequence into a subroutine diagram. Or it may help creating a subroutine diagram for an inserted CALL element without corresponding target.

"Arranger": Rather than holding the opened diagrams in a mere tab list or the like, Structorizer allows you to place ("arrange") them together on a drawing canvas according to your wishes. This is where e.g. the subroutine diagrams for a main diagram you work on may reside and wait for being called. You can organize the arranged diagrams in **groups** (similar to "projects" in many IDEs).

Find & Replace: A dedicated search tool allows you to do a selective search for words or substrings, possibly via regular expressions, within the current diagram or all opened diagrams, restricting the search to certain subsets of element types, to the elements texts or comments and to replace them if needed. This may be of grand help on refactoring the diagrams.