



BPMN-RPA Studio

User guide
2021

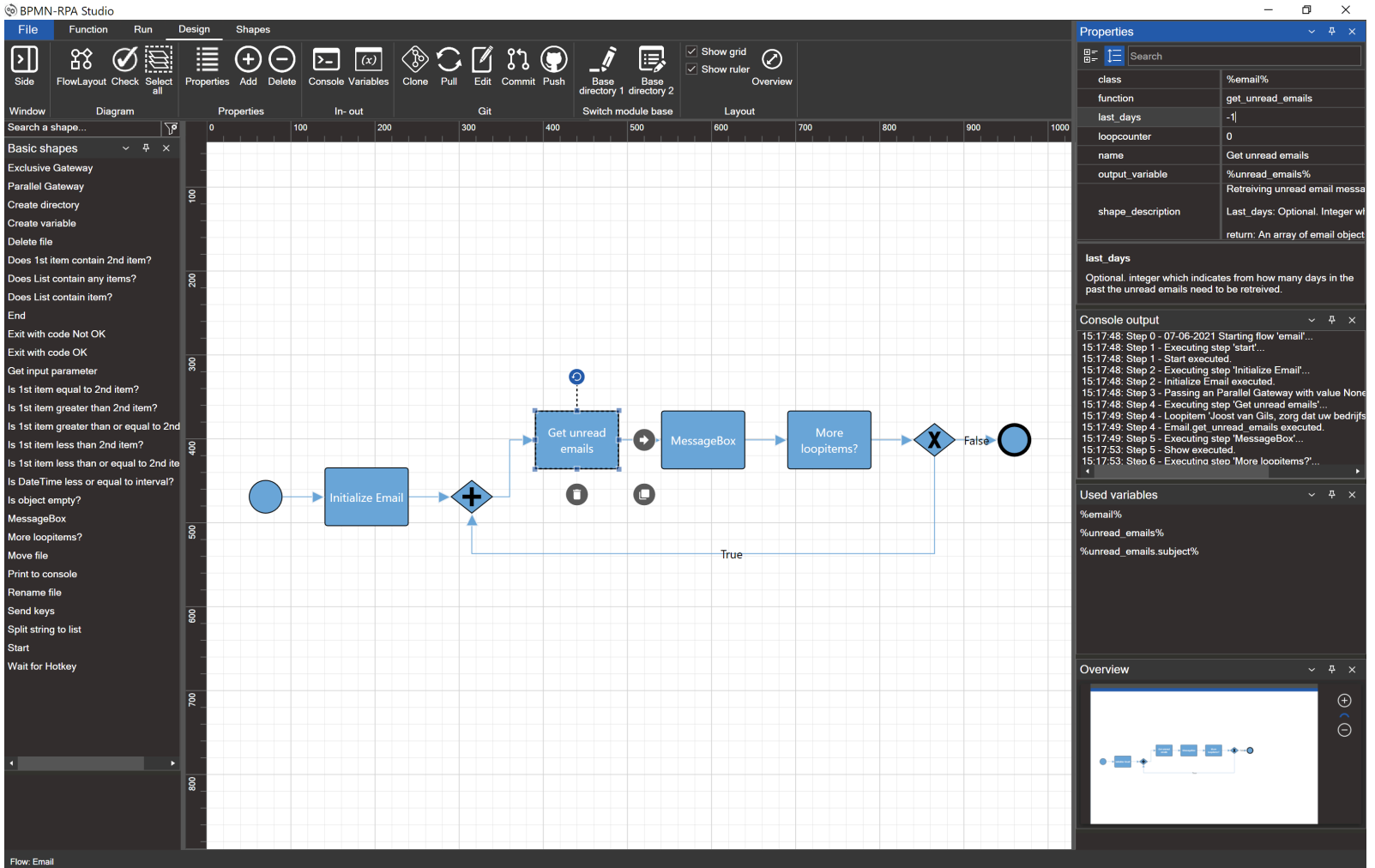
Table of contents

1. Introduction	3
2. Installation.....	4
3. Shapes.....	5
3.1. Shape libraries	7
3.2. Available libraries	11
3.3. The ShapeType parameter	13
4. Variables	16
4.1. Retrieving information from variables	17
5. Loops	18
6. Blocks	19
7. Building a Flow.....	21
8. Running your Flows	24
9. Side window	25
9.1. Properties window	27
9.1.1. Shape description.....	28
9.2. Console window	30
9.3. Variables window.....	31
9.4. Overview window	32
9.5. Debug window.....	33
10. Menu tabs	34
10.1. File menu	35
10.1.1. Options window.....	37
10.1.2. Opening an existing Flow	39
10.1.3. Script generation	41
10.2. Run tab.....	42
10.2.1. Passing input to a flow	43
10.2.2. Debug a Flow.....	45
10.3. Design tab	48
10.3.1. Creating a flow	50
10.3.2. Modules base directory	52
10.3.3. Checking a Flow	54
10.3.4. Git integration	56
10.4. Shapes tab	59
10.4.1. Creating custom Shapes.....	61
10.5. Function tab	64
11. Command line options	65
12. Keyboard Shortcuts.....	66

1. Introduction

What is BPMN-RPA Studio?

BPMN-RPA Studio is a tool for Visual Programming of Python code Flows. It is a low-code solution where you can create Flows that embeds the functionality of your Python code. This means you dont have to write Python code scripts any longer to provide functionality to users!



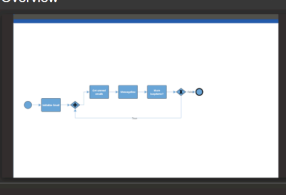
The screenshot displays the BPMN-RPA Studio interface. The main workspace shows a flow diagram with the following steps: a start node, an 'Initialize Email' task, an exclusive gateway, a 'Get unread emails' task, a 'MessageBox' task, a 'More loopitems?' task, and an exclusive gateway. The flow ends at a final node. The 'Get unread emails' task is currently selected, and its properties are shown in the right-hand pane. The console output pane shows the execution log, and the used variables pane lists the variables used in the flow.

class	%email%
function	get_unread_emails
last_days	-1
loopcounter	0
name	Get unread emails
output_variable	%unread_emails%
shape_description	Retrieving unread email messa return: An array of email object

last_days
Optional. integer which indicates from how many days in the past the unread emails need to be retrieved.

Console output
15:17:48: Step 0 - 07-06-2021 Starting flow 'email'...
15:17:48: Step 1 - Executing step 'start'...
15:17:48: Step 1 - Start executed.
15:17:48: Step 2 - Executing step 'Initialize Email'...
15:17:48: Step 2 - Initialize Email executed.
15:17:48: Step 3 - Passing an Parallel Gateway with value None
15:17:48: Step 4 - Executing step 'Get unread emails'...
15:17:49: Step 4 - Loopitem 'Joost van Gils, zorg dat uw bedrijfs
15:17:49: Step 4 - Email.get_unread_emails executed.
15:17:49: Step 5 - Executing step 'MessageBox'...
15:17:53: Step 5 - Show executed.
15:17:53: Step 6 - Executing step 'More loopitems?'...

Used variables
%email%
%unread_emails%
%unread_emails.subject%

Overview


2. Installation

Prerequisites

BPMN-RPA is a Windows program. To install BPMN-RPA Studio you need to have Python installed. BPMN-RPA Studio works best with Python3x. Also the Python [BPMN-RPA Library](#) must be installed.

Installation

You can download the setup installation program from <https://1ic.nl/download>. When you run the setup program, you will see the Windows Defender Smartscreen warning message "Windows protected your PC" popping up. Click on the "More info" link, so the "Run anyway" button will be visible. Then continue the installation process.

Location

BPMN-RPA must be installed on a location where you will have full read and write access. Templates and flows will be saved in subdirectories of the installation location. Therefore it is advised **not** to install the program in the "c:\program files" folder.

3. Shapes

Shapes

A BPMN-RPA flow consists of Shapes (or "steps") that are connected to each other by a connector. Connectors are shown as an arrow between the connected Shapes. The basic rule is that each Shape of the BPMN-RPA flow only has only one incoming and/or outgoing connector, except for Gateway shapes.

Shape types

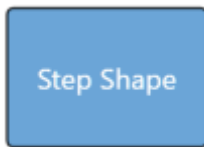
BPMN-RPA Studio uses 5 types of Shapes:

1. Start Shape:



This Shape marks the start of the Flow.

2. Step Shape:



This Shape contains a reference to Python code (module, class or function). This Shape can be seen as a "step" in the flow that will execute code.

3. Exclusive Gateway:



This Shape marks a decision point in the flow. The Shape itself has no editable attributes. An exclusive Gateway always has two outgoing connectors with the value (text) "True" or "False". The decision itself (the outcome of "True" or "False") isn't made in the Exclusive Gateway itself, but should be made in the predecesing Shape. Therefore an Exclusive Gateway will only route the flow to the next Shape to execute.

4. Parallel Gateway:



This shape combines multiple connectors into one outgoing connector. For now the Parallel Gateway can only have one outgoing connector, meaning only sequential flows can be created.

5. End Shape



This Shape marks the end of the Flow.

Shape properties

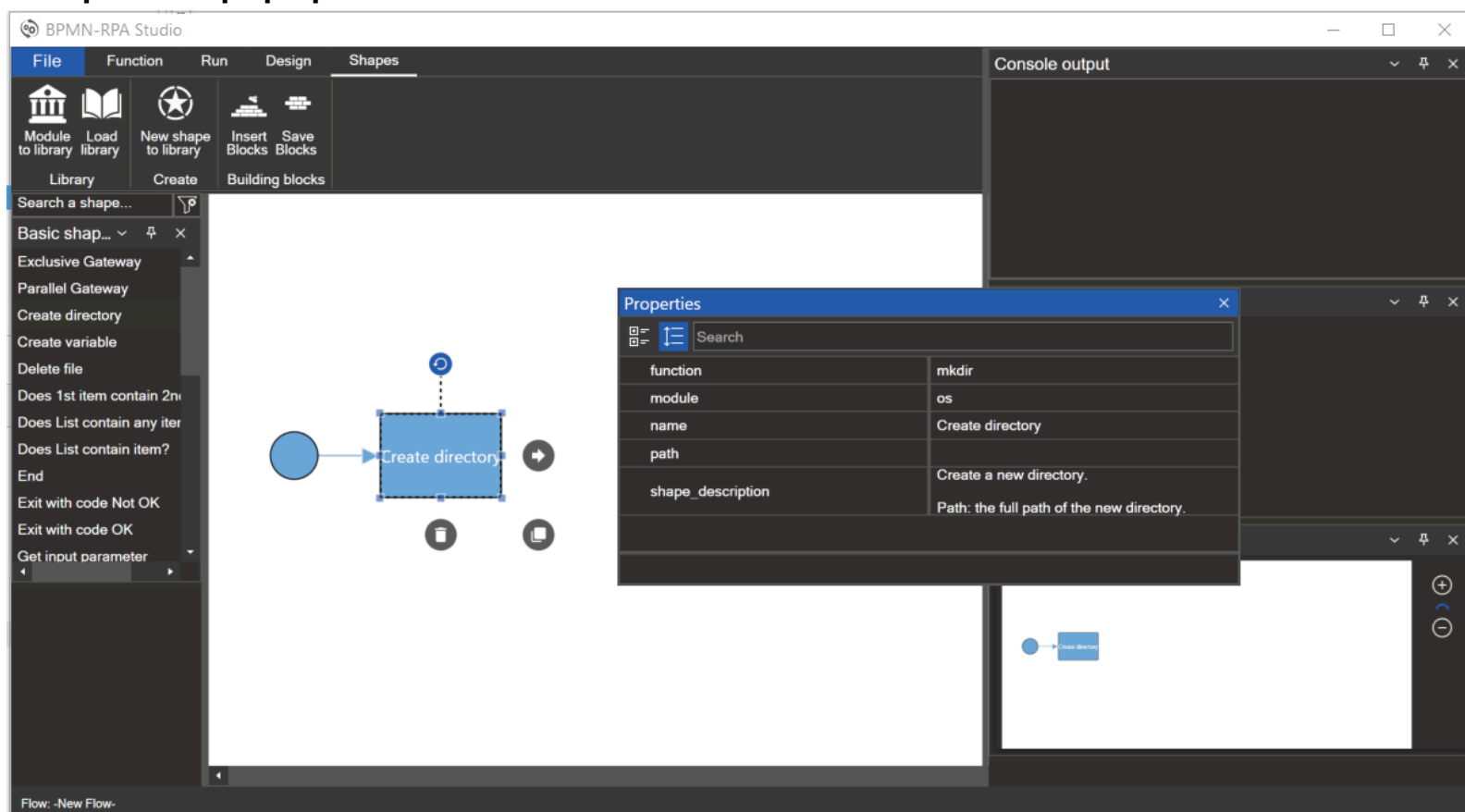
For the WorkflowEngine to recognize the Shapes, each Shape has to contain certain properties. A Shape can have multiple properties. The number of properties depends on the Python module and function the Shape refers to. The main properties of a Shape are:

- *Name*: The name of the Shape. The name of the Shape will also be shown as text in the Shape.
- *Module*: This is the full path to the Python file that contains your Class and/or function.
 - *From file*: specify the full path (including extension .py) if you want to load you module from a specific file location.
 - *From file in the BPMN_RPA/Script directory*: specify only the module name (including extension .py) of the module you want to use.
 - *From installed package*: specify only the module name (without extension .py).
 - *No Module field*: you can delete the Module field to call a function in the WorkflowEngine class directly.

BPMN-RPA Studio

- **Class:** for reference to the Class to use in the Module. You can delete this field if the called module has only functions and no class.
- **Function:** The name of the Function to Call. This field is mandatory.
- **Output_variable:** The name of the variable that must store the output of the current action, so it can be used in the following steps of the flow. If you don't use this field (or delete it), the current Shape will have no output that can be used by other Shapes in the flow.
- **Shape_description:** A description of the Python module, function or class called, including its parameters.
- **<Optional attributes>:** You can specify any input value for the called function directly by adding an extra attribute to the shape with exactly the same name as the expected input parameter(s) of the function.

Example of Shape properties



In the example above a Shape is added to the flow to create a directory. The function "mkdir" is part of the "os"-module that is already installed in one of the subdirectories of your Python installation. According to the "module" attribute we are referring to a function and module *From installed package*, so we specify only the module name (without extension .py). According to the documentation we MUST set the parameter "path", so this attribute is named exactly the same as the os.mkdir function expects:

Syntax: `os.mkdir(path, mode = 0o777, *, dir_fd = None)`

Parameter:

path: A path-like object representing a file system path. A path-like object is either a string or bytes object representing a path.

mode (optional) : A Integer value representing mode of the directory to be created. If this parameter is omitted then default value 0o777 is used.

dir_fd (optional) : A file descriptor referring to a directory. The default value of this parameter is None. If the specified path is absolute then dir_fd is ignored.

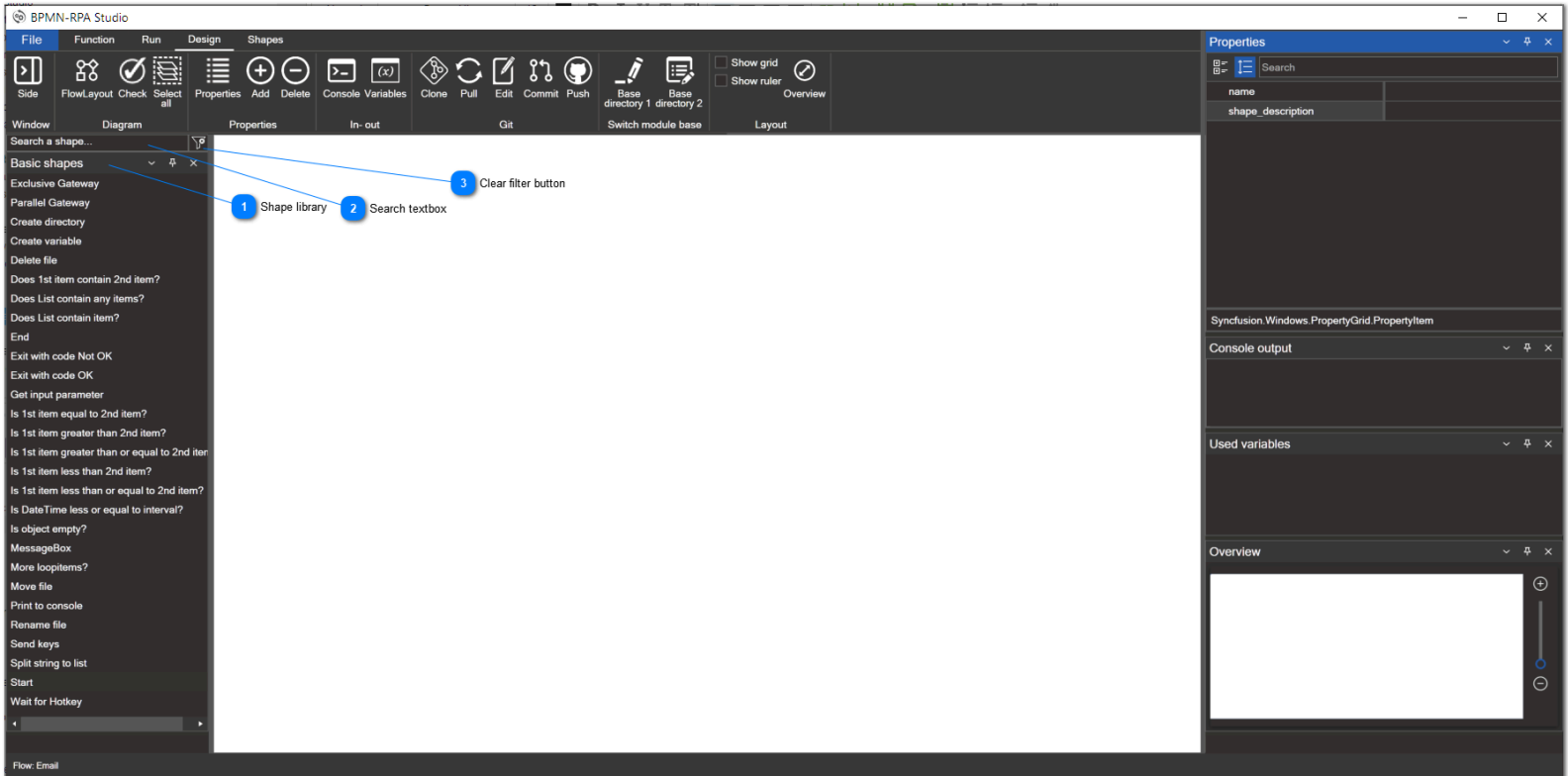
Note: The '*' in parameter list indicates that all following parameters (Here in our case 'dir_fd') are keyword-only parameters and they can be provided using their name, not as positional parameter.

Return Type: This method does not return any value.

We neither need the parameters "mode" nor the parameter "dir_fd", so we do not specify these in the properties (but we could have!).

3.1. Shape libraries

In BPMN-RPA Studio, you create a flow by adding Shapes and connecting them with connectors. Shapes are grouped into Shape libraries. BPMN-RPA Studio comes with several Shape libraries out-of-the-box (see: [Available libraries](#)). By default, the Basic Shapes library is opened when you start the program and will be visible on the side menu on the left:



1 Shape library

Location of the loaded Shape library (in this example the "Basic Shapes" library)

2 Search textbox

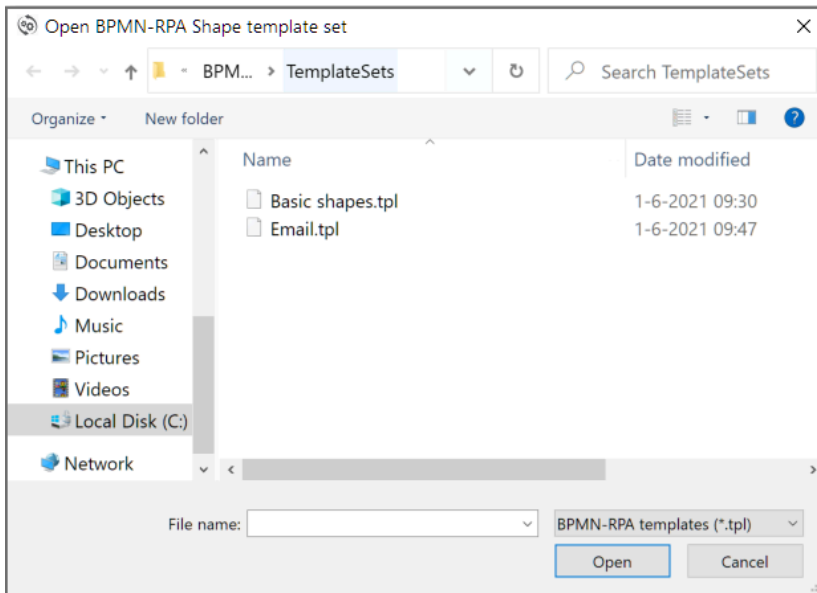
Search textbox for searching in the opened shape libraries. The Shapes are filtered while typing. All Shapes, no matter in what opened library, will be filtered on (having a part of) their name that matches the typed text.

3 Clear filter button

This button clears the filter that was set. By clearing the filter all of the Shapes in all the loaded libraries will become visible again.

Loading a Shape Library

Shape libraries are stored at the location that is set in the "Templates folder" field of the [Options window](#). You can load a library by clicking the "" button of the "" tab. The Open File dialog will pop up, showing only the template files (.tpl) of the saved Shape libraries.



Select a file and click on "Open" to open the Shape Library. The library will become visible in the side menu on the left.

Editing a Shape library

It is possible to edit an existing Shape library. You can add Shapes to the library, delete Shapes from the library or edit the parameters of an existing Shape in the library. To edit a Shape in the library, right-click with your mouse on the Shape name. A context menu will become visible:



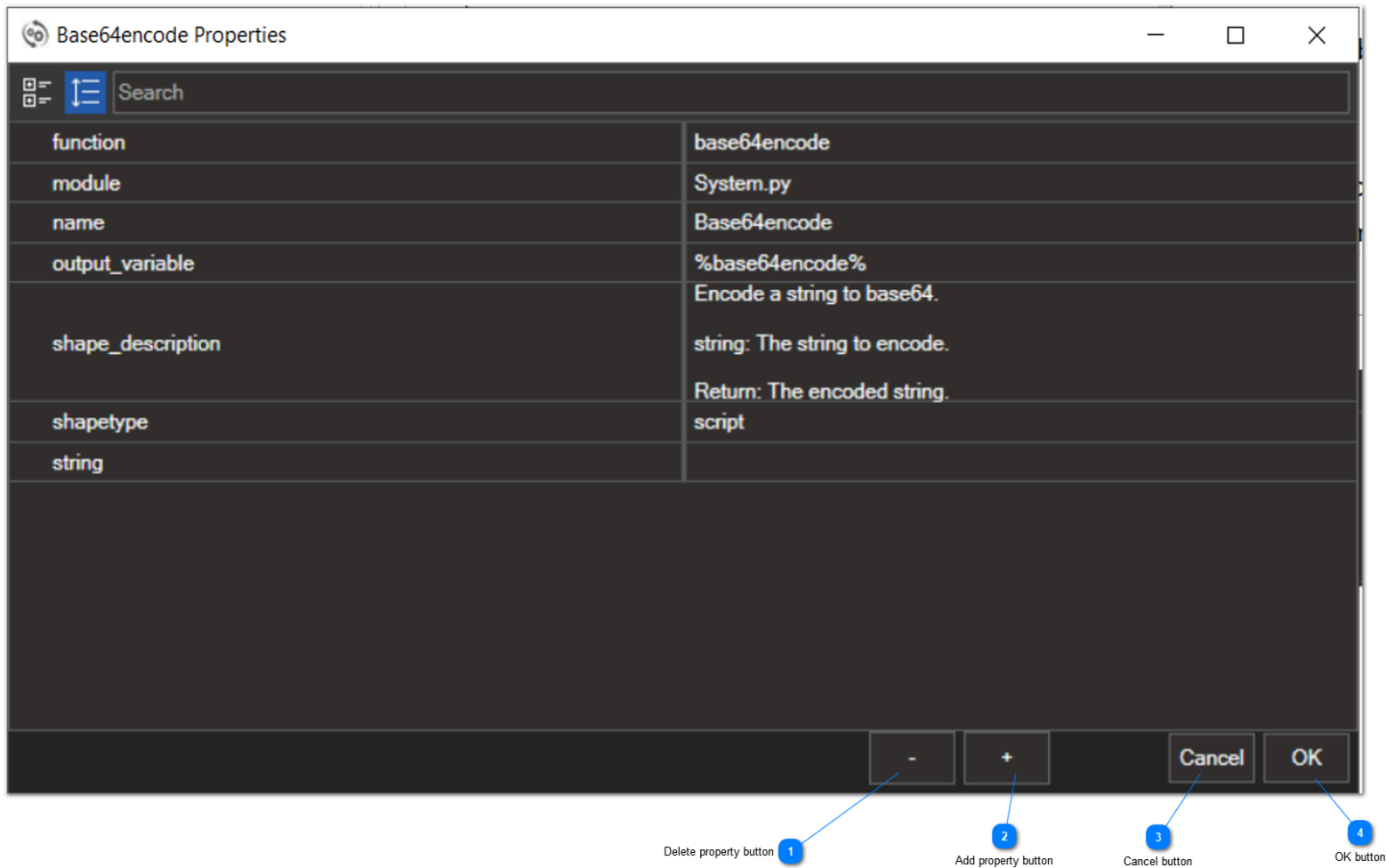
1 Rename option
Click this option when you want to rename the Shape. The rename window will become visible.

2 Delete option
Click this option when you want to delete the selected Shape from the library. A confirmation pop-up will be shown, asking you to confirm the deletion.

Clicking on "Yes" in this pop-up will permanently delete the Shape from the library. The library is automatically saved (without the deleted Shape) in the Template Folder.

3 Edit option

Click this option if you want to edit the parameters of the selected Shape. The Edit Shape properties window will be shown:



Please look to the [ShapeType](#) paragraph for a further explanation of this parameter. This screen is the only location for changing this option.

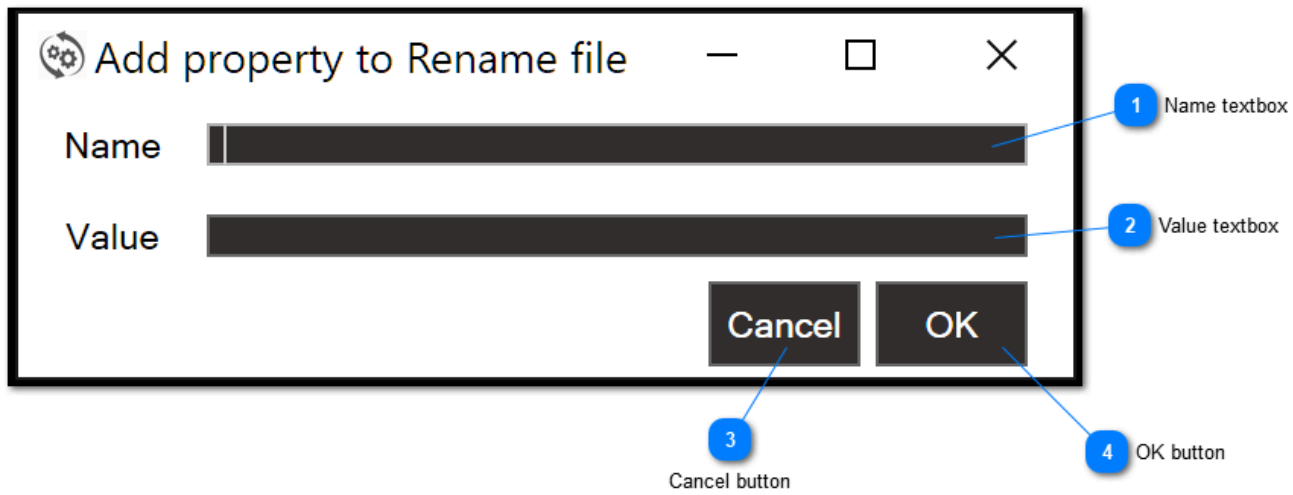
The Edit Shape properties window is justb like the Properties window, except for several extra buttons:

1 Delete property button

Button to delete a property from the Shape. A confirmation message is shown when clicking this button to prevent from deleting by mistake..

2 Add property button

Button to add a property to a Shape. The "Add property to Shape" window is shown:



1 Name textbox
Textbox for entering the name of the property. Property names need to be in lower case.

2 Value textbox
Text for entering the value for the property. This value is optional.

3 Cancel button
Button for closing the window without adding a property to the Shape.

4 OK button
Button for closing the window and adding the property to the Shape.

3 Cancel button
Button for closing the window without saving the properties to the Shape.

4 OK button
Button for closing the window and saving the properties to the Shape.

3.2. Available libraries

BPMN-RPA Studio makes use of Shape libraries (JSON files with the .tpl extension, which refers to Template). You can make those libraries yourself (see: [Creating custom Shapes](#)). BPMN-RPA Studio contains the following Shape libraries:

- **Basic shapes.tpl:** (Windows/Linux)
all the basic shapes you need to create simple flows (like: start, end, exclusive gateway, parallel gateway, create variables etc.)
- **Dialog.tpl:** (Windows/Linux)
to ask for user input by showing a modal dialog on the screen. This Shape template set makes use of the Python TKinter library ([TkDocs Tutorial - Installing Tk](#)).
- **Excel.tpl:** (Windows)
various functions to manipulate Excel content.
- **FTP.tpl:** (Windows/Linux)
functions to up- and download files to an FTP server together with functions for creating, moving, deleting and reporting FTP files and folders.
- **Images.tpl:** (Windows)
functions to get text from images and clicking on images.
- **Jira.tpl:** (Windows/Linux)
functions to control Jira ([Jira | Issue & Project Tracking Software | Atlassian](#)).
- **Keepass.tpl:** (Windows/Linux)
functions to control Keepass for storing and retrieving passwords ([KeePass Password Safe](#)).
- **Keyboard.tpl:** (Windows/Linux)
various functions for sending keyboard keystrokes to console or window.
- **Messagebox.tpl:** (Windows/Linux)
functions for showing messageboxes. This Shape template set makes use of the Python TKinter library ([TkDocs Tutorial - Installing Tk](#)).
- **Mouse.tpl:** (Windows/Linux)
various functions for sending mouse instructions.
- **MsGraph.tpl:** (Windows/Linux)
various functions for making use of the MsGraph library for reading email, files (onedrive) etc ([Microsoft Graph overview - Microsoft Graph | Microsoft Learn](#)).
- **OpenAi.tpl:** (Windows/Linux)
functions to make calls to the OpenAI GPT3 Artificial Intelligence Model (see: [Playground - OpenAI API](#))
- **Outlook.tpl:** (Windows)
various functions to manipulate Windows Outlook Desktop content.
- **PDF.tpl:** (Windows/Linux)
functions for creating and manipulating PDF files (get text, extract images etc).
- **Set Value.tpl:** (Windows/Linux)
functions for creating and manipulating variables in a flow.
- **SQLite.tpl:** (Windows/Linux)
functions for SQLite databases (create, read, update and delete data).
- **SQLserver.tpl:** (Windows/Linux)
functions for Microsoft SQLserver databases (create, read, update and delete data).
- **System.tpl:** (Windows/Linux)
functions for operating systems (like file- and folder operations, start program etc).
- **TextMining.tpl:** (Windows/Linux)
various TextMining functions (extract named entities, part-of-speech tagging etc.). You can even train and use your own classification models! This library is based on Spacy ([spaCy · Industrial-strength Natural Language Processing in Python](#)).
- **Web.tpl:** (Windows/Linux)
functions for automating web-pages.
- **Windows.tpl:** (Windows)
Functions for manipulating windows on the Windows desktop.




























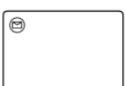








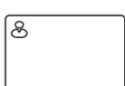

 **BPMN-RPA Studio**

- **Word.tpl:** (Windows)

Functions for automating Microsoft Word on the Windows Desktop.

3.3. The ShapeType parameter

A special property of a Shape in a Shape library is the "ShapeType" parameter (see: [Shape libraries - edit option](#)). The name of the shapetype will decide how the shape will look. Options available:

Event / Trigger	Start	End	DataObject Type	Symbol
None			None	
Message			Data Input	
Timer				
Conditional			Data Output	
Link			datastore	
Signal			Task type	Symbol
Error			Service	
Escalation			Send	
Termination			Receive	
Compensation			Instantiating Receive	
Cancel			Manual	
Multiple			Business Rule	
Parallel			User	
			Script	

4. Variables

Variables are used to catch the output of the Python code called, so the output can be used in the succeeding Shapes of the flow.

Naming convention

The % sign is used as brackets around a Variable name. For example, "%my_variable%" is the Variable 'my_variable'. When you use %my_variable% as an input, a Shape will use the value that has previously been stored in that Variable, so you should have an earlier Shape that assigned a value to %my_variable% as an output. By assigning output values to Variables, and then using them as input in later steps, you can pass information through a Workflow.

Variable values

You can store any type of information into a variable, like:

- Texts
- Numbers
- Booleans
- Lists
- Dictionary
- Class objects
- etc. etc.

System variables

System variables are pre-defined variables that provide information that can be used in flow attributes. The System variables content is set by the WorkflowEngine automatically and cannot be changed. All system variables begin with a double underscore between percent signs. Available variables:

- %__folder_desktop__%: returns the desktop folder of the current user
- %__folder_downloads__%: returns the downloads folder of the current user
- %__folder_system__%: returns the windows system folder
- %__month__%: returns the current month
- %__now__%: returns the datetime now() object
- %__now_formatted__%: returns today with time in the format 'dd-mm-yyyy_hhmmss'
- %__time__%: returns the current time
- %__time_formatted__%: returns the current time in 'hh:mm:ss' format
- %__today__%: returns the current date
- %__today_formatted__%: returns the current date in 'dd-mm-yyyy' format
- %__tomorrow__%: returns the date of tomorrow
- %__tomorrow_formatted__%: returns the date of tomorrow in 'dd-mm-yyyy' format
- %__user_name__%: returns the account name of the user that is currently logged in
- %__weeknumber__%: returns the current weeknumber
- %__year__%: returns the current year number
- %__yesterday__%: returns the date of yesterday
- %__yesterday_formatted__%: returns the date of yesterday in 'dd-mm-yyyy' format

4.1. Retrieving information from variables

Variables can hold any value (see [Variables](#)).


Value from list

In order to retrieve a specific item from a list in a variable, you must use the following format (notation): %VariableName[ItemNumber]%. The "ItemNumber" should be 0 for the first item of the list, 1 for the second and so on. For example, if you have a list that is stored in the variable %MyList% and contains 10 items, you can retrieve the first item with: %MyList[0]% and the last item with %MyList[9]%. For data tables, you must use the following notation: %VariableName[RowNumber][ColumnNumber]%.

Value from object or dictionary

If you would like to retrieve an attribute of a stored object or dictionary in a variable, then you must use the %VariableName.attributeName% notation. Just use the %VariableName% notation to retrieve the full object or dictionary.

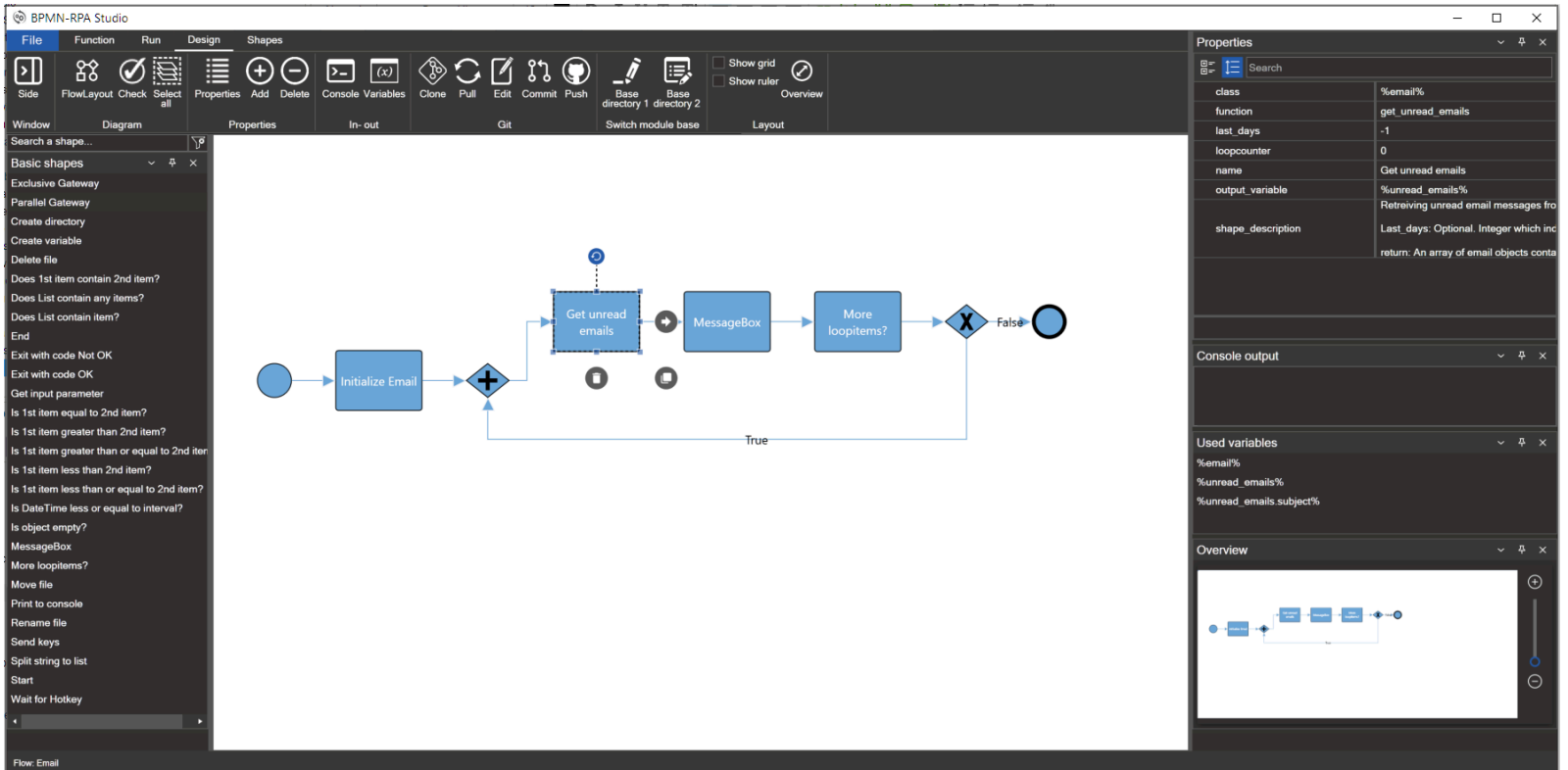
5. Loops

You can create loops by using an exclusive gateway .

An exclusive gateway should always have two sequence flow arrows: one with the label "True" and the other with the label "False". The actual true/false decision isn't made in the exclusive gateway itself, but in the last Shape before the exclusive gateway. A loop is started by a Shape that is calling a Python script that returns a list. The Shape is recognized as the start of the loop by adding/using the attribute 'Loopcounter'. The loopcounter number is the starting point for the loop (for returning the n-th element of the list). The Shape before the Exclusive Gateway should be the 'More loop items?' Shape. You can find this Shape in the predefined "Basic" Shapes. This Shape should only have two attributes: 'Function' with value 'loop_items_check' will call the loop_items_check() function in the WorkflowEngine object, and 'Loop_variable' with the variable name to loop as value. The loop_items_check() function will return True or False, which will be used by the WorkflowEngine to decide which connector/arrow to follow.

Example

Below is an example of a loop in a flow. Please notice the "loopcounter" attribute of the "Get unread emails" Shape. This shape saves its output in the variable "%unread_emails%", so the variable "%unread_emails%" must also be used in the "More loopitems?" Shape (as a value of the attribute "loop_variable").



The screenshot displays the BPMN-RPA Studio interface. The main workspace shows a flow diagram for a loop. The flow starts with a start node leading to an 'Initialize Email' task. This is followed by an 'Exclusive Gateway' (diamond with a plus sign). The flow then enters a loop structure: a 'Get unread emails' task, a 'MessageBox' task, and a 'More loopitems?' task. The flow then reaches another 'Exclusive Gateway' (diamond with an X). From this gateway, a 'True' path loops back to the start of the loop, and a 'False' path exits the loop. The 'Properties' panel on the right shows the configuration for the 'Get unread emails' task, including 'class: %email%', 'function: get_unread_emails', 'loopcounter: 0', and 'output_variable: %unread_emails%'. The 'More loopitems?' task is also visible in the diagram.

Special loop variable options

You can get the value of the loopvariable counter by using the '.counter' attribute of the loopvariable (p.e.: %test.counter%). To get the whole list that is looped, use the '.object' attribute of the loopvariable (p.e.: %test.object%) or just the variable name (like %test%).

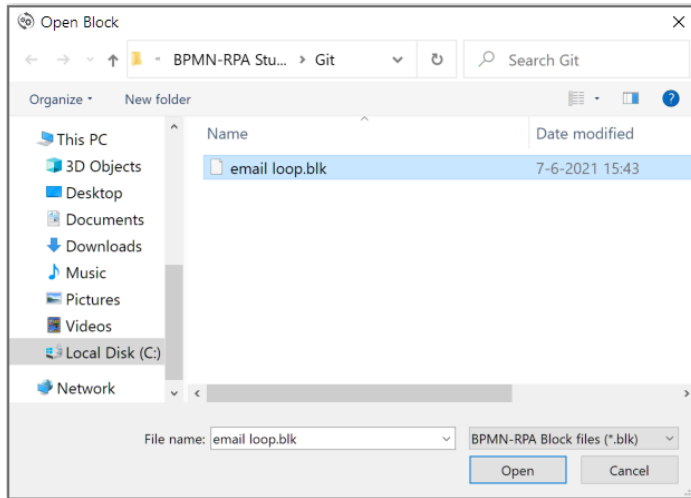
6. Blocks

Blocks (or: "building Blocks") are specific parts of a flow that you save or load into the current Flow design. This enables you to quickly create complex Flows. Blocks are saved in their own .blk format, to distinguish between complete Flows (.flw) and the saved Blocks (.blk).

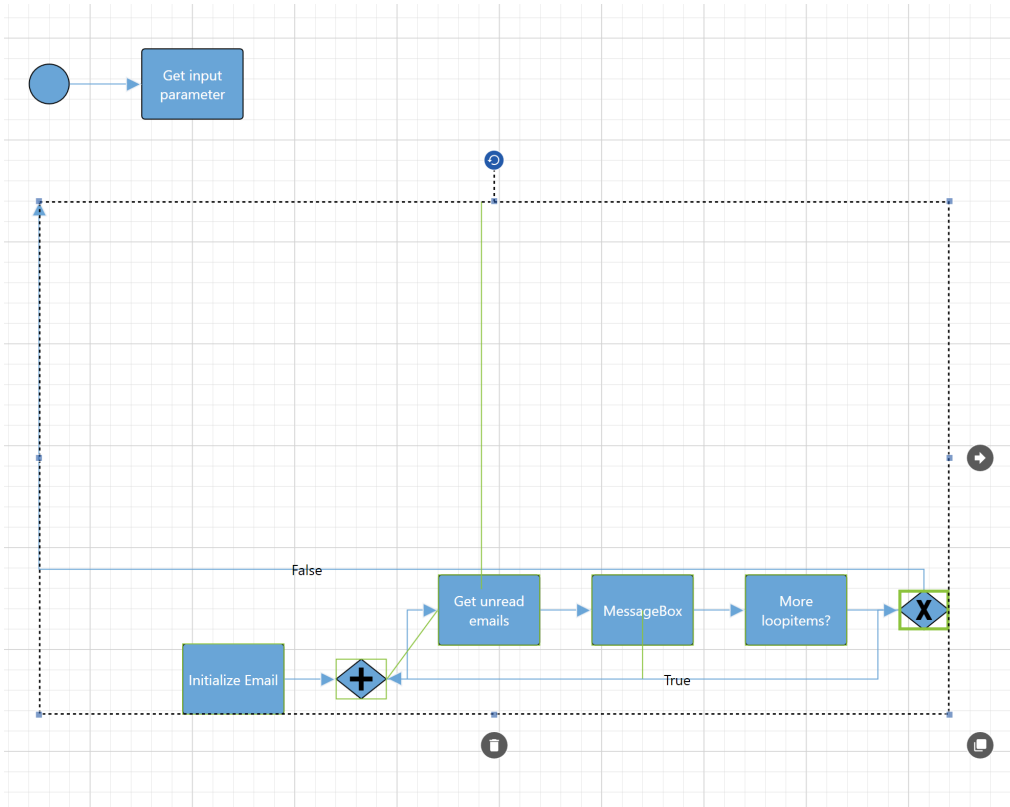
Open a Block



Click on the "Insert Blocks" button of the [Shapes tab](#). The Windows Open File Dialog will be shown:

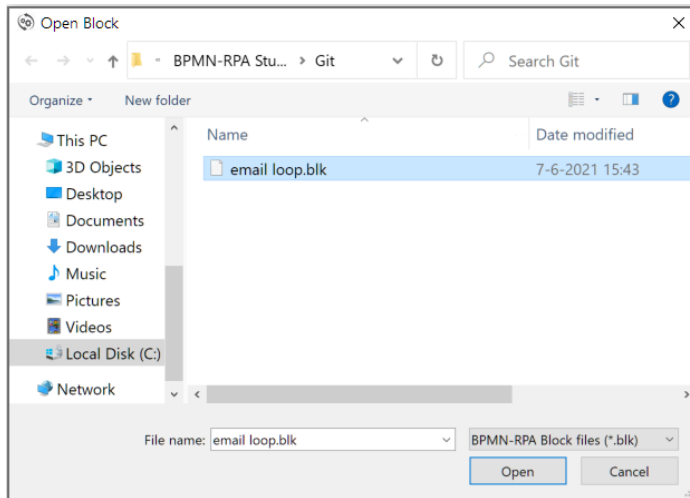


Open the Block (.blk) file you want to insert and click on "open", or double click the Block file. The Block will be inserted into your flow and all inserted Shapes and connectoes (thw Block) will be selected, so you can replace the Block to the location where you want to connect the Shapes in your Flow.



Save a Block

Click on the "Save Blocks"  button of the [Shapes tab](#). The Windows Save File Dialog will be shown:



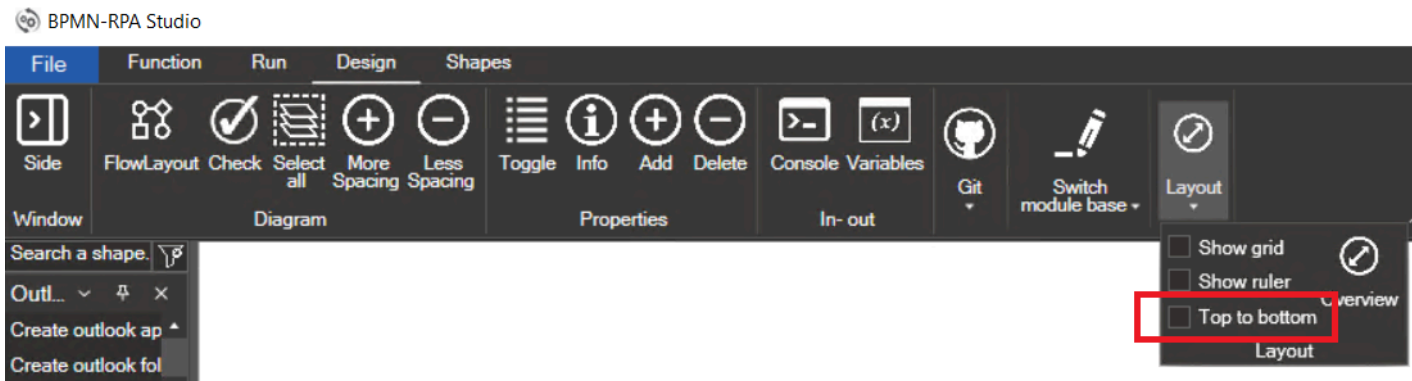
Enter the name you want to give to the Block in the "Filename" field and click on "save" to save the Block file.

7. Building a Flow

You can build your flow by using items from loaded template sets:

- Drag an item on the canvas, or:
- Double click on the item in the Shapes tab

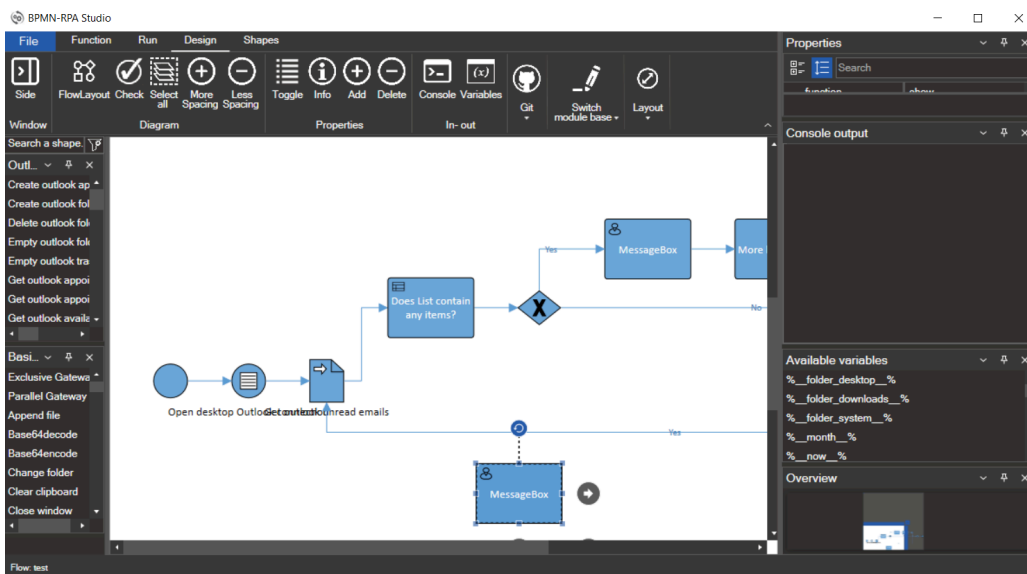
You can choose your layout: from "left to right" or "top to bottom" by selecting or deselecting the "Top to bottom" checkbox.



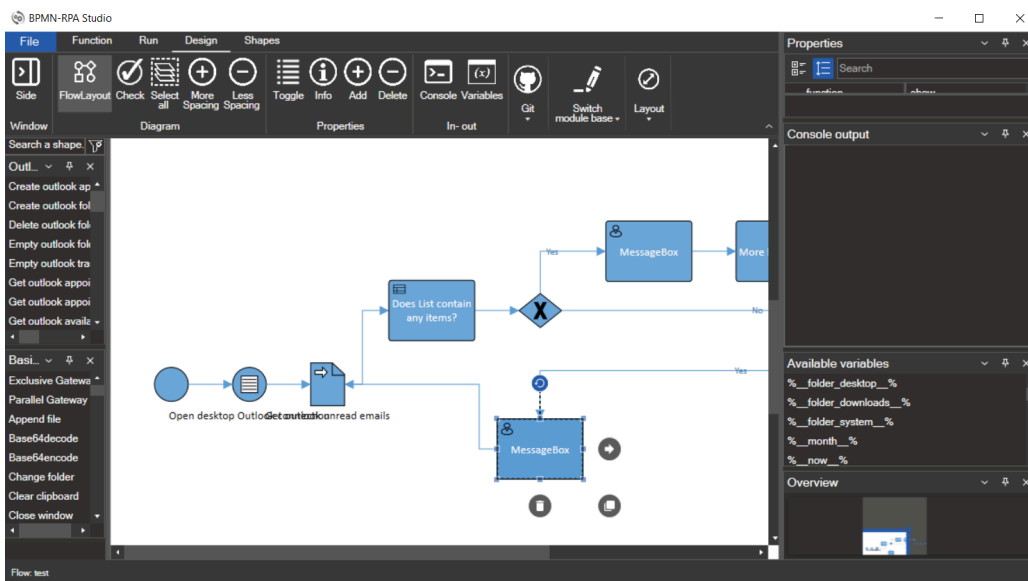
When the "Top to bottom" checkbox is unchecked with an empty canvas and you double click on an item in one of the template sets, the start-item will be added automatically. When dragging the first step of the flow, the start shape will automatically align.

Inserting a step in a flow

1. select the step that needs to be inserted into the flow and release your mousebutton.



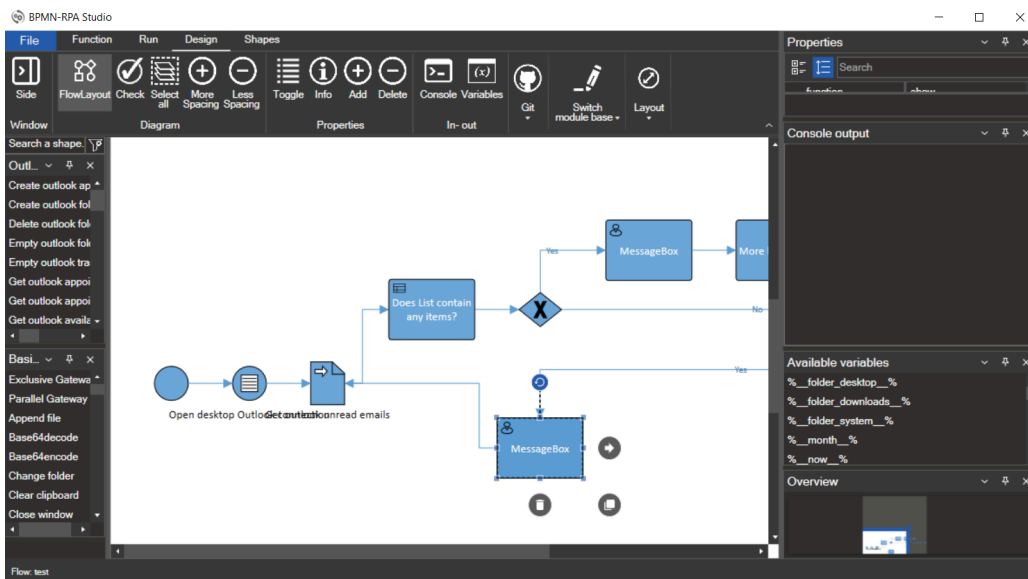
2. Hold the CTRL-Key.
3. While holding the CTRL-Key, drag the step over the connector where you want to insert it into the flow.



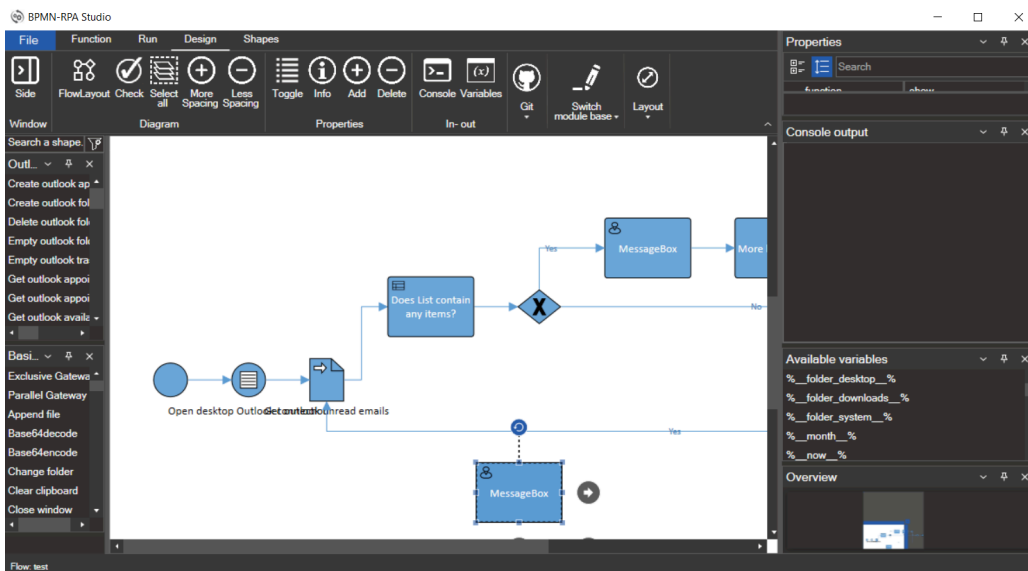
4. The connector will automatically split and the step is inserted.
5. Release the CTRL-Key and your mouse button.

Repositioning or deleting a step in a flow

1. select the step that needs to be repositioned or deleted from the flow and release your mousebutton.



2. Hold the CTRL-Key.
3. While holding the CTRL-Key, drag the step outside the flow.
4. The connector will automatically connect to the next step of the flow.



5. Release the CTRL-Key and your mouse button.

8. Running your Flows

You have several options to run your Flows:

- by generating a single Python script in [BPMN-RPA Studio](#) that can contains both the code to start the WorkflowEngine as the flow steps itself (see: [Script generation](#)).
- by using the [BPMN RPA Starter.py](#). This is a Python module which accepts commandline inputs.

The format for executing flows:

```
<path to Python.exe> BPMN_RPA_Starter.py <full path to your Flow (.flw file)> <Input string (if any)>
```

- start the flow with the WorkflowEngine in code.

Example code:

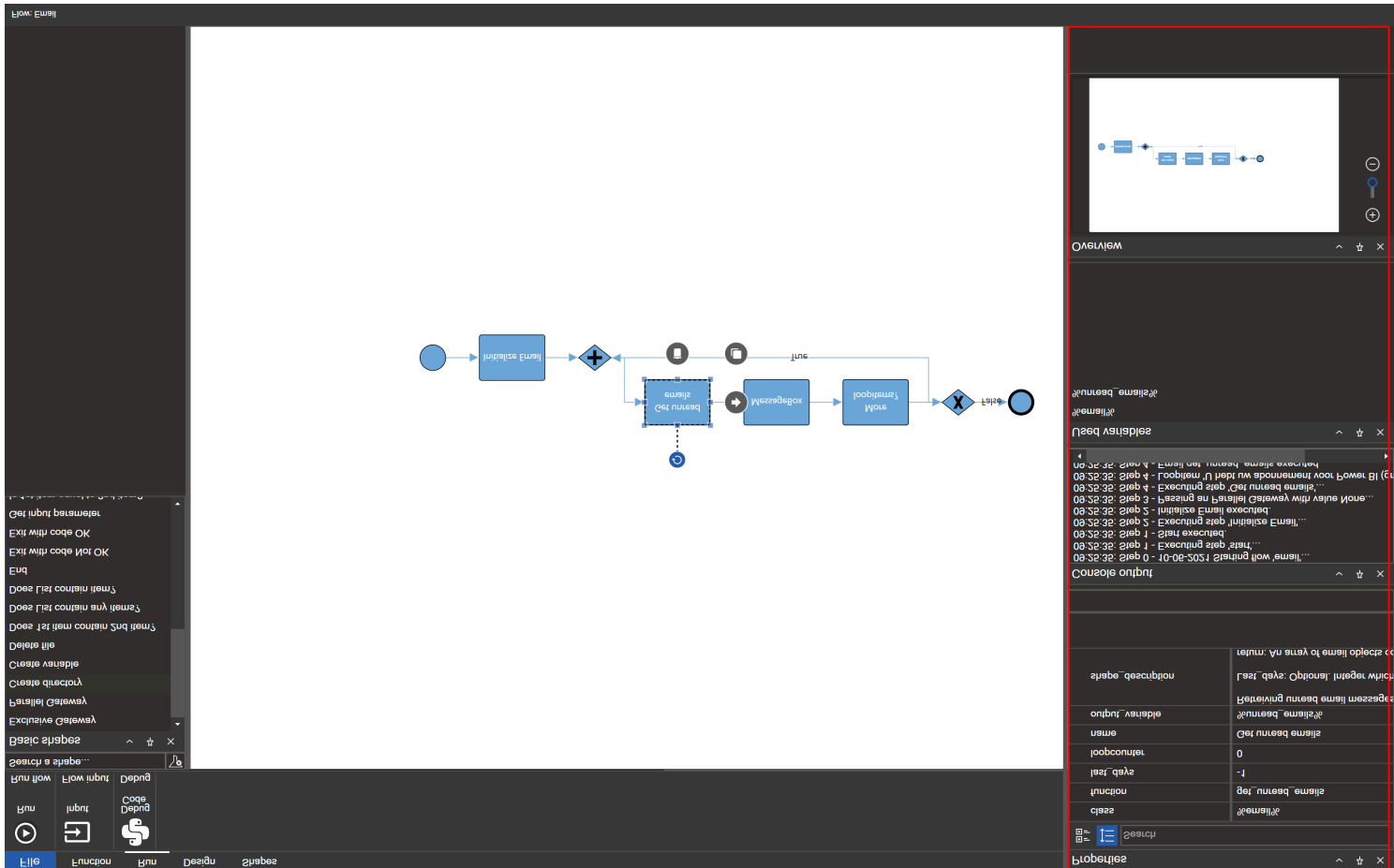
```
inp = "testinput" # Or: '{"first": "Hello","second": "World!"}'
engine = WorkflowEngine(inp)
doc = engine.open("test.xml")
steps = engine.get_flow(doc)
engine.run_flow(steps)
```



9. Side window

The right Side window (referred to as "Side window") is a container that is the default position for showing 4 windows:

- [Properties window](#)
- [Console window](#)
- [Variables window](#)
- [Overview window](#)

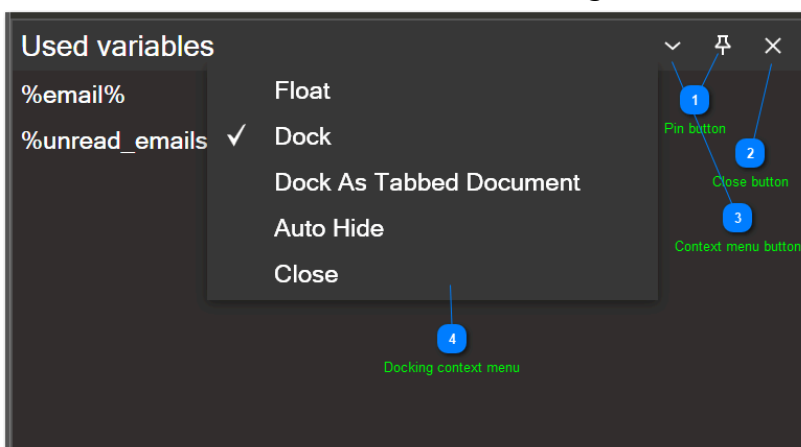
The Side window (marked red):



The Side window can be hidden or made visible with the "Side" button  in the [Design tab](#).

Dockable windows

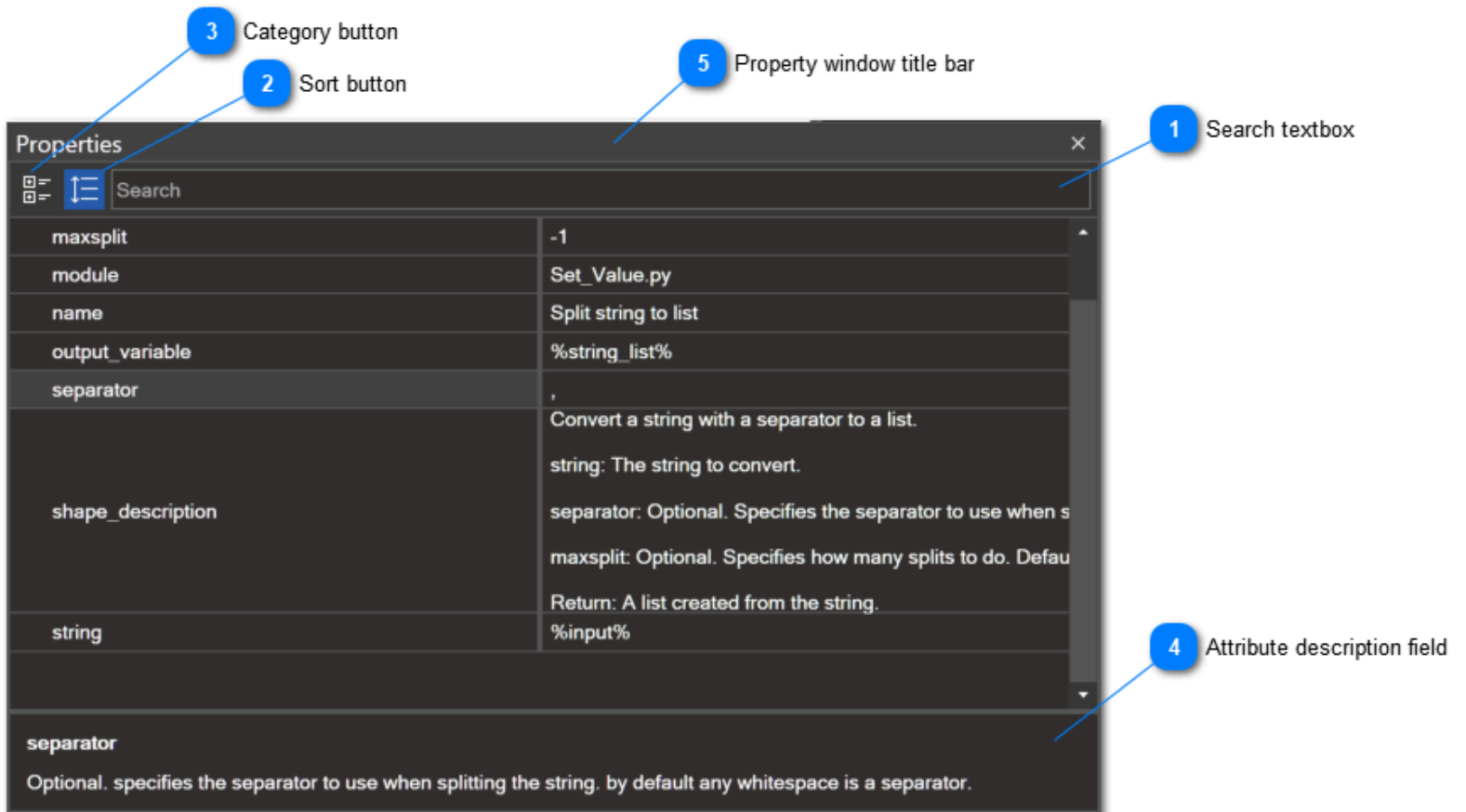
All four windows in the Side window are dockable. This means you can drag the window by its title bar and position it anywhere on the screen. When dragged outside the Side window, the windows will be resizable. Each window in the Side window has its docking menu in the title bar. For instance:



- 1 Pin button**
This button will collapse the window to its title bar or restore it to its expanded position. Right clicking the titlebar will show its content while collapsed.
- 2 Close button**
Button to close the window.
- 3 Context menu button**
Button to show or hide the docking contextmenu
- 4 Docking context menu**
Menu with all available options to dock/undock/hide/close the window.

9.1. Properties window


The Properties window will show all the properties of the currently selected Shape. Although the Properties Window is dockable (you can drag it by its title bar to any location on the screen), its default location is in the right Side window.



- 1 Search textbox**
Search for an attribute by using this textbox.
- 2 Sort button**
Button to sort the attributes alphabetically (up/down).
- 3 Category button**
Show attributes by category.
- 4 Attribute description field**
The description of the currently selected attribute.
- 5 Property window title bar**
The title bar of the property window. Use this to dock or undock the properties window from the [Side menu](#).

9.1.1. Shape description

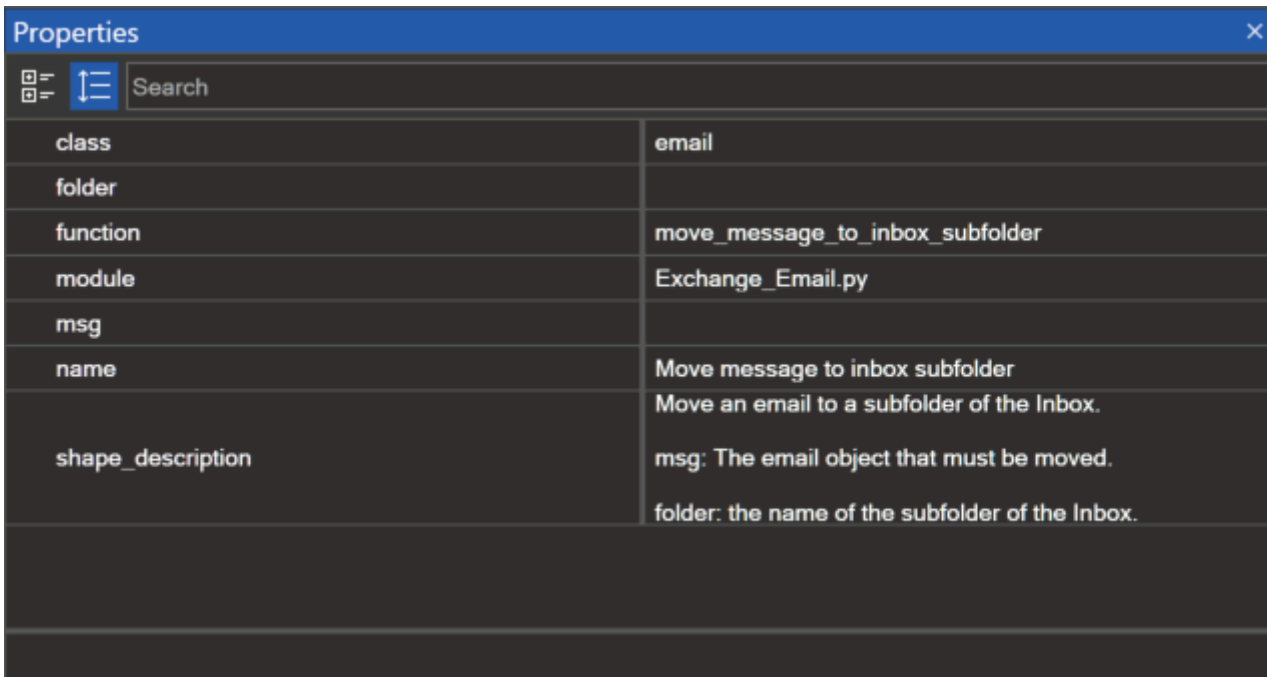
In the [Properties window](#), each property you select will show the part of the docstring that you've added to your Python code. To make this work, you must use the **reStructuredText** format for creating Python docstrings.

When a Shape library is created with the "Module to library" button  in the Shapes tab, The docstring for each function or class is automatically extracted and saved as the "shape_description" property of the Shape (see [Shape properties](#)).

An example:

```
def move_message_to_inbox_subfolder(self, msg: object, folder: str):
    """
    Move an email to a subfolder of the Inbox.
    :param msg: The email object that must be moved.
    :param folder: the name of the subfolder of the Inbox.
    """
    to_folder = self.account.inbox / folder
    getattr(msg, "move")(to_folder)
```

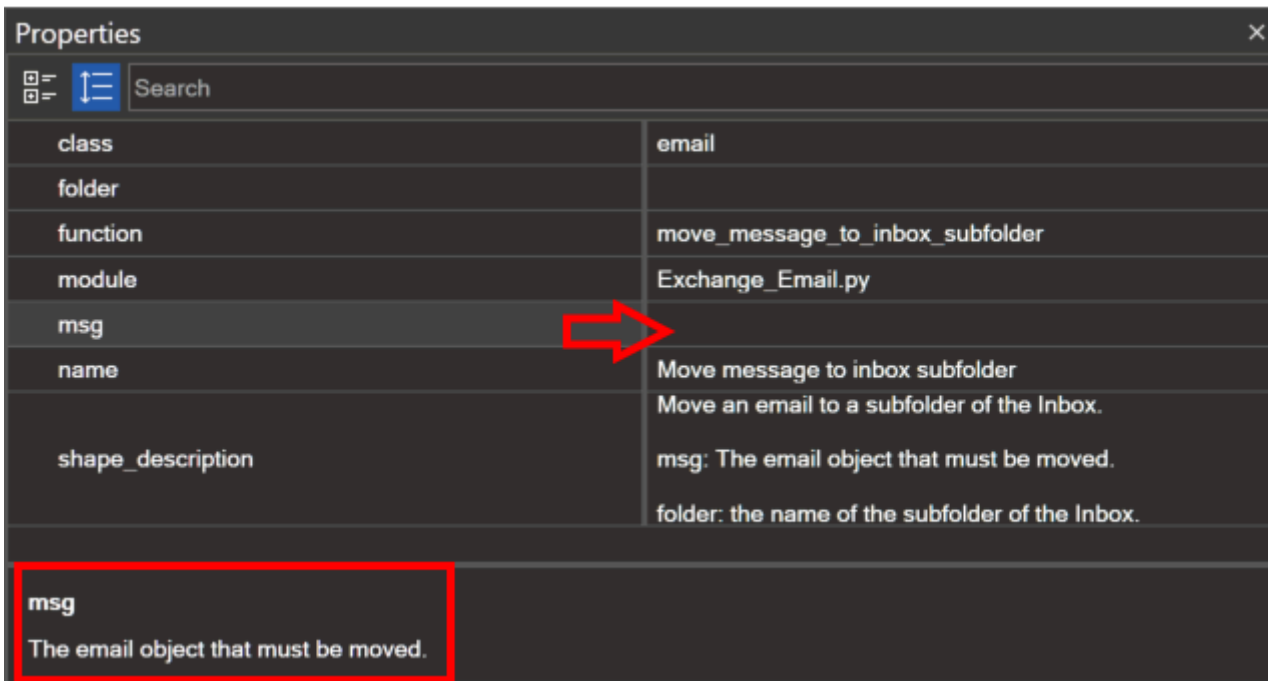
The description is extracted and saved as the "shape_description" property of the function:



The screenshot shows a 'Properties' window with a search bar and a table of properties. The 'shape_description' property is highlighted, showing its value as a docstring with parameter descriptions.

Property	Value
class	email
folder	
function	move_message_to_inbox_subfolder
module	Exchange_Email.py
msg	
name	Move message to inbox subfolder
	Move an email to a subfolder of the Inbox.
shape_description	msg: The email object that must be moved. folder: the name of the subfolder of the Inbox.

Please notice that the *":param"* docstring references are deleted and only the readable docstring text remains. The [Properties window](#) can now show the description for each property by extracting the parameter by its name from the shape_description:

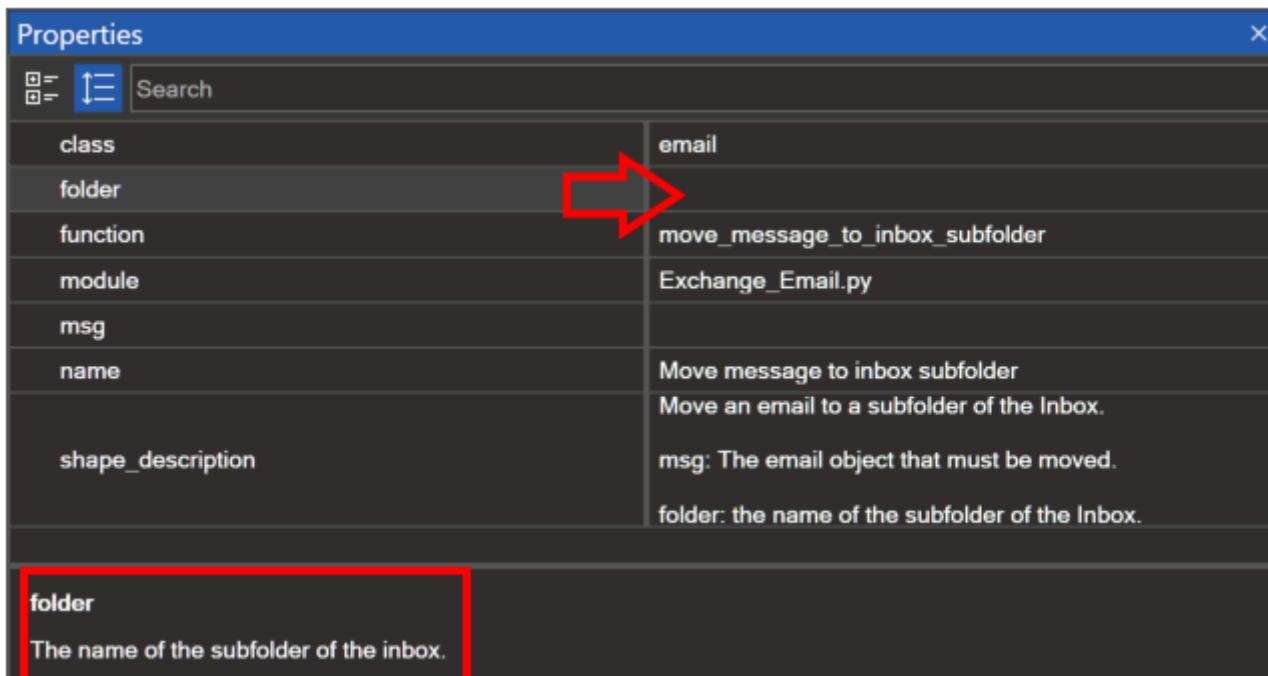


Properties

class	email
folder	
function	move_message_to_inbox_subfolder
module	Exchange_Email.py
msg	
name	Move message to inbox subfolder Move an email to a subfolder of the Inbox.
shape_description	msg: The email object that must be moved. folder: the name of the subfolder of the Inbox.

msg
The email object that must be moved.

and:




Properties

class	email
folder	
function	move_message_to_inbox_subfolder
module	Exchange_Email.py
msg	
name	Move message to inbox subfolder Move an email to a subfolder of the Inbox.
shape_description	msg: The email object that must be moved. folder: the name of the subfolder of the Inbox.

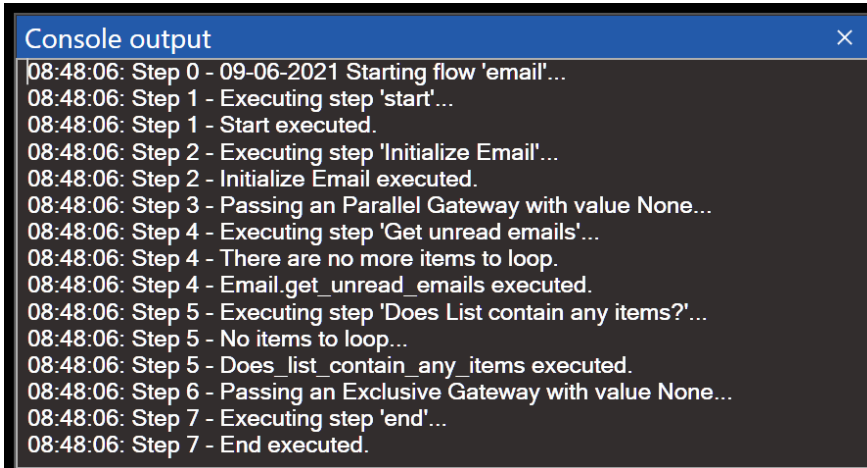
folder
The name of the subfolder of the inbox.

9.2. Console window

The Console window holds the output of a Flow that you have executed by clicking on the "Run" button  in the [Run tab](#). Although the Console Window is dockable (you can drag it by its title bar to any location on the screen), its default location is in the right Side window.

The console output will be refreshed each time you click on the "Run" button. The console output is exactly the same as the output you get by running the Flow from the commandline of your operating system.

An example of the Console window output:



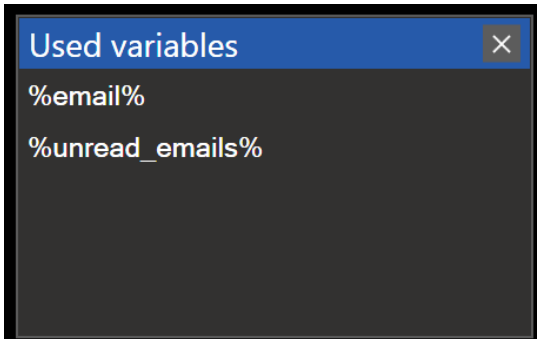
```
Console output
08:48:06: Step 0 - 09-06-2021 Starting flow 'email'...
08:48:06: Step 1 - Executing step 'start'...
08:48:06: Step 1 - Start executed.
08:48:06: Step 2 - Executing step 'Initialize Email'...
08:48:06: Step 2 - Initialize Email executed.
08:48:06: Step 3 - Passing an Parallel Gateway with value None...
08:48:06: Step 4 - Executing step 'Get unread emails'...
08:48:06: Step 4 - There are no more items to loop.
08:48:06: Step 4 - Email.get_unread_emails executed.
08:48:06: Step 5 - Executing step 'Does List contain any items?'...
08:48:06: Step 5 - No items to loop...
08:48:06: Step 5 - Does_list_contain_any_items executed.
08:48:06: Step 6 - Passing an Exclusive Gateway with value None...
08:48:06: Step 7 - Executing step 'end'...
08:48:06: Step 7 - End executed.
```

9.3. Variables window

The Variables window is an aid to assist creating your Flow. This window will hold all the variables that you have used in your flow. Although the Variables Window is dockable (you can drag it by its title bar to any location on the screen), its default location is in the right Side window.

Only the 'root' of the variable is shown, so any references to attributes (like for example %email.subject%) or items (like %email[1]%) will not show up in this window.

An example of the Variables window:



9.4. Overview window

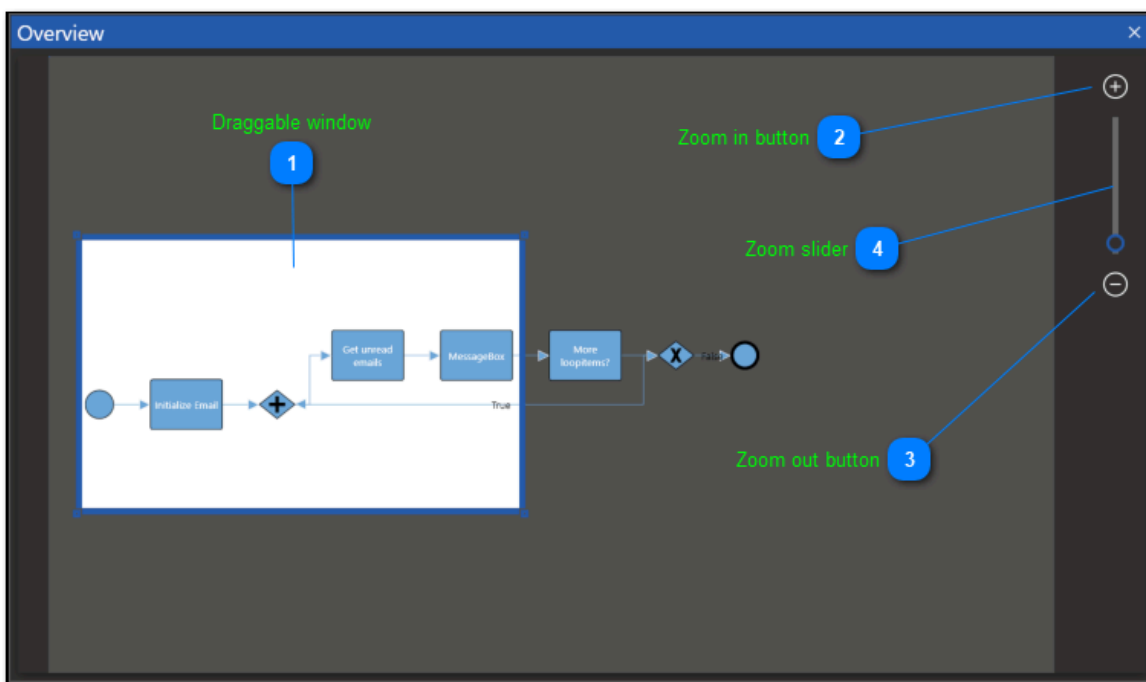
The Overview window will show an overview of the current Flow. Although the Overview Window is dockable (you can drag it by its title bar to any location on the screen), its default location is in the right Side window.

The window is used to display a preview (overall view) of the entire content of a Flow. This helps you to look overall picture of large Flow and easy to navigate (pan or zoom) to a particular position of the page.

When you work on a huge and complex diagram, you may not know the part where you are actually working, and navigating from one part to another might be difficult. To navigation, zoom out entire Flow and find where you are. This solution is not suitable when you need some frequent navigation.

The Overview window solved this problem by displaying a preview (overall view) of the entire Flow with option to pan and zoom.

Example of an Overview window:



1 Draggable window

Drag this window to any position in the overview to show that part of the Flow.

2 Zoom in button

Button to zoom the draggable window in on the Flow.

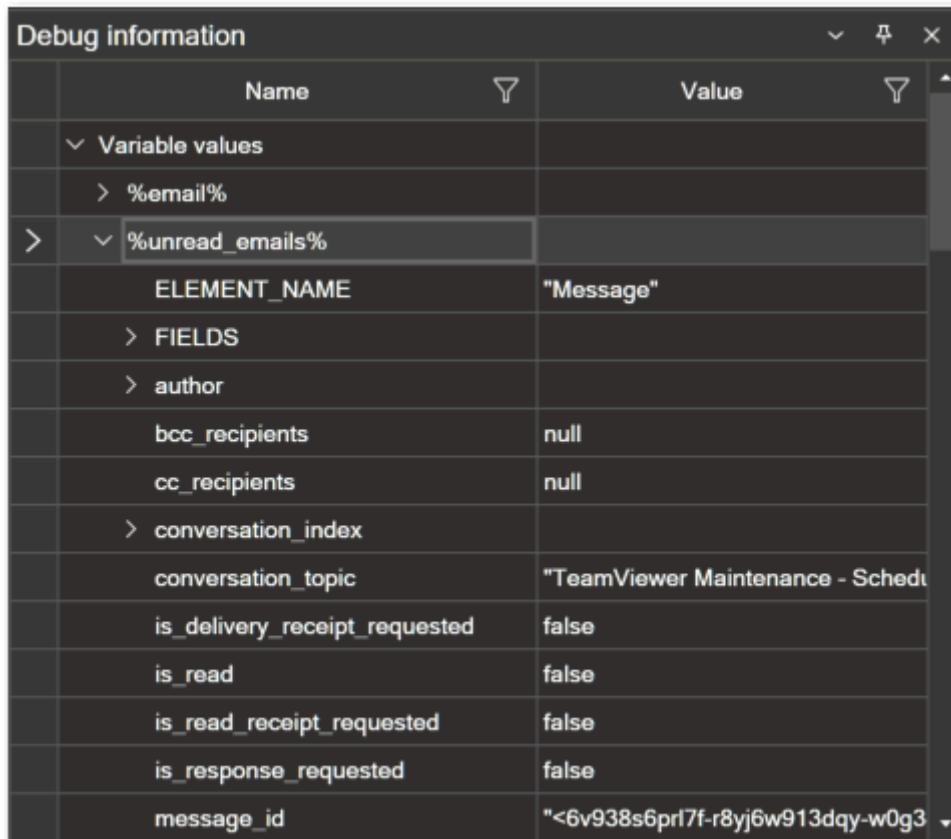
3 Zoom out button

Button to zoom the draggable window out from the Flow.

4 Zoom slider

Slider to zoom the draggable window in- or out from the Flow.

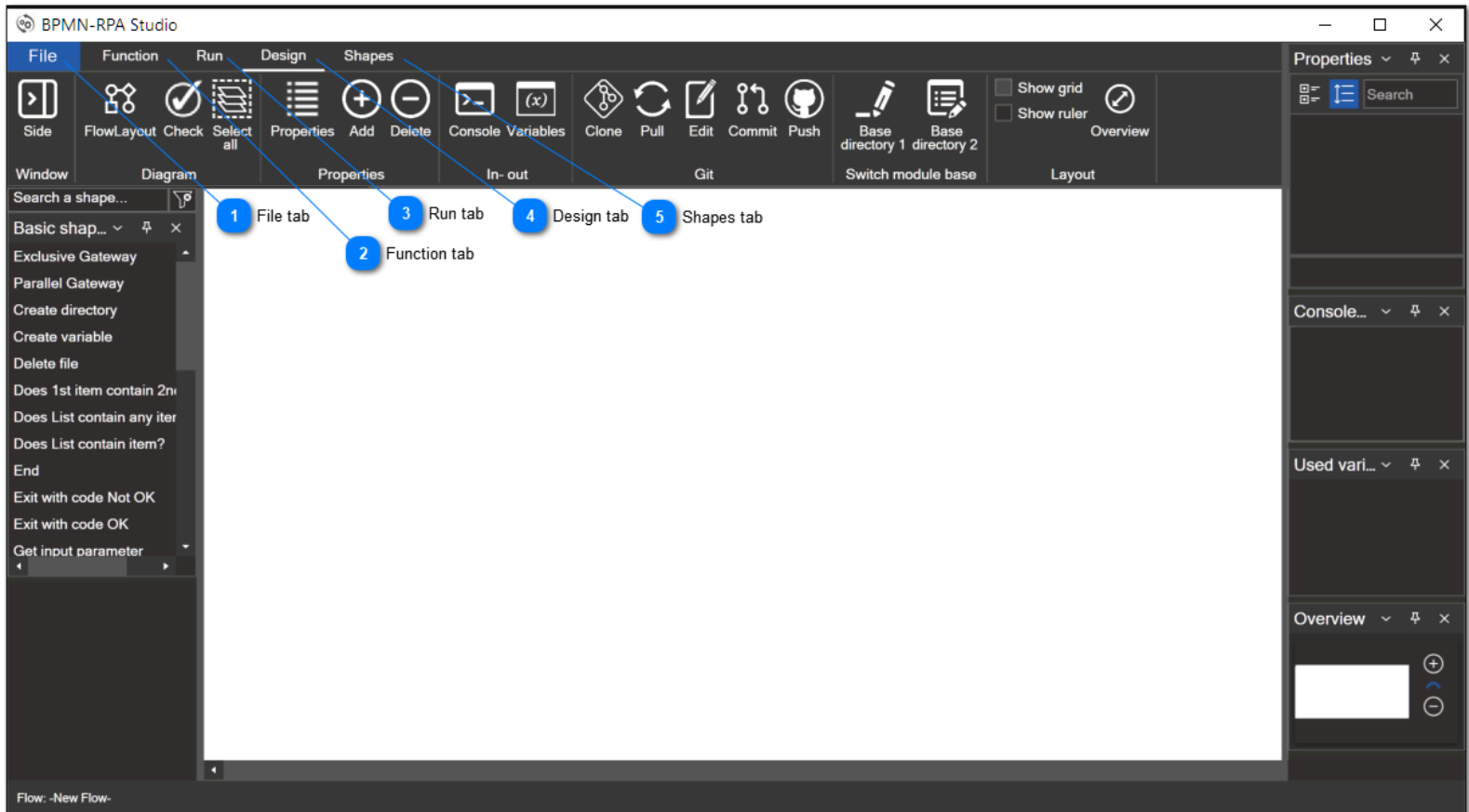
9.5. Debug window



Name	Value
Variable values	
> %email%	
> %unread_emails%	
ELEMENT_NAME	"Message"
> FIELDS	
> author	
bcc_recipients	null
cc_recipients	null
> conversation_index	
conversation_topic	"TeamViewer Maintenance - Schedu"
is_delivery_receipt_requested	false
is_read	false
is_read_receipt_requested	false
is_response_requested	false
message_id	"<6v938s6pr17f-r8yj6w913dqy-w0g3"

The Debug window is only shown when you are debugging the Flow [step-by-step](#). When variables are used in the flow, this window will show the variable names with the actual values that the variables will contain each step of the Flow.

10. Menu tabs



- 1 File tab**
This tab will bring you the backstage window and shows you the "[File menu](#)".
- 2 Function tab**
This will bring up the menu option buttons of the "[Function tab](#)".
- 3 Run tab**
This will bring up the menu option buttons of the "[Run tab](#)".
- 4 Design tab**
This will bring up the menu option buttons of the "[Design tab](#)".
- 5 Shapes tab**
This will bring up the menu option buttons of the "[Shapes tab](#)".

10.1. File menu



1 New
This button will empty the flow designer window so a new flow can be created.

2 Open
This button will bring up the Windows "Open file" dialog, making it possible to open a .flw file. After opening the .flw file, BPMN-RPA Studio will automatically switch to the designer view, showing the opened flow.

3 Save
This button will bring up the Windows "Save file" dialog, so the current flow can be saved as a .flw file.

4 Print
This button will pop up the print window which offers several options pro printing out the current flow.

5 Script
This button will create a single Python script from your current flow. Both the steps in the flow as the initiation of the BPMN-RPA Workflow engine are part of this script, making it a single file that can be executed by the Python.exe program. See [Script generation](#).

6 Image
This button will bring up the Export window for exporting the current flow to different image formats.

7 Options
This button will show the "[Options](#)" window.

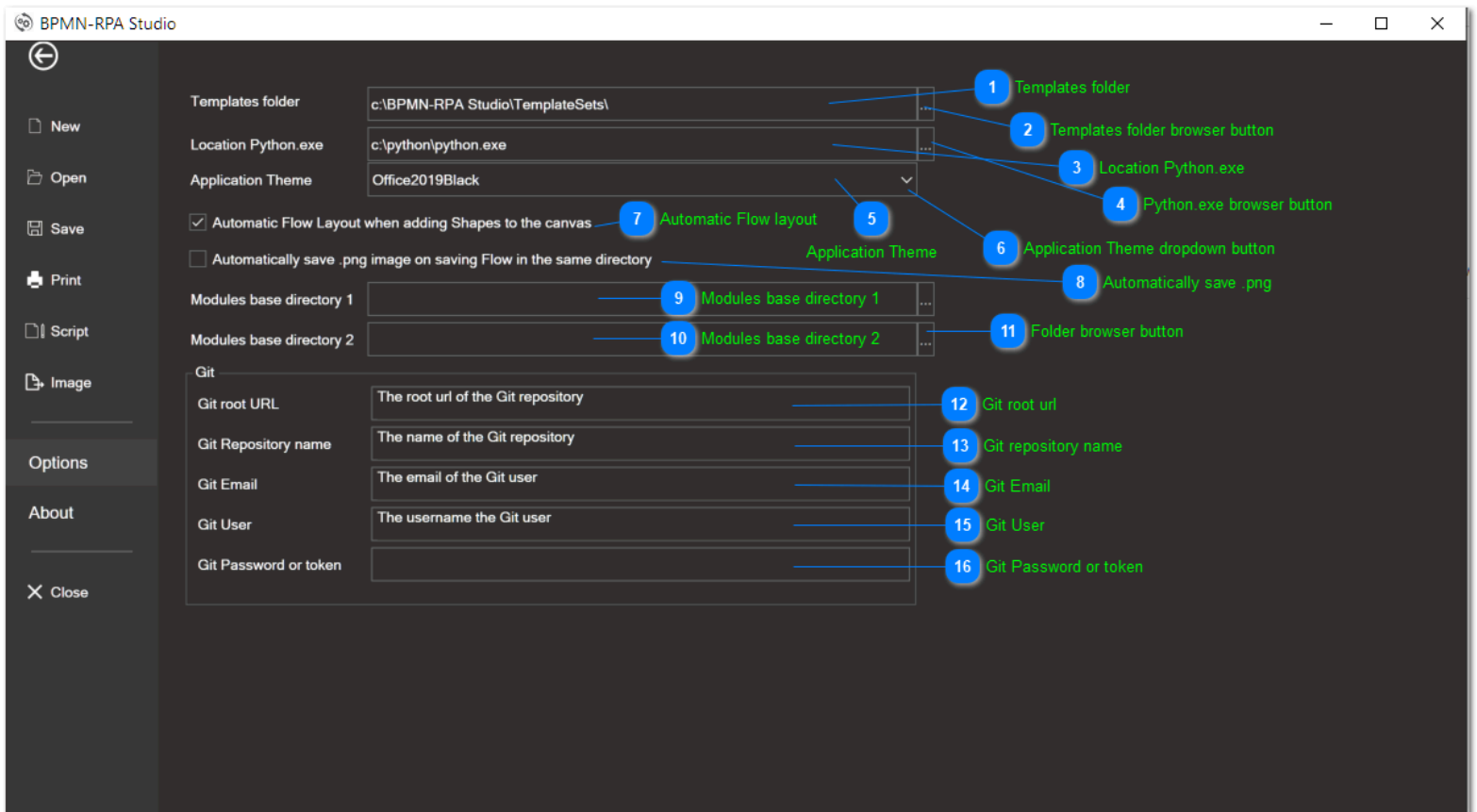
8 About

This button will show you information about the BPMN-RPA Program, like: author, copyright notice and license information. It also offers you the possibility to make a donation to the author of the program.

9 Close

This button closes the application. All settings made in the "[Options](#)" window will be automatically saved.

10.1.1. Options window



- 1 Templates folder**
 This is the location where all your shape templates will be stored. This needs to be a location with full read and write access. The default value for this field is the "TemplateSets" subfolder of your installation folder.
- 2 Templates folder browser button**
 Click this button to browse for the location that needs to be set as your Templates folder. When you've selected the folder, the location will be shown in the "Templates folder" textbox.
- 3 Location Python.exe**
 This textbox should hold the full path (including "python.exe") to the location where python is installed. BPMN-RPA needs this location to run your flows.
- 4 Python.exe browser button**
 Click this button to browse for the location where Python.exe is located. When you've selected the location, it will be shown in the "Location Python.exe" textbox.
- 5 Application Theme**
 You can change the theme of the application by selecting the preferred theme. The current theme that is applied to the program is shown in this textbox. You can choose a different application by clicking on the dropdown button.
- 6 Application Theme dropdown button**
 Click this button to show the list of available themes for this application. When you choose a different Theme from the current applied Theme, a warning message box is shown that you need to restart the application for the changes to take effect.

- 7 Automatic Flow layout**

When creating a flow, you will be able to automatically rearrange/change the layout of your flow. When this checkbox is checked, all the shapes in the currently visible flow will be rearranged and aligned into a flow layout. Please be aware that the automatic flow layout only will be triggered when you add a shape to the flow by double clicking the shape that you want to add. You can always use the "Flow Layout" button of the "Design" tab to apply the flow layout to all shapes.
- 8 Automatically save .png**

When this checkbox is checked, the program automatically will save a PNG image of the current flow. This PNG image will have the same name as the flow and will be saved into the same directory as your flow.
- 9 Modules base directory 1**

BPMN-RPA studio makes it possible to switch between different locations of your modules. This textbox holds the first base location of your Python modules.
- 10 Modules base directory 2**

BPMN-RPA studio makes it possible to switch between different locations of your modules. This textbox holds the second base location of your Python modules.
- 11 Folder browser button**

Use this button to browse for the folder location and update the textbox with the selected location.
- 12 Git root url**

This is the root location (without the name of the repository) of your GIT repository. P.e.: the root location of your GitHub or GitLab repository.
- 13 Git repository name**

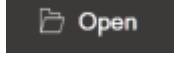
The name of the Git repository where you want to store your BPMN-RPA Studio files.
- 14 Git Email**

The email address you will use when committing and pushing your updates to the GIT repository. This email address is not needed for authentication, but is only used to show to others who has made the updates in the repository.
- 15 Git User**

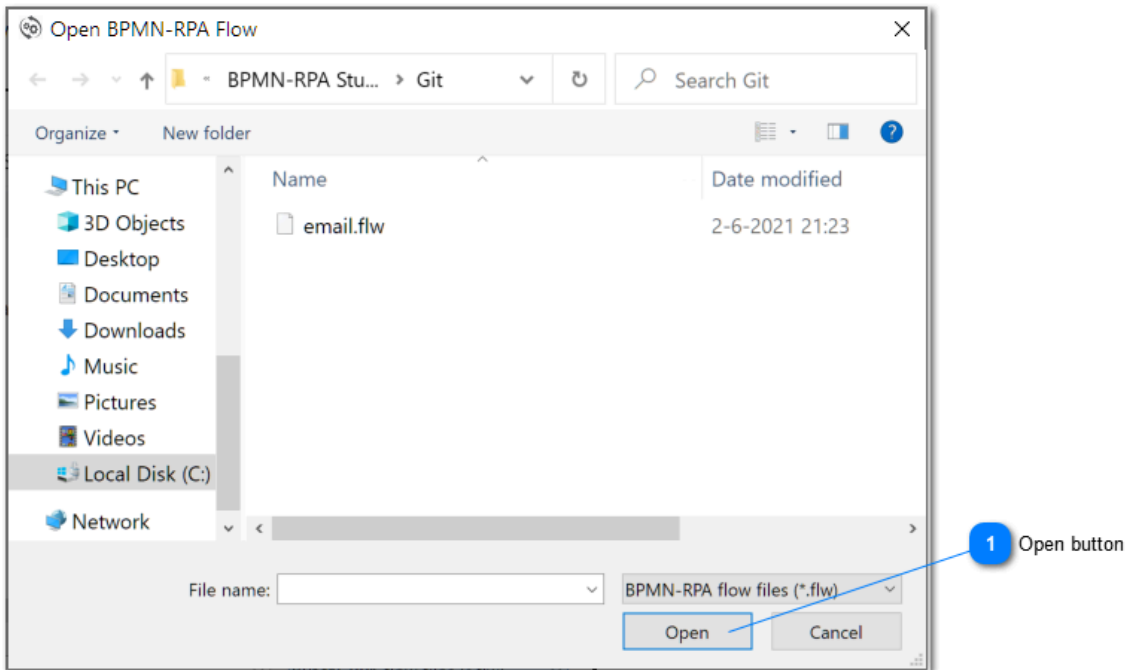
The GIT username that will be used for GIT authentication. P.e.: your GitLab or GitHub account name.
- 16 Git Password or token**

The GIT password or token that will be used for GIT authentication. P.e.: your GitLab or GitHub password or token.

10.1.2. Opening an existing Flow

You can open an existing flow by clicking on the open button  of the [File menu](#).

When you click this button, the Windows "Open File"-dialog will be shown:



1 Open button

Button for opening an existing flow from a file.

Browse for the file that you would like to open and click on the "Open button". Double clicking on the file will also open the flow from the file.

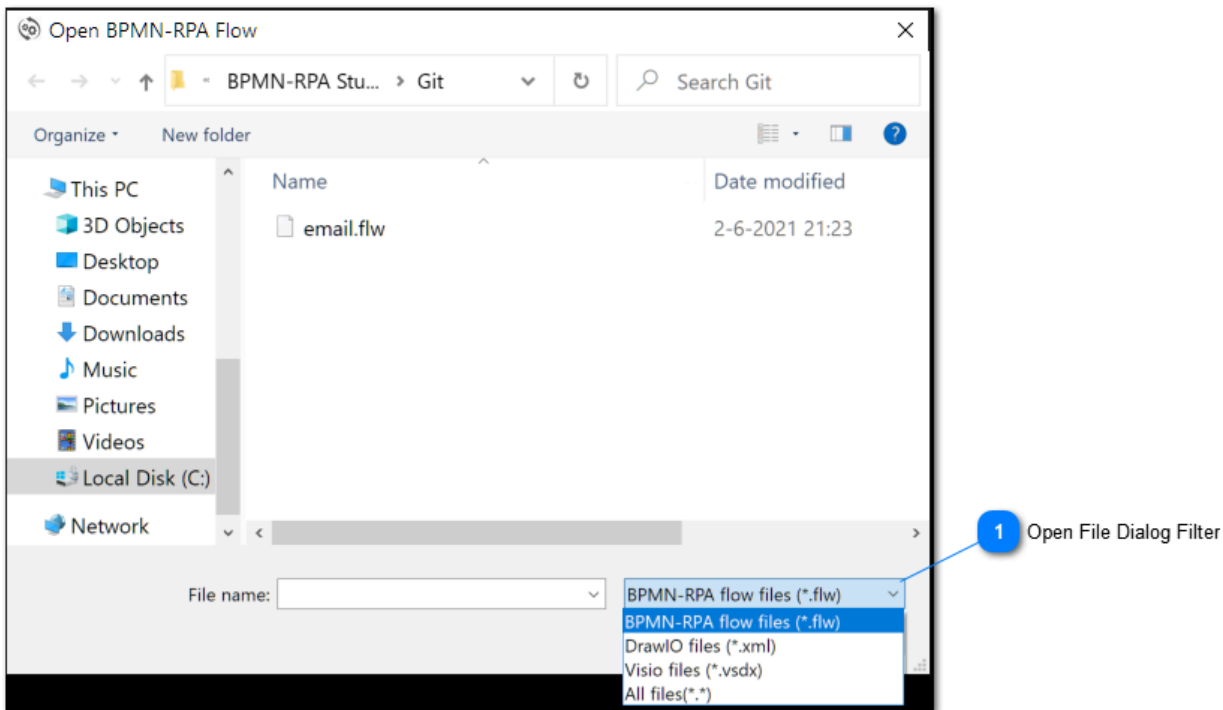
Opening flows created in other programs than BPMN-RPA Studio:

When the Open file dialog is opened, default all BPMN-RPA Studio flow files with the extension ".flw" will be filtered.

BPMN-RPA Studio can open flow files that are created with:

- BPMN-RPA Studio (.flw files)
- DrawIO (.xml files)
- Microsoft Visio (.vsdx files)

You can change the filter and show all files with the same extension by selecting the desired file option in the Open File Dialog Filter:




1 Open File Dialog Filter

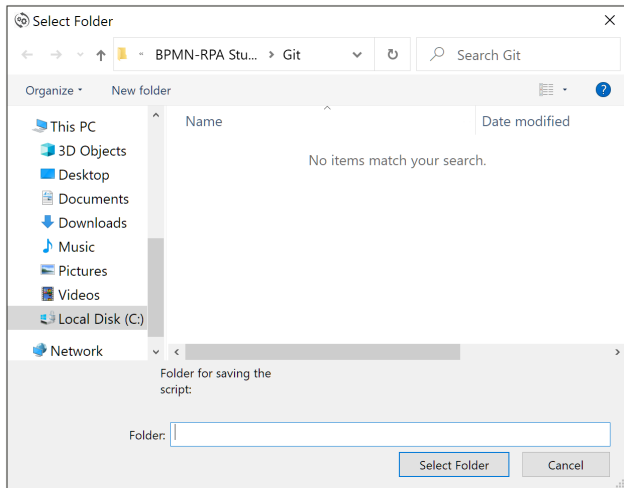
Filter options for showing only the files with the chosen file extension. BPMN-RPA Studio will filter out .flw, .xml and .vsdx files.

10.1.3. Script generation

BPMN-RPA Studio makes it possible to generate a single Python file from the Flow you've created. A single Python file makes it easier to automatically execute the code, p.e. by a scheduler or other programs. The code that is generated will hold both the flow steps (as a JSON object) as the initiation of the BPMN-RPA Workflow engine.

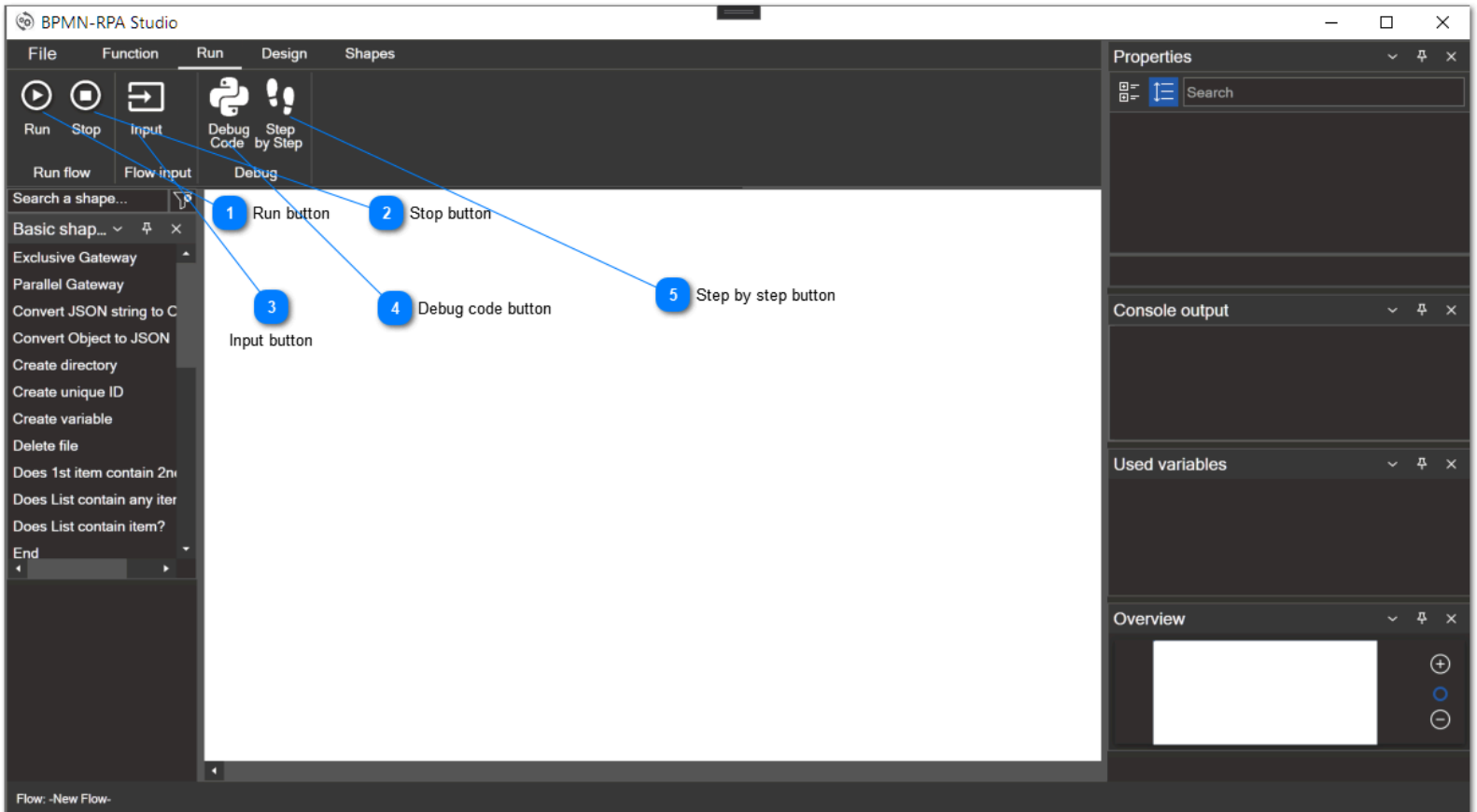
To generate a single Python script from your Flow:

- Click the "Script" button  in the [File menu](#).
- The Windows Folder Browser Dialog will be opened. Select the folder where you want to have the single Python script stored



- When clicking on the "Select Folder" button of the Windows Folder Browser Dialog, the script will be generated and saved in the selected folder. Your script will have the same name as the current Flow (with the .py extension).


10.2. Run tab

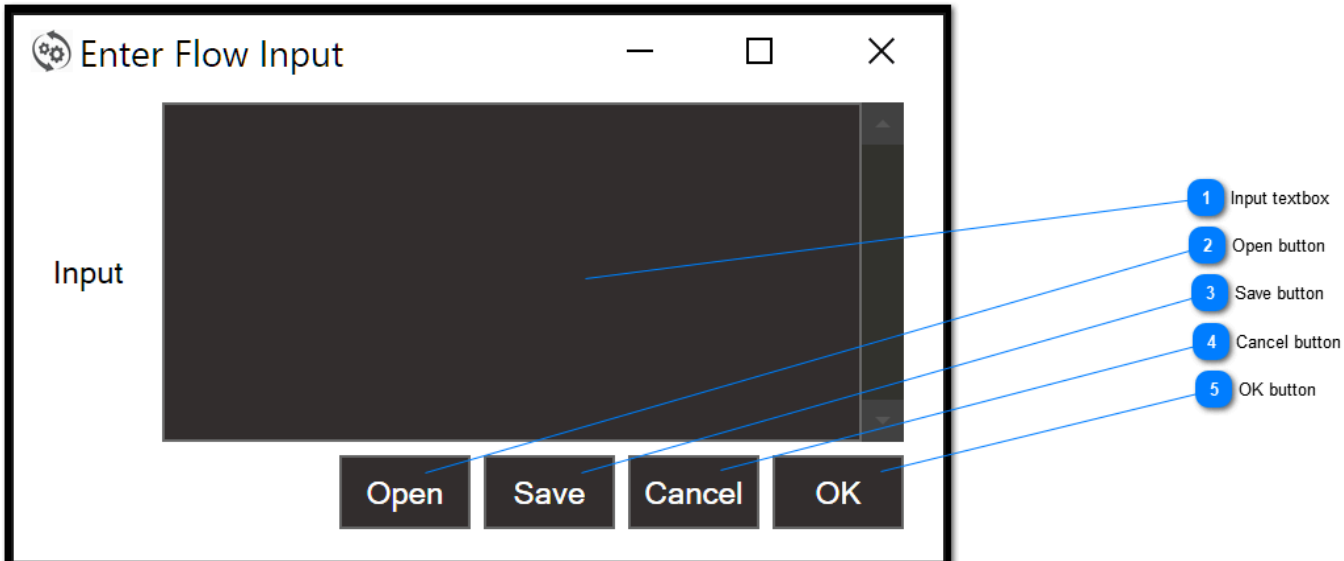


- 1 Run button**
This button will run the current flow in Python code and shows the console output in the Console window.
- 2 Stop button**
The Stop button will stop Flow execution. This may come in handy when you run a Flow that has an endless loop or when you are debugging Step-by-Step.
- 3 Input button**
This button will pop up the Input Window for entering the input for the current flow.
- 4 Debug code button**
This button will create a Python script which you can debug with your favorite debugger. The last error trace will be also visible in this script and are shown as remarks for easier debugging.
- 5 Step by step button**
With this button you can debug the flow one step at a time. The step that is executed will turn green. The sequential steps are shown in the [Console Window](#). If the Flow uses variables, the variable names are shown in the [Variables Window](#). The debugging information is shown in the [Debug Information Window](#).

10.2.1. Passing input to a flow

Passing input

You can pass input to the Flow by clicking the "Input button"  of the [Run tab](#). Clicking this button will show this window:



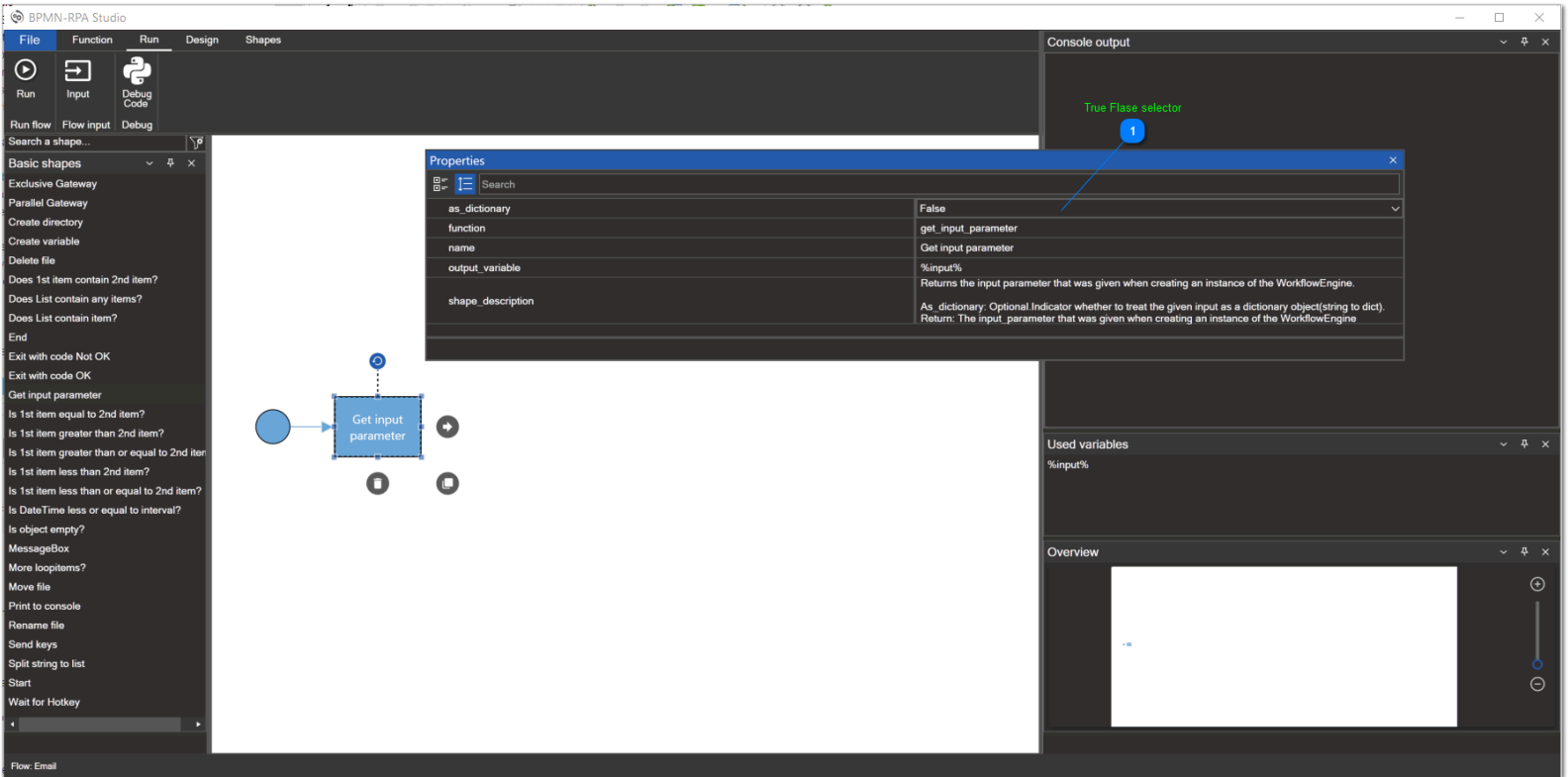
- 1 Input textbox**
The text to use as input for the current flow
- 2 Open button**
To open a saved input file (.txt file).
- 3 Save button**
To save the text of the "Input textbox" to a file for later use (.txt file).
- 4 Cancel button**
To close the Input window without remembering the text that is entered in the "Input textbox".
- 5 OK button**
To close the Input window and remembering the text that is entered in the "Input textbox".

Input Text

You can enter any text as input for the Flow. If you would like to pass a dictionary to the flow, then you can use the standard format. P.e.: {"testkey":"testvalue"}.

Get input parameter Shape

You must use the "Get Input parameter"-Shape of the Basic shapes templateset to retrieve the input value and save it to a variable the Flow can use in its next Shapes:

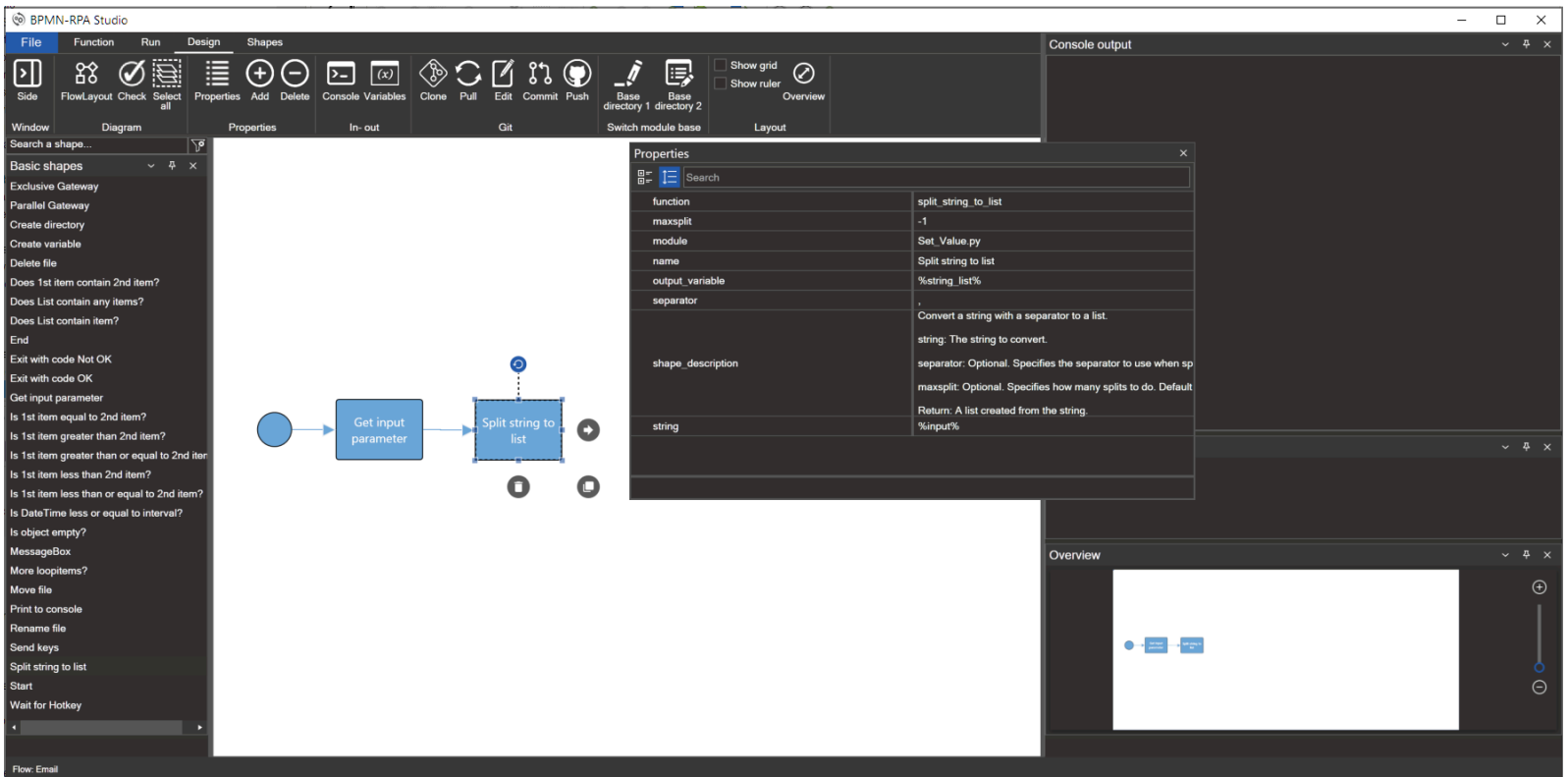


1 True False selector

Select "True" when the passed input text is a dictionary. Otherwise choose "False".

Input text to List

The "Get input parameter" Shape has no function for treating the text input as a list. If you want to process a list, then you can use the "Split string to list" Shape of the Basic Shapes templateset and pass the "%input%" variable as the string parameter:




10.2.2. Debug a Flow

BPMN-RPA Studio makes it possible to debug your Flow from within the application itself.

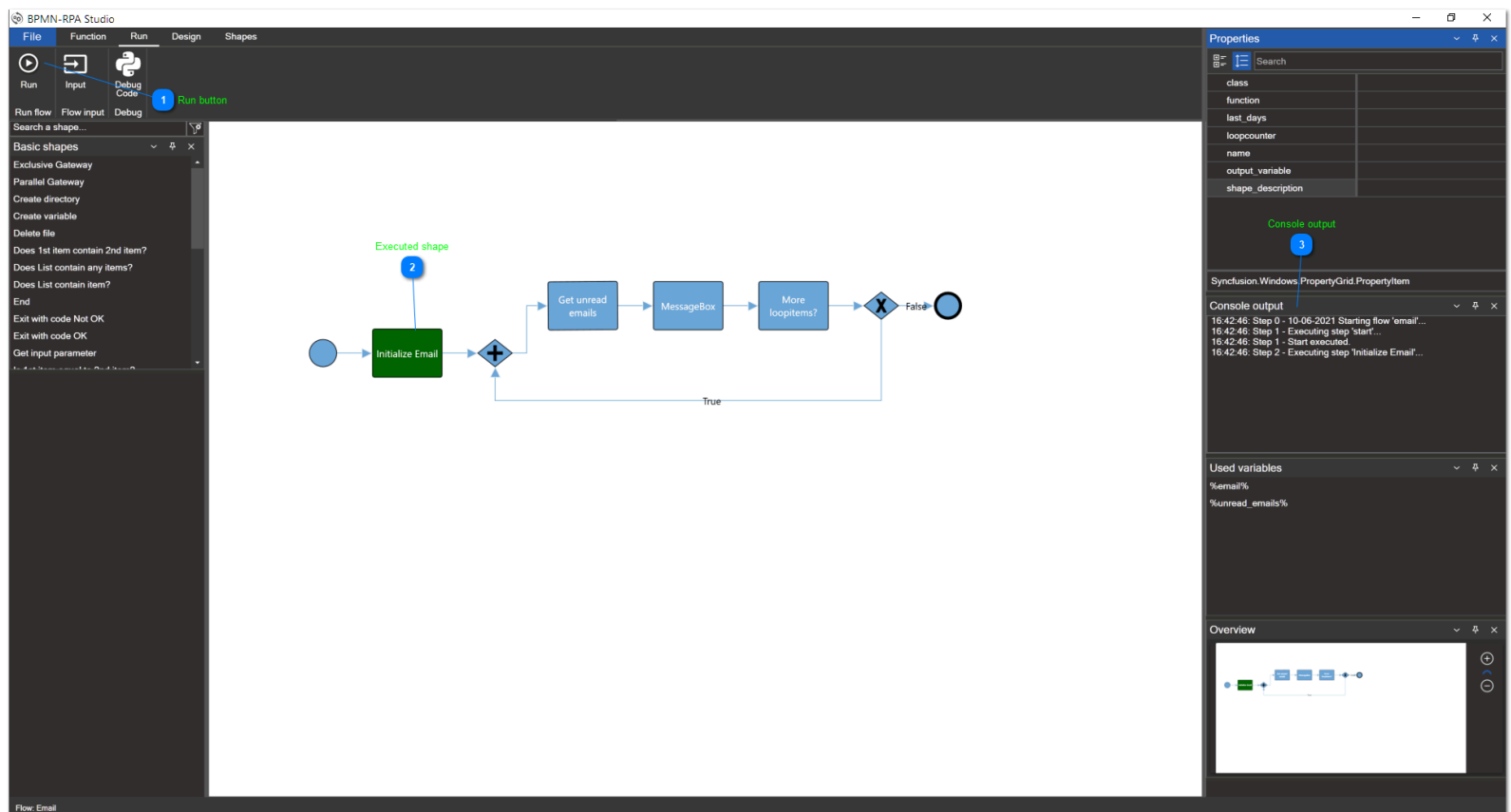
Debugging a Flow takes two steps:

1. Running the Flow
2. Debugging the code

Running the Flow

You can run the Flow by clicking the "Run" button  in the [Run tab](#). You can specify any input to the flow before running it (see "[Passing input to a Flow](#)"). Make sure you have [checked the Flow for errors](#) before starting to debug. When the Flow is executing, you can follow its progress:

- The step that is currently executed will change its color to green. After the step is successfully executed, the Shape will return blue.
- The console window will log the execution of the steps in the Flow.
- When the Flow is finished, all Shapes will be colored blue. The logged steps remain visible in the Console output window.

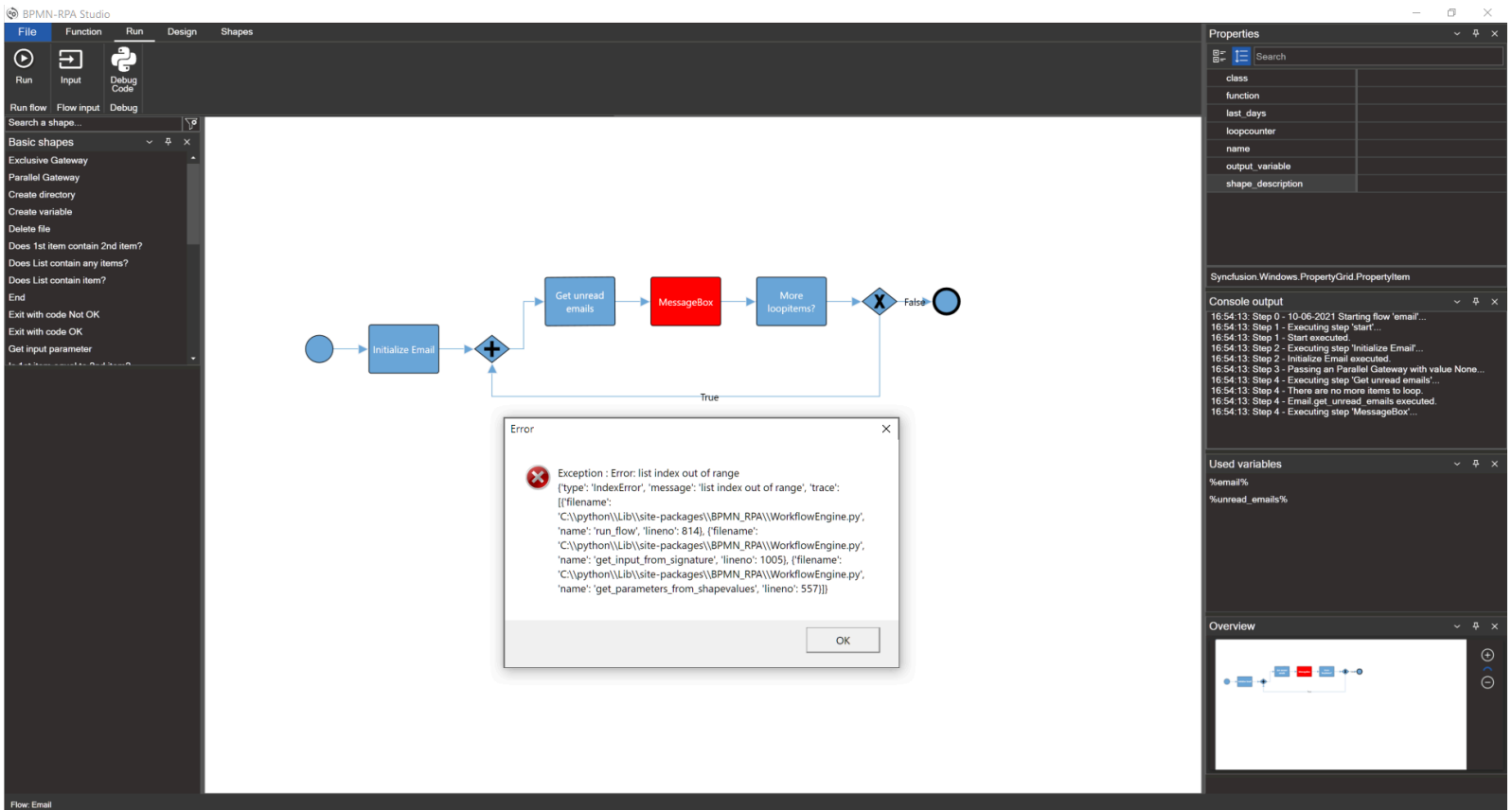


1 Run button
Button to start the Flow and run the Python code in the background.

2 Executed shape
The Shape that is executed will color green.


3 Console output
You can follow the executed steps in the Console output window.

When an error occurs, the Shape responsible for the error will color red and a message box with the error trace is shown on screen:



You do not need to take notes of the error trace shown in the messagebox. This will be taken care of by the "Debug code" option in the [Run tab](#).

Debugging the code

If you want to debug the Python code the Flow is referring to, then click the "Debug code" button  in the [Run tab](#). When you click this button, a Python file will be generated in the install directory of the program. This Python module file is always called "Debug.py". The file will be generated each time you click the "Debug code" button and will overwrite the debug.py file if it exists. After generating the python module file, the file is opened in your favorite debugger (depending on your system settings for opening .py files).

An example of the generated code:

```

1  # Debugging script for flow 'email.flw'.
2  # Please use this script to debug the Flow and Python code in your favorite debugger
3
4  # Your last Error Trace:
5  # Exception : Error: list index out of range
6  # filename: 'C:\\python\\Lib\\site-packages\\BPMN_RPA\\WorkflowEngine.py', 'name': 'run_flow', 'lineno': 814
7  # filename: 'C:\\python\\Lib\\site-packages\\BPMN_RPA\\WorkflowEngine.py', 'name': 'get_input_from_signature', 'lineno': 1005
8  # filename: 'C:\\python\\Lib\\site-packages\\BPMN_RPA\\WorkflowEngine.py', 'name': 'get_parameters_from_shapevalues', 'lineno': 557
9
10 from BPMN_RPA.WorkflowEngine import WorkflowEngine
11 import sys
12 sys.path += [r'd:\temp']
13 input_parameter = None
14 engine = WorkflowEngine(input_parameter, input_parameter)
15 doc = engine.open(r"C:\BPMN-RPA Studio\Git\email.flw")
16 steps = engine.get_flow(doc)
17 engine.run_flow(steps)
18

```

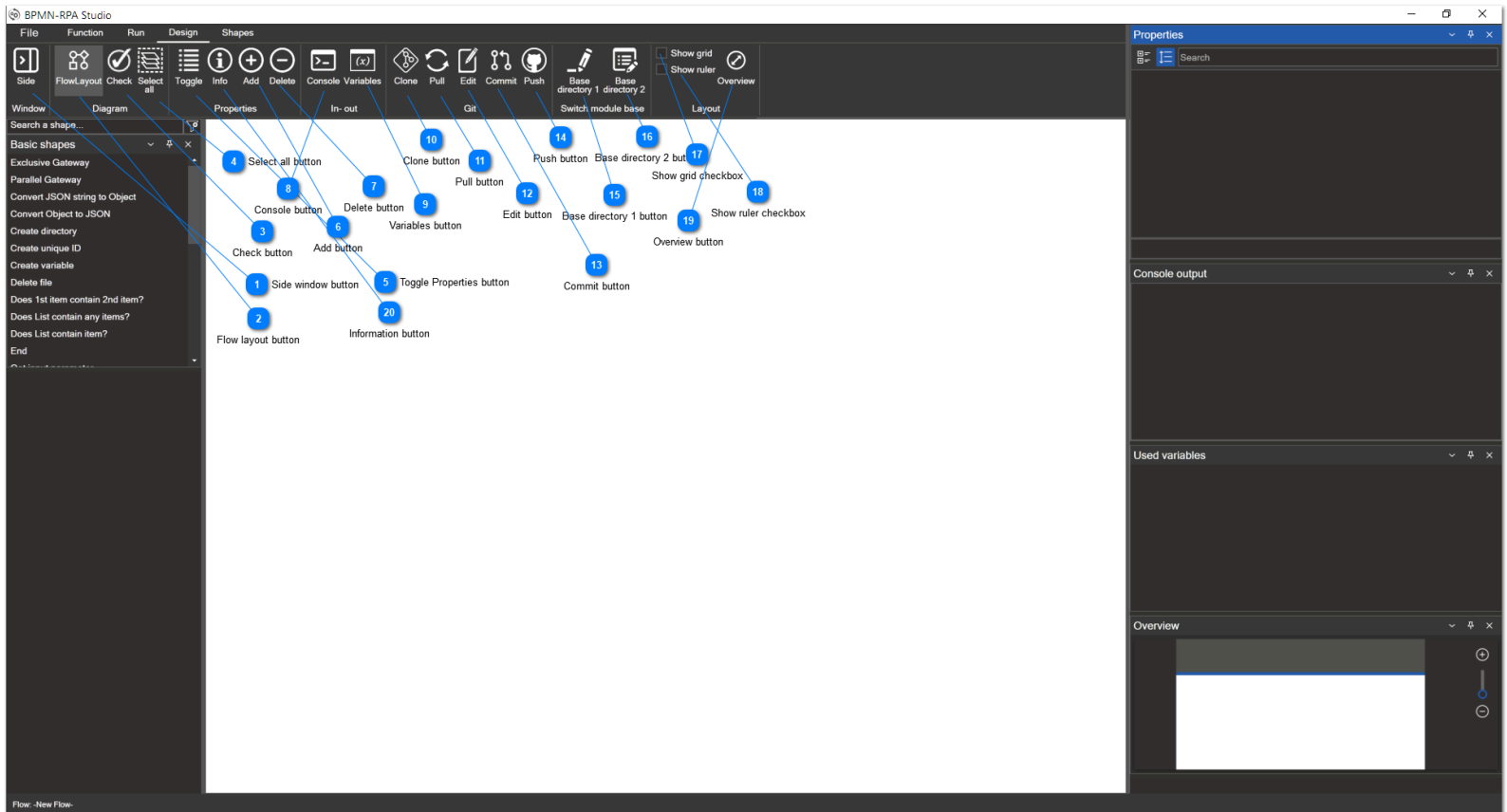
1 The Name of the flow

The script contains the name of the Flow as a remark.

- 2 Last error trace**
The script contains the last error trace as remarks, like it is shown in the error trace messagebox when running the flow. This makes it easier to trace back to the module call in the Flow that generated the error.
- 3 Additions to the Environment PATH**
If applicable, any folder references are added to the Environment PATH to make sure all code references are correct and it makes it easier to debug the code.
- 4 Path to Flow**
The script contains the full path to the saved Flow.
- 5 Run WorkflowEngine**
The script contains the code to create, load and start a WorkflowEngine (from the BPMN-RPA Python library).

You can debug the generated code and correct your Python modules if applicable.

10.3. Design tab



1 Side window button

This button will show or hide the ["Side window"](#).

2 Flow layout button

This button rearranges all Shapes of the flow into the flow-layout. All Shapes and connectors will be perfectly aligned.

3 Check button

This button will check if your flow contains any errors.

4 Select all button

This button selects all Shapes and connectors of the flow.

5 Toggle Properties button

This button will show or hide the [Properties window](#) in the side window.

6 Add button

This button will add a property to the selected Shape of the current Flow.

7 Delete button

This button will delete properties from the selected Shape of the current Flow.

8 Console button

This button will show or hide the [Console window](#) in the side window.

- 9 Variables button**
This button will show or hide the [Variables window](#) in the side window.
- 10 Clone button**
Clone a Git repository. The repository to clone is set in the "Git root url" and "Git Repository name" field of the "[Options window](#)".
- 11 Pull button**
Pull changes from a Git repository. The repository to clone is set in the "Git root url" and "Git Repository name" field of the "[Options window](#)".
- 12 Edit button**
Edit the local Git repository.
- 13 Commit button**
By clicking this button the current flow will be committed to the local repository.
- 14 Push button**
Push local Git changes to a Git repository. The repository to clone is set in the "Git root url" and "Git Repository name" field of the "[Options window](#)".
- 15 Base directory 1 button**
This button will switch all module paths to the path that is defined in the field "Modules base directory 1" of the "[Options window](#)".
- 16 Base directory 2 button**
This button will switch all module paths to the path that is defined in the field "Modules base directory 2" of the "[Options window](#)".
- 17 Show grid checkbox**
When checked, a grid will be shown on the Flow canvas to easily line out Shapes of the Flow. When the grid is visible, the Shapes will snap to the grid. When unchecked, the grid will be hidden.
- 18 Show ruler checkbox**
When checked, a horizontal and vertical ruler will be shown to assist you in outlining the Shapes of the Flow. When unchecked, the ruler will be hidden.
- 19 Overview button**
This button will show or hide the [Overview window](#) in the side window.
- 20 Information button**
This button will show a window where you can store information about the flow (like: what does the flow do? Who is the Author? etc.). The information will be saved with the Flow in the .flw file.

10.3.1. Creating a flow

 New

You can create a flow by clicking on the "New" button in the [File menu](#). The Flow canvas will be cleaned of all current shapes.

Flow convention

When creating a Flow you must make sure that:

- A Flow must always have a Start Shape
- A Flow must have an End Shape, except for never ending loops
- All Shapes must be connected by a connector
- All Shapes, except for the Parallel Gateway Shape, must only have one incoming connector
- All Shapes, except for the Exclusive- and Parallel Gateway Shapes, must only have one outgoing connector
- The Exclusive Gateway Shape always has one incoming- and two outgoing connectors. One connector must have the label/text "True" and the other the label/text "False".

Adding the Start Shape

When you create a flow, you always must start with the "Start" Shape. To assist you in quickly creating Flows, the "Start" Shape is automatically added when you double click the Shape in the [Shape library](#) that you would like to be the first step of your flow. To provide easy outlining, the horizontal position of the "Start" Shape will automatically line out to the horizontal center position of the first step of your flow.

Adding Shapes

You can add a Shape to the new Flow by:

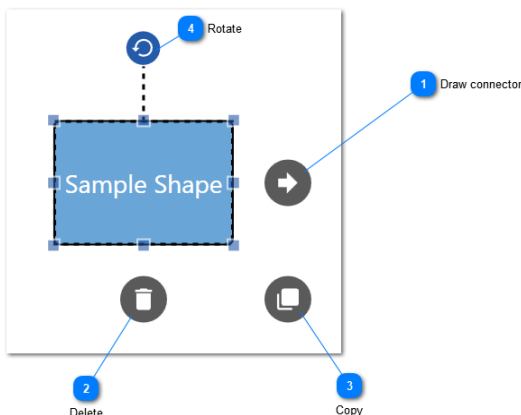
- double clicking the Shape that you would like add in the [Shape library](#).
 - Double clicking a Shape will result in a different outcome than dragging and dropping:
 - A connector is automatically added between the predecesing Shape and the Shape added
 - If the Shape is the first step of the flow, a Start Shape is added (see "Adding the Start Shape")
 - If the "Automatic Flow Layout when adding Shapes to the canvas" option in the [Options window](#) is checked, all Shapes on the Flow canvas will be automatically rearranged to the flow layout.
- dragging the Shape that you would like add from the [Shape library](#) to the canvas and drop it on the location where the Shape needs to be inserted in the Flow

Drawing connectors

To connect two Shapes in the Flow by a connector, use the "Draw connector" button of a Shape (see "Shape context menu" below). When you want to label the two outgoing connectors of an Exclusive Gateway, then double click the connector. A textbox will become visible at the center of the connector, enabling you to type the text "True" or "False".

Shape context menu

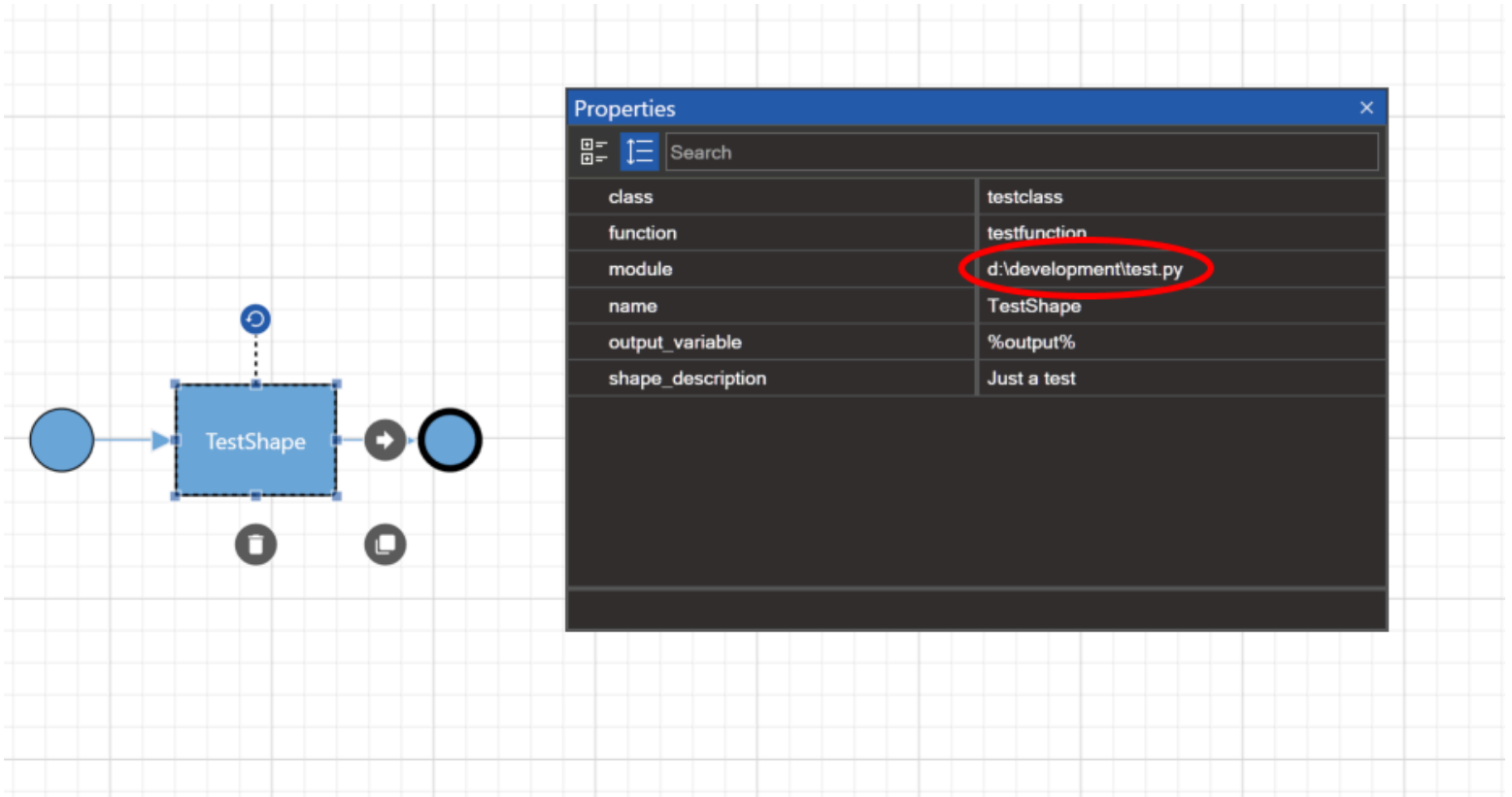
When you select a single shape, the Shape context menu becomes visible:



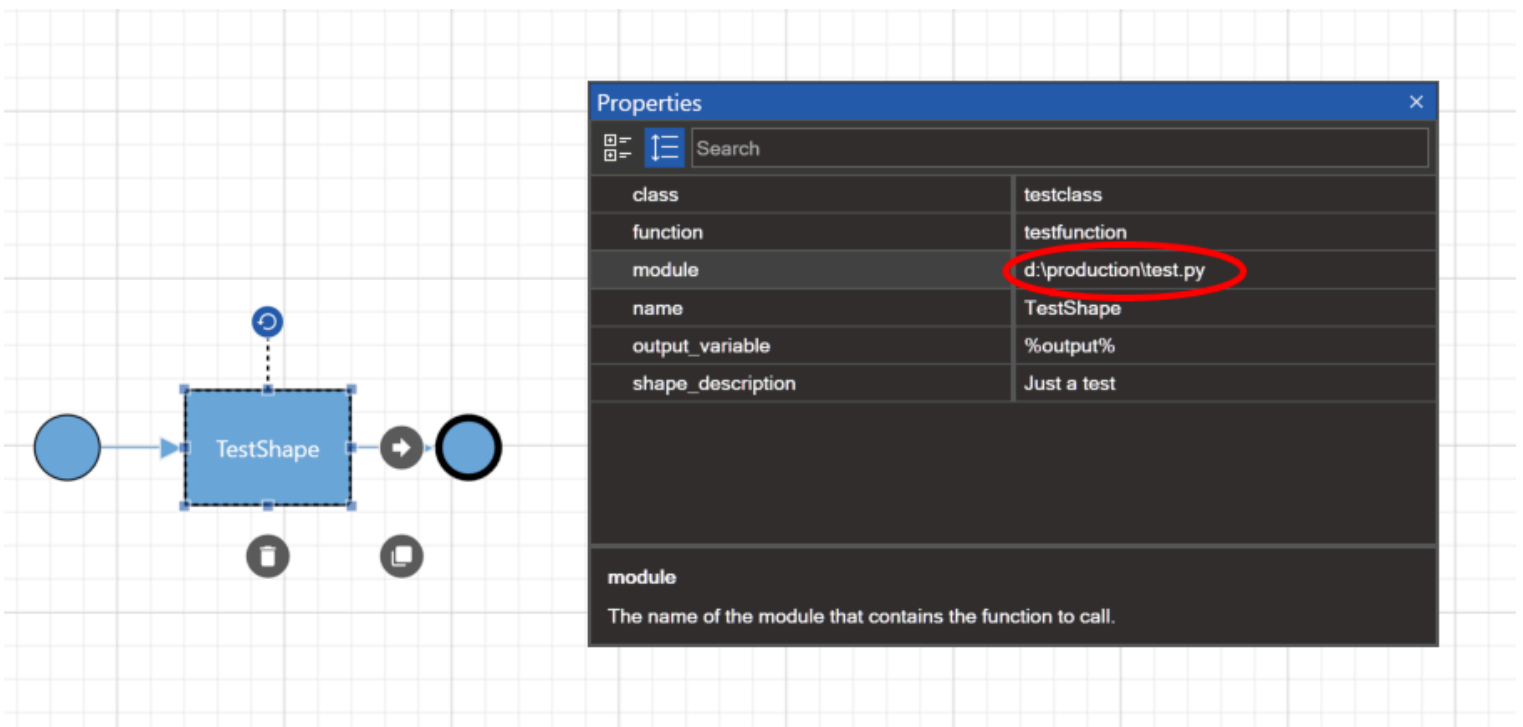
- 1 Draw connector**
Button to draw a connector between Shapes. Click this button and, while keeping the mouse button pressed down, draw the connector to the other Shape.
- 2 Delete**
Button to delete the selected Shape. There will be no warning asking you to confirm the deletion.
- 3 Copy**
Button to copy the selected Shape. When clicked, the selected Shape with all its properties will be copied. The copied Shape is automatically selected.
- 4 Rotate**
Button to rotate the Shape. Shapes within the Flow may be rotated to your liking. Click this button and, while keeping the mouse button pressed down, move your mouse to rotate the Shape.

10.3.2. Modules base directory

Sometimes when creating a Flow, you want to base your flow on Python modules that have not been released for production. It is advised to put all Python modules that still are under development and those that are in production in separate folders which can be used for Python module reference in the Shapes of the Flow. In this way you can test your Flow with the Python modules that are under development.

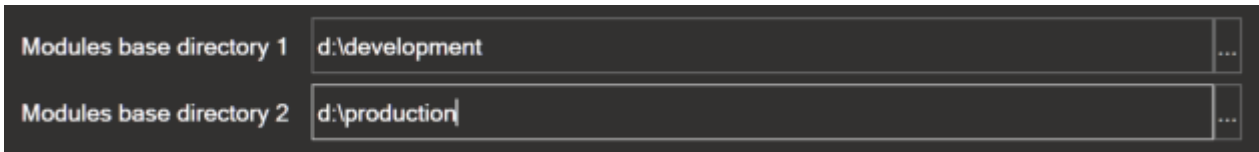


Then, after the Python modules are tested and released for production, you would want to use the same Flow you've created, but with different module references:

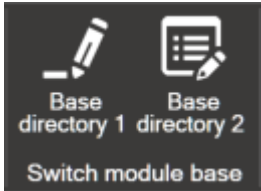


This is the reason why BPMN-RPA has two "Base module directory" settings: one for development and one for production.

You can enter your settings in the [Options window](#):



You can switch the module references of all Shapes in the Flow by clicking on one of these buttons that can be found on the [Design tab](#):



10.3.3. Checking a Flow

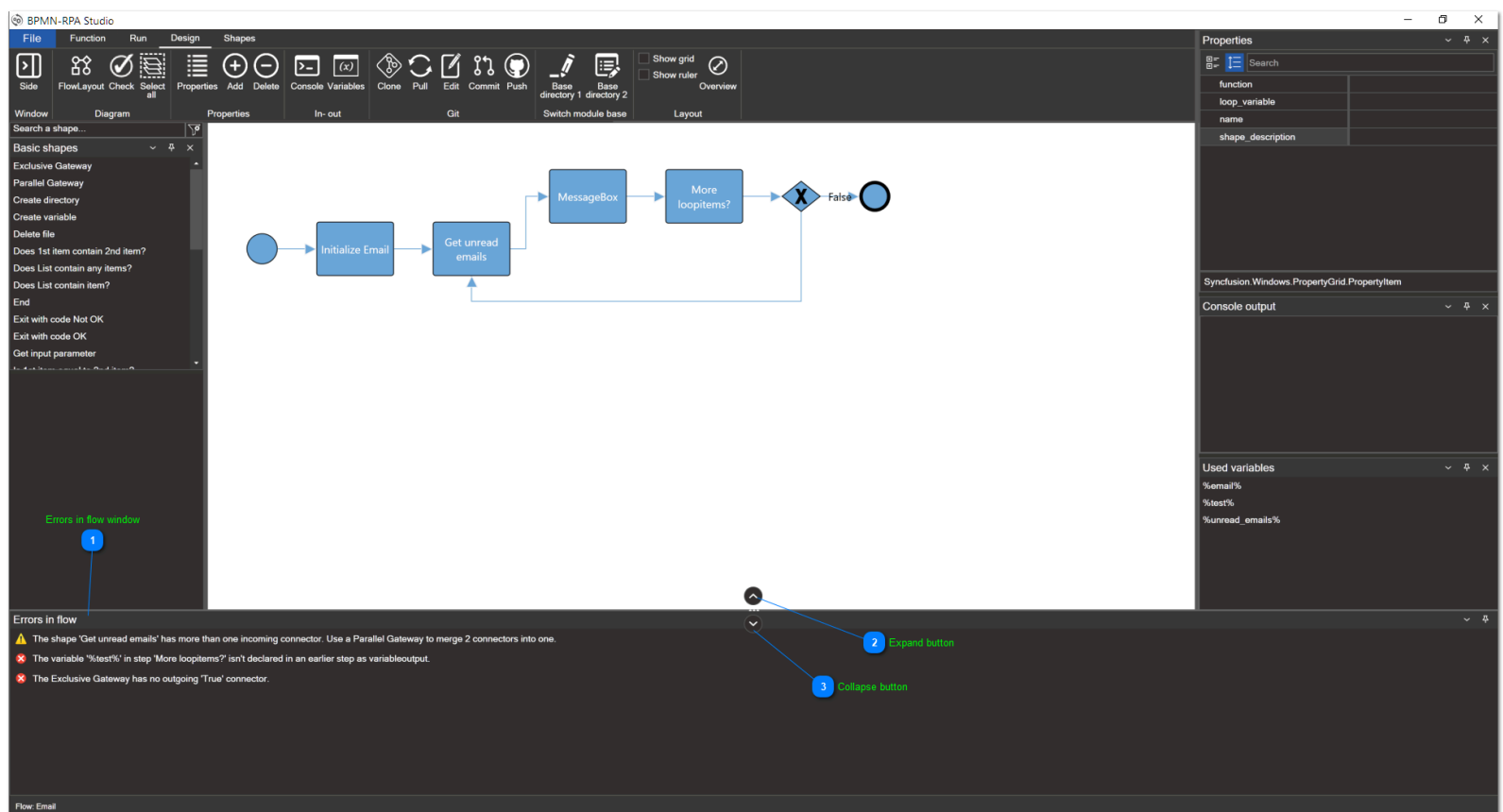
Flow convention

Flows created with BPMN-RPA Studio will be executed by the [BPMN-RPA Python library](#). The Flow will only execute when it doesn't have any errors, like:

- *Missing Start Shape*: all Flows must start with the Start shape.
- *Non connected Shapes*: all Shapes (except for Start, End, Exclusive Gateway and Parallel Gateway) must have 1 incoming and 1 outgoing connector.
- *Missing Exclusive Gateway outgoing connectors*: all Exclusive Gateway Shapes must have 1 incoming and 2 outgoing connectors. These outgoing connectors must be labeled: one with "True" and one with "False".
- *Undefined variables*: a [Variable](#) must be declared (filled with the outcome of the referenced Python module/class/function) in the "output_variable" field of a Shape before it can be used in the successive Shapes.
- *Missing Parallel Gateway*: although Flow with Shapes that have multiple incoming connectors will run without errors, it is recommended that multiple incoming connectors are merged into 1 connector by a Parallel Gateway Shape.
- *Missing End Shape*: all Flows (except for never ending loops) must have an End Shape.

Check a Flow

You can check a Flow for errors by clicking the "Check"  button in the [Design tab](#). The Flow is checked and errors found are shown in the "Errors in flow" window:



1 Errors in flow window

Window that will pop up on checking the flow and shows the errors found.

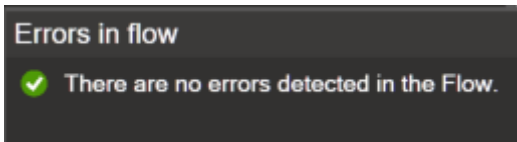
2 Expand button

Visible when hovering at the center of the error window. Button to expand the errors window (up). When collapsed, hover the center of the statusbar of the main window to make this button visible.

3 Collapse button

Visible when hovering at the center of the error window. Button to collapse the errors window (down).

When the Flow has no errors, it also will be visible in the error window:



10.3.4. Git integration

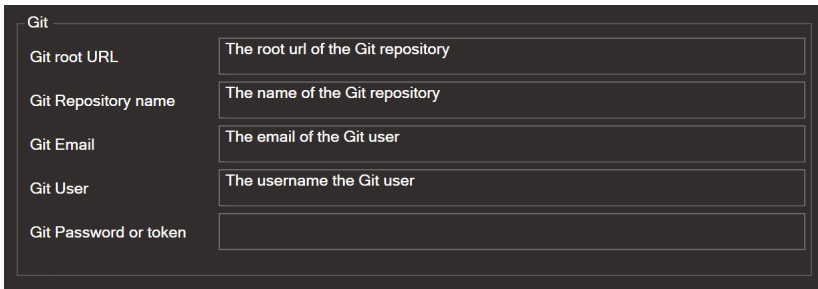
BPMN-RPA Studio has a simple Git integration. For extensive Git tasks it is advised to use a [Git client](#).

With the Git integration in BPMN-RPA Studio you can:

- Clone a remote repository repository
- Edit your local Git for BPMN-RPA Studio files (.flw, .blk, .tlp and .png files)
- Pull changes from the remote repository to your local Git.
- Push changes of your local Git to the remote repository
- Directly commit the changes of the current Flow to your local Git


Setting up Git integration

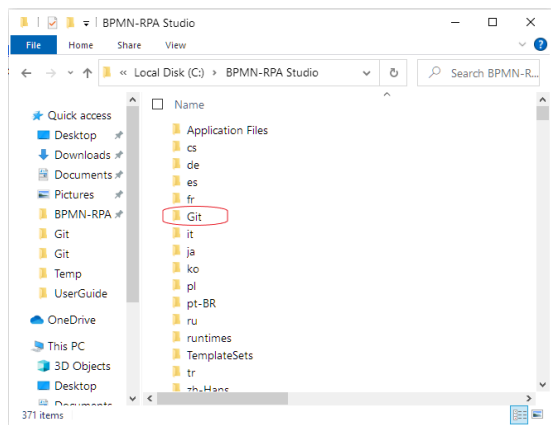
Before you can use the BPMN-RPA Studio Git integration, you must set the Git references in the "Git" fields of the [Options window](#) in the [File menu](#):



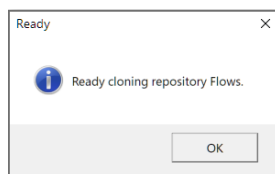
When you edit these fields, the values are automatically stored in the application.

Clone a repository

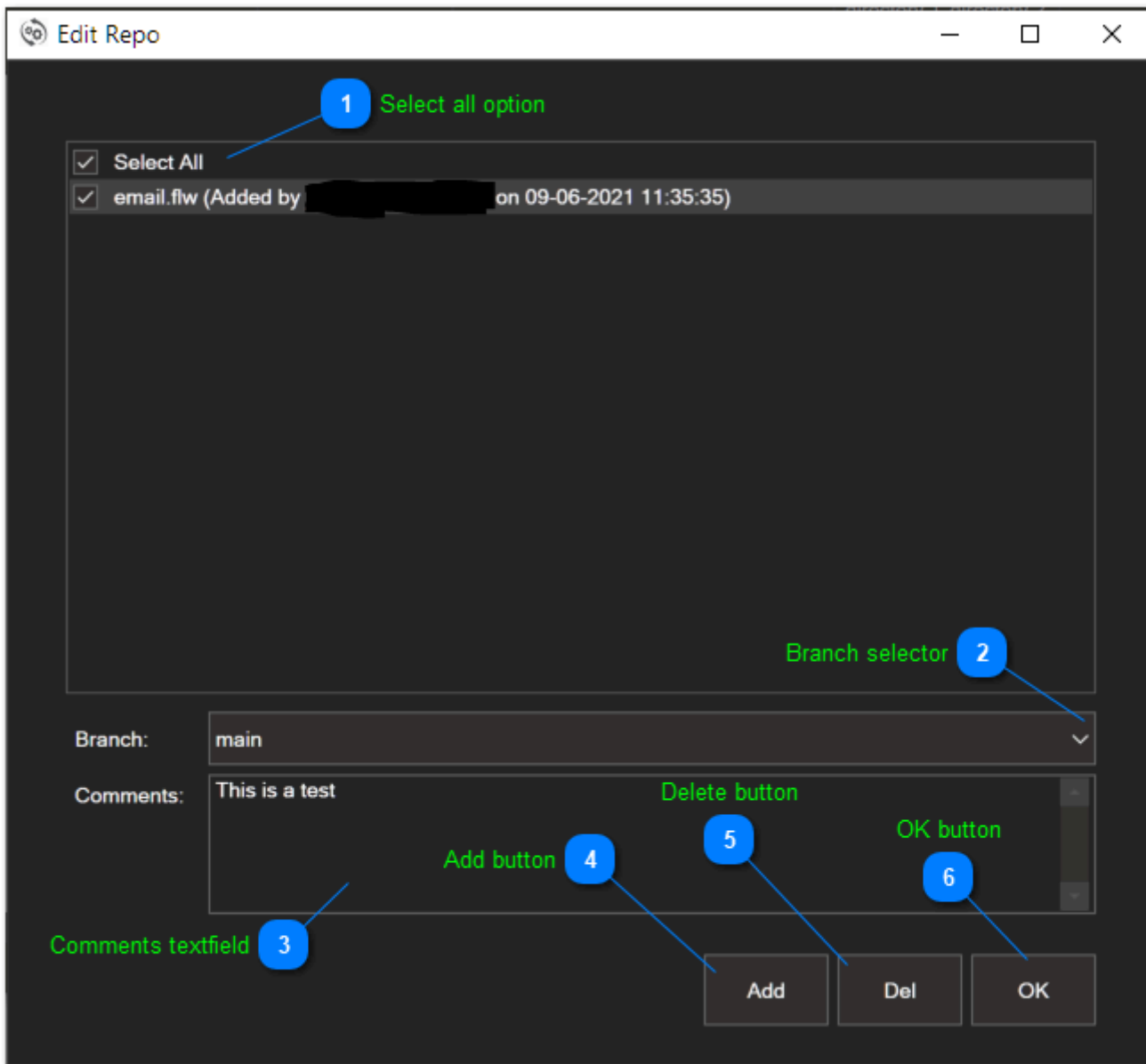
With the button "Clone"  in the [Design tab](#), you can clone (make a local copy) of a remote Git repository (p.e.: a repository on GitHub or GitLab). When cloning a remote repository, a folder named "Git" will be created in your installation folder



All files from the remote repository will be downloaded to this location. A message box will pop up to show you that cloning has finished:



Edit Git



1 Select all option

With this option you can select or deselect all repository items.

2 Branch selector

With this selector you can change the branch which is applicable. The branch information is downloaded when cloning the remote repository.

3 Comments textfield

This field will show you the comments of the editor for each selected repository item. Please notice that this field only shows the comments when one repository item is selected.

4 Add button

When clicking on the "Add" button, a Windows Folder Browser Dialog will pop up. You can browse to the location of the file you want to add to the repository. Please notice that only BPMN-RPA files can be added this way, because of the filter in the Windows Folder Browser Dialog (*.flw, *.blk, *.tpl, *.txt). Selecting a file and clicking on "Open" will bring up the "Commit remarks" window (see: "commit").


5 Delete button

When clicking on the Del button, a Windows Folder Browser Dialog will


6 OK button

Use this button to close the "Edit Repo" window.

Pull

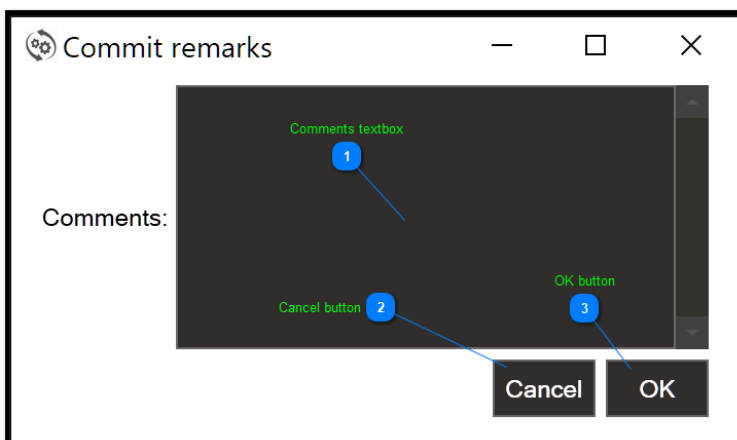
The "Pull" button  in the [Design tab](#) pulls the changes of the remote repository to the local Git.

Push

The "Push" button  in the [Design tab](#) pushes your local Git commits to the remote repository.

Commit

You can commit your Flow changes to the local Git by one click on this button. When clicking this button, the "Commit remarks" window is shown:



1 Comments textbox

This is the location where you can enter your commit comments.

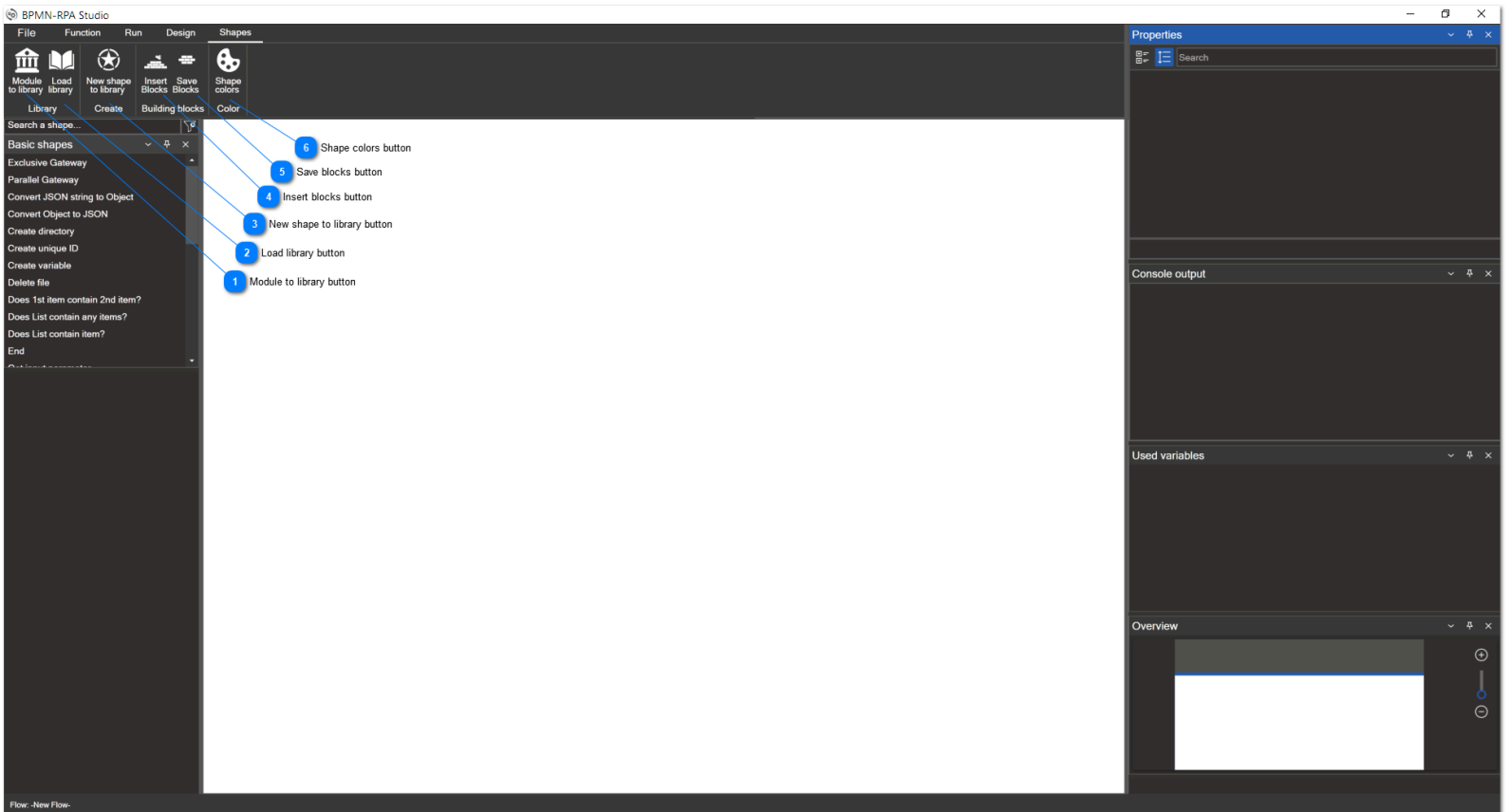
2 Cancel button

Close the "Commit remarks" window without saving your comments.

3 OK button

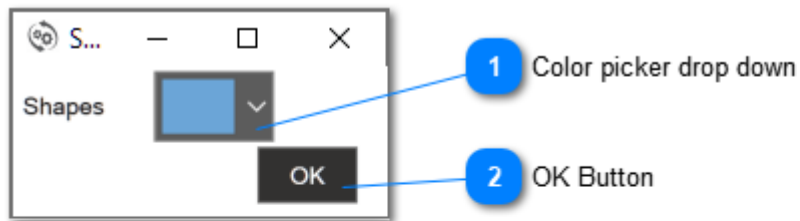
Click this button to save your comments to the commit.

10.4. Shapes tab

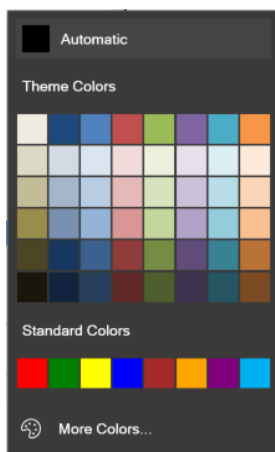


- 1 Module to library button**
This button will create a shape library from a Python module.
- 2 Load library button**
This button will load an existing shape library from file.
- 3 New shape to library button**
This button will add a shape to an existing (opened) library.
- 4 Insert blocks button**
This button will insert a saved block of shapes from a saved file into the current flow.
- 5 Save blocks button**
This button will save a selected part of the current flow as a block into a file.
- 6 Shape colors button**
With this button you can change the colors of the shapes in the Flow. The color setting will be saved with the Flow in the .flw file.

BPMN-RPA Studio



- 1 Color picker drop down**
This drop-down button will show the Colorpicker window




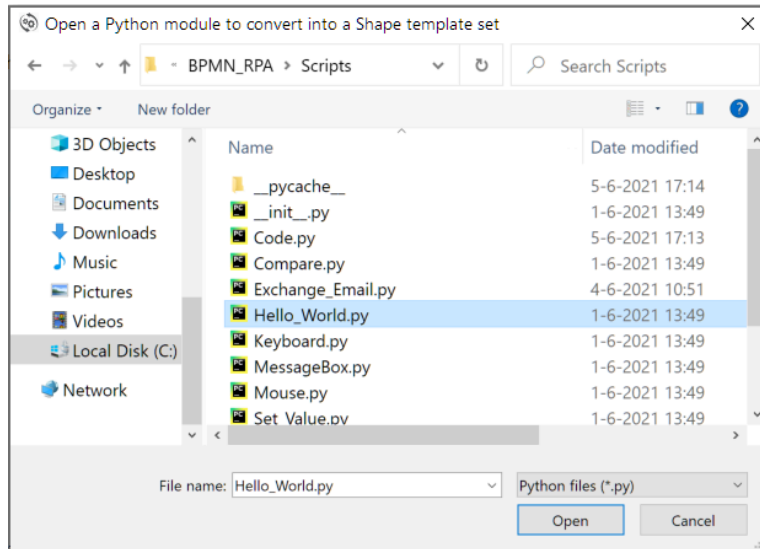
- 2 OK Button**
This button will close the color setting window and applies the chosen color to all steps of the current Flow.

10.4.1. Creating custom Shapes

BPMN-RPA Studio comes with several libraries of Shapes (see [Available libraries](#)). Of course these Shapes won't be enough for you to create your Flows. Therefore we've made it very easy to add custom libraries and add custom Shapes to them. You can add Shapes to libraries in two ways:

1. Generate a library with Shapes from a Python module file

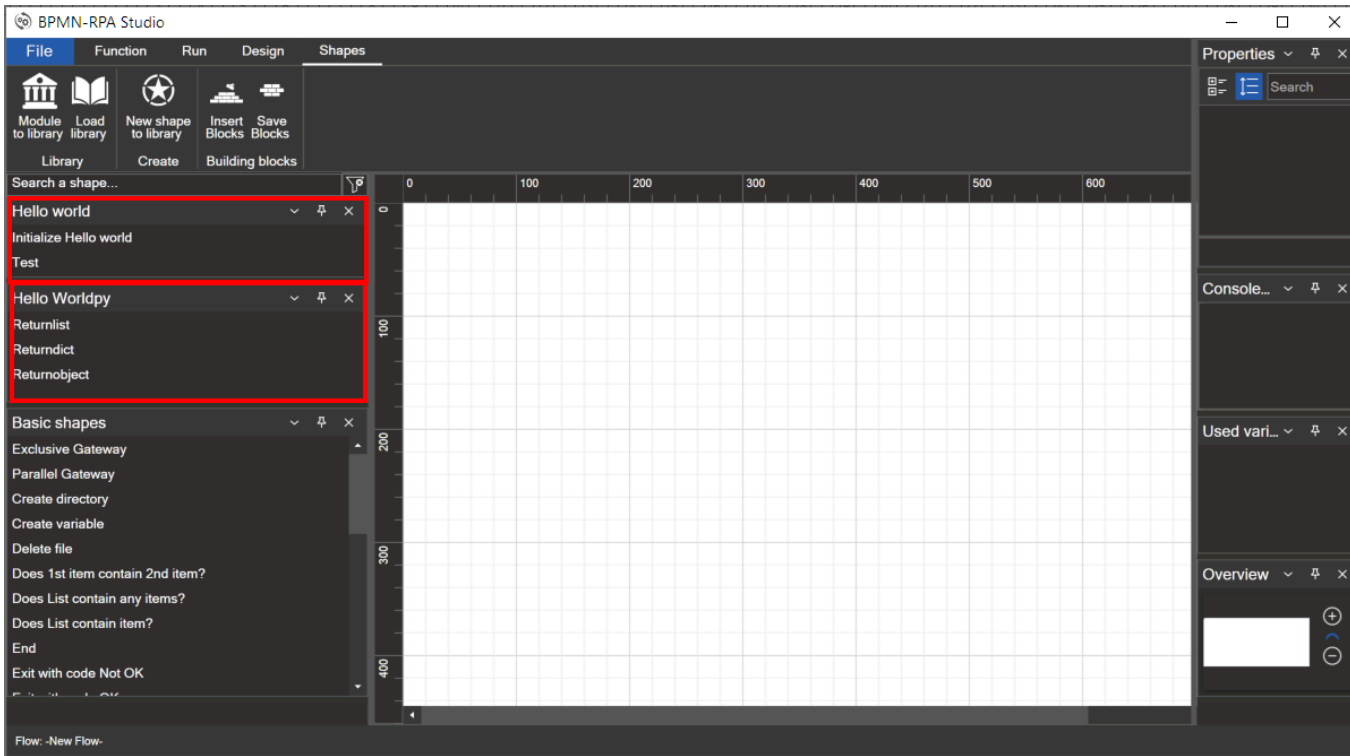
On the [Shapes tab](#) click on the "Module to library" button . This will bring up the Windows Open File Dialog:




Now browse for the Python module file for which you want to generate a Shape library, select the file and click on open (or: double click the file to open it). Wait for several seconds for the magic to happen..... and the Shape library will be created with all Shapes in it! The Shape library will be shown in the left side window.

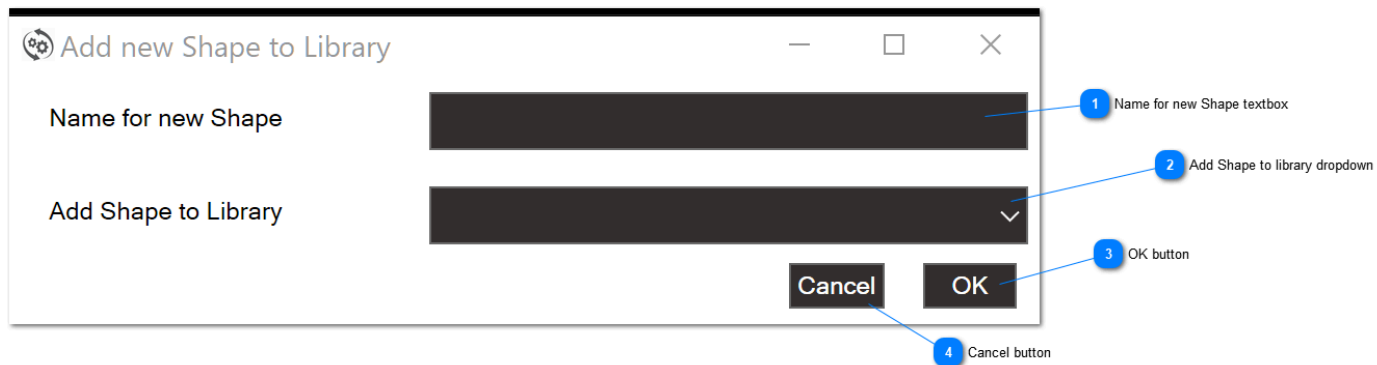
Now what is happening "under the hood"?

- The Python file is opened and all classes in the module are read (if any).
- A Shape library will be created for each class that is in the module file. Each library will be named after its class. If no classes are present in the module, then a Shape library will be created for the entire module with the same name as de module file.
- For each class (or module if there are no classes in the module) all functions will be extracted and Shapes are generated with the same name as the function. All input parameters are read and added as properties to the Shape.
- The document string of each function is extracted from the file and added to the Shape. This Shape description will provide information for each property in the [Properties window](#) (see "[Shape descriptions](#)").
- The generated Shape libraries are automatically stored in the location that is entered in the "Templates Folder" field of the [Options window](#).



2. Add a custom Shape to a library or create library with new Shape

Click on the "New Shape to Library" button  in the [Shapes tab](#). The "Add new Shape to library" window will pop up.

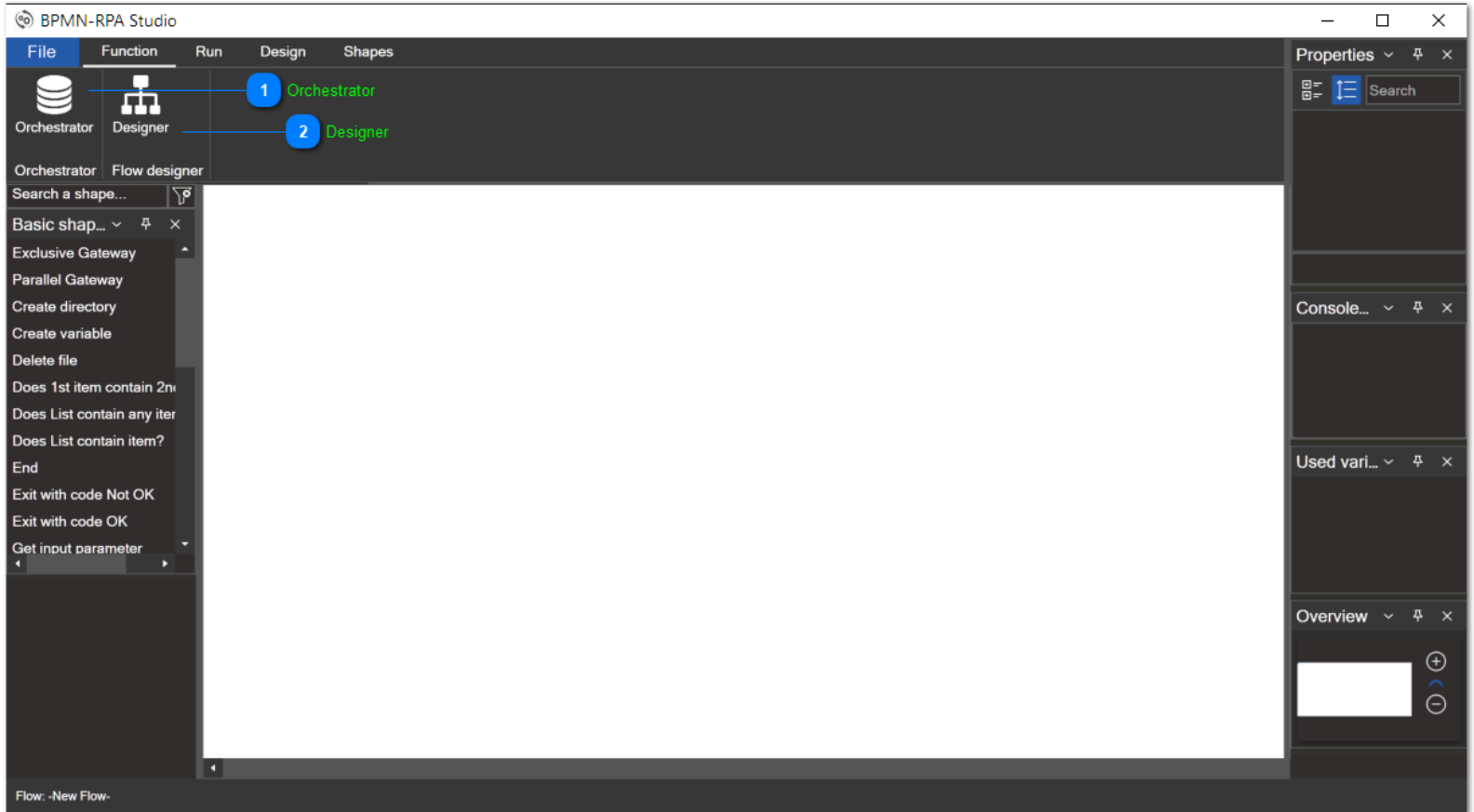


- 1 Name for new Shape textbox**
Enter the name for the new Shape in this textbox.
- 2 Add Shape to library dropdown**
Dropdown selector for choosing the currently loaded library to add the new Shape to. Besides choosing a value from the dropdown, you can also manually enter any value in this field. If a library with the same name doesn't exist, a new library with that name will be created.
- 3 OK button**
Click on this button to add the Shape to the library (or: create a library and then add the Shape to it), then close this window.
- 4 Cancel button**
Close this window and do nothing.

BPMN-RPA Studio

When you have manually added a Shape to a library (or: created a new library and added a Shape to it), the Shape isn't ready to be used in a Flow. This is because the Shape has not yet has any properties the Flow can work with. You can add or edit these properties as described in "[Edit a Shape library](#)" in chapter [Shape libraries](#).

10.5. Function tab



- 1 Orchestrator**

This button will show the Orchestrator database window. In the Orchestrator database stores all results of flows that have run on the system. When you are in the Orchestrator window, all tabs ("Run", "Design" and "Shapes") will be closed.
- 2 Designer**

This button will bring back the Designer window and tabs ("Run", "Design" and "Shapes").

11. Command line options

By using command line options you can:

1. Start the program and **opening a flow**. Usage: <BPMN-RPA Studio.exe> <name of the Flow to open>

Example:

```
C:\>"c:\BPMN-RPA Studio\BPMN-RPA Studio.exe" "c:\Temp\test.flw"
```

2. **Create an image** of a Flow and save it to disk. The following will happen:

- the program is opened
- the Flow is rendered
- the Flow is exported as an image and saved to the path given
- the program exits

Usage: <BPMN-RPA Studio.exe> <name of the Flow to open> -i <full path where to save the image of the Flow>

Example:

```
C:\>"c:\BPMN-RPA Studio\BPMN-RPA Studio.exe" "c:\temp\test.flw" -i "c:\temp\testimage.png"
```

12. Keyboard Shortcuts

You can use the following keyboard shortcuts:

- **ctrl+O**: Open a Flow
- **ctrl+S**: Save the current Flow
- **ctrl+dragging a step**: insert, reposition or delete a step in a flow (see: [Building a Flow](#))