USER GUIDE

DOCUMENTATION

OPENFLOW / OPENRPA / NODERED

v1.0.1 JUL/2020

BPA technologies
Business Process Automation

OPEN FLOW by OpenIAP

Node-RED

About this Documentation

This documentation is an effort of **BPA Technologies** (www.bpatechnologies.com), a RPA company based in São Paulo and it is currently being mantained by the same company. Thanks to **Allan Zimmermann**, creator, mantainer and founder of **OpenRPA** (www.openrpa.dk), and his cooperation in providing the necessary resources we were able to make this manual happen. We also thank **Thiago Pestitschek** for his cheerful and promising ideas for the documentation as well as assisting in proof-reading and writing this manual and **Paulo Veras** for helping to write and mantain this documentation.

Here the reader will find useful information on how to use **OpenRPA, OpenFlow and Node-RED** in order to develop useful **RPA workflows** which accomplish the execution of business processes. Although not strictly needed, the user should have some previous experience with RPA concepts and usage.

The **1st chapter** is a brief introduction on what RPA actually is and our choices for using specifically this stack.

The **2nd chapter** is an in-depth exploration of the orchestration framework used, namely OpenFlow.

The **3rd chapter** is a guided tour on the functionalities of OpenRPA - the tool where the actual automation happens.

Finally, the **4th chapter** is an overview of Node-RED functionalities in respect to RPA automation.

If you find these tools exciting and would wajt to participate on a discussion group, we strongly recommend going to rocket.openiap.io (www.rocket.openiap.io/)

In case you find any typos or misinformation please reach us at **thiago@bpatechnologies.com** or **paulo@bpatechnologies.com**. We would be glad to fix it and this way you can help us improve the quality of the material! Also, we'll happily add you to our list of contributors.

Contents

Chapter 1

Introduction to RPA

1.1 What is RPA?

RPA or Robotic Process Automation are software base solutions designed to mimic the same "manual" path taken through applications by an end user. It is typically used to automate repetitive tasks previously performed by a humans as an effective way to cut costs, eliminate keying errors, speed up processes and easily link applications together. These tasks can be successfully modeled and converted into a business process and then automated using a combination of (UI) interactions, connectors to client servers, mainframes, database, web, and others.

1.2 Tools (OpenRPA/OpenFlow/NodeRED)

There are many tools available for automating processes using RPA such as UiPath, Automation Anywhere and blueprism. Our stack of tools (``OpenRPA, OpenFlow and NodeRED``[1]) has many advantages and is obviously our suggestion for you. The reasoning for this recommendation is revealed in the next chapter, for now we will cover what each of these tools do and their outcome.

[1] - We're going to refer further to the OpenRPA, OpenFlow and NodeRED stack as OON.

1.2.1 OpenRPA

This tool is basically the main environment where the processes are mapped and designed to fulfill the inherent task at question. Think of it as an `IDE - Integrated Development Environment` for creating workflows. It is where all previously mapped activities will be implemented - such as clicking a message on a mailbox, typing content into an input form.

It is also the software you're supposed to use to export all your workflows into a repository that will manage orchestration and input. The software we'll use for that is OpenFlow, which comes next in this discussion.

1.2.2 OpenFlow

OpenFlow is an orchestrator, a centralized robot management panel and repository, which enables you to manage, invoke and configure your robots the way you need. It also serves as a central cloud repository from where you can save your robots and gather them from the OpenRPA GUI and provides an easy way to integrate many workflows with forms and other kinds of input and processing.

1.2.3 Node-RED

Node-RED is a framework for gathering APIs and online services together with an easy-of-use innovative interface. It provides an in-browser editor where you can connect flows using available nodes at its palette. Each node represents a step that when grouped together form a meaningful task. It also follows a common pattern: input, processing and output. It is noted here that Node-RED functions similarly as a middleware to an information processing system. It simply connects the inputs to the workflows and allows them to process it.

1.3 Why OpenRPA/OpenFlow/NodeRED?

Our reasons for using the OON stack are presented in-depth below.

1.3.1 Low code concept

Low code concept means there is no technical or programing knowledge required to perform a desired task, i.e. you can simply just drag a box and input a few parameters for it to become fully functional.

The term `low code` refers to platforms that resort mainly to GUI-focused interfaces which assist users in the development of a BPM without necessarily requiring basic programming knowledge. These applications use heavy visual modeling in favor of coding to aid in the process of designing and implementing processes.

Low code development platforms have a very gentle learning curve and are often designed with UX-oriented practices in mind. It is a fact that it also helps the user to advance rapidly and design and deploy working processes with few caveats.

[2] - From now on, we're going to refer to low code development platforms as LCDPs.

1.3.2 Time to market

The length of time between a product being conceived until it's completion and deployment is significantly higher using a common application development process in comparison to LCDP. This occurs because there are far less steps involved in this cycle when LCDP is used. As an example, code-related unit tests are not needed, since all the parts of the software used to build the application are already tested and integrated within the framework.

In our stack, this also means it's easy to prototype and creating something actually functional - due to LCDP programming - and the migration from prototype to an actual production product can be easily achieved. This is provided by the OON stack framework native integration and easiness of use.

1.3.3 Framework vs "extra work"

The OON stack significantly minimizes the amount of work needed to be done to implement a final product. This is due to the many features embedded within it, such as privilege and access management, embedded MongoDB database, safe credentials and form. Which is now discussed.

Privilege and Access Management

OpenFlow provides a readily integrated privilege and access management interface which is further discussed in the **Roles** chapter in the OpenFlow's section. It makes possible for the administrator to create users and assign roles to these users which limit - or expand - the features they can access. There is no need to configure an user for every workflow and set the permissions accordingly, you only need to create a role and assign their given permissions.

Embedded MongoDB Database

The OON stack has a single database which they use to feed their data to and from, MongoDB (https://www.mongodb.com/what-is-mongodb). Integration is easy and the data is persisted in JSON.

The fact that there is a central database to the stack provides an easy way of creating **KPI - Key Performance Indicators**, reports and dashboards, graphs, and records of the data processed by the agents. It also measures the effectiveness of robots and their accuracy but remember you can only manage what can be measured.

You can also save files in the cloud and use them as shared files - such as a form presented in a Word file. It also has methods for saving files for future usage - as an example, for tasks that take too long or tasks that are fragmented between many steps/workflows. This provides a centralized way of handling file management.

Safe Credentials

Our safe credentials management system provides an easy way, to share, insert, update and delete credentials between systems and robots using the `SecureString` class[1]. Everything is automatically managed, so if you need to store a specific credential used by a workflow, you do not need to worry about the security of the storage system.

[1] - Secure String class (https://docs.microsoft.com/en-us/dotnet/api/system.security.securestring?view=netcore-3.1).

Form

OpenFlow also provides forms, a user-friendly way of sending input to a workflow by creating dynamic webpage. It gathers all the input a user should give so the robot can use it for its processing and execution. This means you do not need to create a webpage just to gather data from the robot but you can create a form and make it public - or give its acess only to a few roles - so the end user can enter the input.

Message Queuing

Using Node-RED, a MQTT framework - more on this in the Node-RED section, the stack is able to automatically queue messages and tasks handled by the OON stack. There is also a shared workload amongst robots that run the same workflow, causing the orchestration process to be much easier and faster.

Built-in Functions

Many universal RPA activities are already shipped built-in inside OpenRPA - ready to use. For example, reading an excel file, opening a file, browsing a webpage.

1.3.4 How the tools integrate

The development process flows through OpenRPA -> OpenFlow -> Node-RED.

In **OpenRPA** you design the workflows which will execute the desired RPA process.

In **OpenFlow**, you manage them and also create forms for external input for the user.

In **Node-RED**, you wire both of them together in a way similar to building a flowchart.

> Think of it as the **OpenRPA** workflow represent the players in a concert, **OpenFlow** is the maestro, or conductor, and **Node-RED** is the sheet music.

1.3.5 Scalability

Scaling products with the OON stack is much easier than usual. Given that orchestration is directly managed by OpenFlow and assisted by Node-RED. It is also much easier to deploy, verify and provide maintenance to robots.

When you deploy a new change on any of the three applications (OpenFlow, OpenRPA or Node-RED) they are instantly applied to the whole environment - online robots will fetch the changes automatically and offline robots will do it the next time they get online - worry free. This is due to the main repository from which the robots gather their workflows from.

OpenFlow can handle a limitless number of robots thanks to RabbitMQ[1]. The required technical specs for the server running OpenFlow/Node-RED it are intensely low.

OpenRPA agents can scale up by installing OpenRPA on different physical or virtual machines, or by using HDRobots extension where a windows terminal machine can run many users/instances simultaneously. Now, if a given process requires no GUI - in other words, all processing can be done without the need of keyboard, mouse or image recognition, a single instance of OpenRPA may run many workflows in parallel.

[1] - RabbitMQ (https://www.rabbitmq.com/).

1.3.6 Security/No exposed code

No code is exposed to the end-user or agents, since there is rarely any code written (due to LCDPs) and viewing of the code itself can be disabled if needed.

The credentials are also very safely stored, there is one for each robot, user, machine and Node-RED. Authentication is done through SAML using your OpenFlow credentials. Each credential has different levels of permissions and each password is encrypted as well using cryptography ciphers, which makes it even harder for someone to attempt an authentication attack.

Chapter 2

OpenFlow

2.1 What is OpenFlow?

OpenFlow is an application, which enables the user to manage, invoke and configure your robots through workflow entries. It also serves as a central cloud repository from where it is available to save your robots and gather them from the OpenRPA GUI and provides an easy way to integrate many workflows with forms and other kinds of input and processing.

2.2 Installing, First Run

The following guide's purpose is to help the user install an instance of OpenFlow **locally**. If you already have an URL provided in which to use OpenFlow, please refer to the next section.

The user must make sure he already has Visual Studio Code (`https://code.visualstudio.com/download`), NodeJS (`https://nodejs.org/en/`) - preferably using nvm (`https://github.com/nvm-sh/nvm`), MongoDB Community Server (`https://www.mongodb.com/try/download/community`) and RabbitMQ (`https://www.rabbitmq.com/download.html`).

2.2.1 Quickstart Running OpenFlow using NPM

In this section lies a brief and concise description of how to install OpenFlow through NPM.

Then, the user must run the following command from a command prompt with escalated privileges in Windows 10.

```
npm install --global windows-built-tools
```

If using any GNU/Linux distro which uses the `APT` package manager, the following command replaces the one above.

```
sudo apt-get install gcc g++ make
```

Then, the user must install the `openiap` package.

```
npm i openiap -g
```

Then he shall create a template configuration file.

```
openflow-cli --init
```

Now the user can install the service.

```
openflow-cli --install openflow
```

All the user has to do now to access OpenFlow is visit the OpenFlow page (`http://localhost.openrpa.dk`).

2.2.2 Developer Setup

In this section lies a brief and concise description of how to install OpenFlow by cloning the OpenFlow's GitHub Repository (`https://github.com/open-rpa/openflow`).

First, the user must clone the repository into a folder.

```
git clone https://github.com/open-rpa/openflow
```

Then, the user must initialize a command prompt and move to the desired directory.

```
cd openflow
```

Now the user must install `gulpy` and `typescript` globally.

```
npm i gulp typescript browserify tsify -gather
```

And also install packages for OpenFlow API/Web.

```
npm i
```

The user must now install the packages for Node-RED by using the following command.

```
cd openflow_node_red_configure_form
npm i
cd ..
```

Now the user will open the working folder in VS Code.

```
code OpenFlow.code-workspace
```

And create a folder at the top level of the OpenFlow folder called `config` and inside it a file called `.env` - be careful since the name of the file has a dot starting it!

Lastly, he shall add the following content to the `.env` file:

```
nodered_id=nodered1
nodered_sa=nodered1
port=80
nodered_port=1880
nodered_domain_schema=$nodered_id$.localhost.openrpa.dk
tls_crt=
tls_key=
tls_ca=

signing_crt=LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURqRENDQW5TZ0F3SUJBZ0lKQUp5N0tI
MTE1dkQ4TUEwR0NTcUdTSWIzRFFFQkN3VUFNRnN4Q3pBSkJnTlYKQkFZVEFrUkxNUk13RVFZRFZRUUlEQXBUY
```

(continues on next page)

(continued from previous page)

```
jIxbExWTjBZWFJsTVNFd0h3WURWUVFLREJoSmJuUmxjbTVsZENCWAphV1JuYVhSeklGQjBlU0JNZEdReEZEQV
NCZ05WQkFNTUMzTnBaMjVwYm1kalpYSjBNQjRYRFRFNU1EUXdOekUwCk5ETXpORm9YRFRJNU1EUXdOREUwTkR
Nek5Gb3dXekVMTUFrR0ExVUVCaE1DUkVzeEV6QVJCZ05WQkFnTUNsTnYKYldVdFUzUmhkR1V4SVRBZkJnTlZC
QW9NR0VsdWRHVnlibVYwSUZkcFpHZHBkSE1nVUhSNUlFeDBaREVVTUJJRwpBMVVFQXd3TGMybG5ibWx1WjJObG
NuUXdnZ0VpTUEwR0NTcUdTSWIzRFFFQkFRVUFBNElCRHdBd2dnRUtBb0lCCkFRQy9JSkdEaGxLTU9SWkoycXQ
wSWpjSDZOWUFtZDVxQzQ4dkNJRE54QWZCbmQxQnN4WlVjWkl5dkFlT28yNDcKM3I0eTYwNDgxRHVUS2JaMTBTN
jRqRU05aW1XTXB1TFlJRnVyQ3BWNzVEWWhxMS85Q0FJVHJqNjlmVDluSkptcwpjM2lxTnJ1Tlg1bDlISXdadWt
QM1ZNRkJRNWZVd3N1ZnE0YW1NbnVnZmtyUEVzSngxK3VJb0NYU3pyblZvcnZpClZ0ZFh4a3M4N0l1S0ZnMDJIZ
1RQSzdwc0FXYTBRY3g2ck04bkV5TUhwNUdlR1Rvb1NNbkcyZ1RGNWZOSVFNdTMKVEVoc2p3SWRTYmRwck1Gb1V
ZV05Bc2FueTJOQk0wREhZRUdjQlZhZ0xWNUhFUW5ySUM3NEhtNjYxdG9HaU5VSAoveW04U3VndTgwWVFiVGxPc
TFWNnNkaVRBZ01CQUFHalV6QlJNQjBHQTFVZERnUVdCQlMrYWc1NGtITllKZ29pCm9yRnlia293THR5R3ZqQWZ
CZ05WSFNNRUdEQVdnQlMrYWc1NGtITllKZ29pb3JGeWJrb3dMdHlHdmpBUEJnTlYKSFJNQkFmOEVCVEFEQVFIL
01BMEdDU3FHU0liM0RRRUJDd1VBQTRJQkFRQTZpZHBYdzdSR2pzcUpNanI2c1YvaQpXeXFsL2xkSy9sa1NCdnN
BSENieDdQYi9rVUd2NHJJNbndYMnBHdTR0YkFnSDc4eStmS3dzazllYkxDeTA4Y1k0Ckt5czhzbUpLenhWN0R6U
3RVR1NvZmZMaFliVVVMK3UyNU5vVXc0TG1WQU5FU0NaMTZ3aTdPQUdJMkJnNFR6TXoKdnlIUHRaaE9wTXBNV2lz
M2ZnRXFzV3QxS2VLcXo0Z2M5RnJtZDZPNlQzVVAxWTRBR3VEWnNScnpiU2RQS2JxbApxekprT2tQcGtHOGo3ZjF
WNkk1ZlkzblZaSWk2YW1TcTM1RTJkQzVMY0dIQXRtT1lWL0c4TEQ3OUFnTVpFUU5vCkF2R2RnV1gvbXBFVkFjM
mRFQkJlcUN6WjF1aVhWUjdERld2ZDFKTEpsdVRyTm9jMUROc0xNKzNEZFFiQ2JnYVcKLS0tLS1FTkQgQ0VSVElG
SUNBVEUtLS0tLQo=
singing_
↪key=LS0tLS1CRUdJTiBQUklWQVRFIEtFWS0tLS0tCk1JSUV2UUlCQURBTkJna3Foa2lHOXcwQkFRRUZB
QVNDQktjd2dnU2pBZ0VBQW9JQkFRQy9JSkdEaGxLTU9SWkoKMnF0MElqY0g2TllBbWQ1cUM0OHZDSUROeEFmQm
5kMUJzeFpVY1pJeXZBZU9vMjQ3M3I0eTYwNDgxRHVUS2JaMQowUzY0akVNOWltV01wdUxZSUZ1ckNwVjc1RF
locTEvOUNBSVRyajY5ZlQ5bkpKbXNjM2lxTnJ1Tlg1bDlISXdaCnVrUDNWTUZCUTVmVXdzdWZxNGFtTW51Z2
ZrclBFc0p4MSt1SW9DWFN6cm5Wb3J2aVZ0ZFh4a3M4N0l1S0ZnMDIKSGdUUEs3cHNBV2EwUWN4NnJNOG5FeU
1IcDVHZUdUb29TTW5HMmdURjVmTklRTXUzVEVoc2p3SWRTYmRwck1GbwpVWVdOQXNhbnkyTkJNMERIWUVHY0
JWYWdMVjVIRVFucklDNzRIbTY2MXRvR2lOVUgveW04U3VndTgwWVFiVGxPCnExVjZzZGlUQWdNQkFBRUNnZ0V
BVXBjZ1NsV2hGamNWQ3BVVHdmdUhERVB4TmhGSHEwdVRkQitZaVZKTWg3NVAKL2pRRlVqaEJsT3JyMlJlR2F4
aTEyQXNXby9LU1MrV2Frdzd4d1kzYkFKenRoUG9Zekl3dkVKcGlQa2MvblEwUgpUYVpJUDNqc1k3WGIwQlpn
MGNTVVAvbW0wbENkWXhNUzk0c21FNXJzWitkdGxPTVlXc2NrU0cxSVB2SlVJV2FZCnl3NC9kaHJ3TWRsREVZS
2tSbks1aDR1dXR1dzA1Q1VzNkZWN2F5cEJRRStGM3RxVlF3QWxGbWNueXdvZTB5WjQKZW1tWkRvT1dzNUk4cG
NGbjZCSW1wZjN3UEg5UWhlQXJVaXRqV3YrZmI2cWRVaHJFVDBxMzh4dTZ5M1lJNFNLYQpXME9kUng4L3FTYll
XdkpzbmxscDR0aUpDWE5IdnV6MVBKSGhxOUprQVFLQmdRRCt2dHlWcVJoaEJZTmd6WWorClFCCaDNLcktrNEVqT
XZaSGhEelhES0pMZ2JEVmJzNExrQkYySWZGOTF0S1k3d09CNVpxXcU82Z3FqVVJBVE5hc1YKOExCOGE2TEpXYVJ
uTklLMnJkd1FwalFYcy92OVBSYnJwc2tTbDRJdUsyZWNBMjBSQkhicW5yNHZ5ZkQ4U3BzaApSdHlTUk5CRGVsaU
01Z1JDM0JKKzBZbjBVUUtCZ1FEQUVZSUp3Y2xnQloyWmNNTlh1dW1lLzUxdHBHeGppRTJpCjZ3SDNsOHNTVDN1U
1A4MHdndGdHaVFsUTRsR3BmSThqWkl4N0pwcGw0Qko1ZEtuVnpIS1dqMzA3YXYxcjdpU3QKLzJOVDNobzdaYkNl
YzlhMHlJU2E3dTNGZGxzZ0VPcE45dURmbG5GQVQ3ZmIrM2d4Sk9DUWp1TkFCZXZaK2pScwpZY0ZhQWhGNW93S0J
nUUNGUG9HVVNsRDFGblFrWXYwL3QzalVnK0hTK1hrNmxnRkNqYmthTGhPOURQeFB6Ykl0CjM5YW9lQjFhTExZeV
ZPMVVzZVl0Z0Y4MkUwVnNOc3NZKzc3a0pVeU5NclVhUWs0SWpTR3BGN1h4bS9PMi9vZ0oKbEVCaDJCdUFXTFdsM
WVqcldNRjJjTGVidVcyeUdMZlJqUVg3LzhCTE95Z3I4bmZTSE5nVHV6Z0VNUUtCZ0JrZgpNUjhObGNWVmRyT25L
Q1hGY09FM0ZlUk5hVS9yZUJ3akdQTEZpKzR0TDBDRno5VFVpR1R5YjZHQXVLV3VnUnBrCkFHdnJOSzYyakRRT3F
sZ29rYVJYeUUySlJQUmxCYThzaEZWbjY0NXhVcFNuR2lJelNBVHIwM1hNY1ViVWI1RWIKQlhhNU9yN3FybVc3a3
BENi9kUnFuQmEzcjQyblNFd1V6VEYwcTh4NUFvR0FIcXdRSyt1R0NTdlNsNENJUGhyRQpDREIvcytDK2NJNXVCe
FJCMHVlNjc3L2lpdGZPSU9lOUNiTHE3R0tib0w4RVg3eXhKNVRLWjlYQmh5LzNCWmVNCllydEx3M2JicTNTN2hp
UGFYSmE1dXZma3BWR1RnNEdzTnBJQ3VNTEJUaXJ6M0ZRV25UNFNZbzkrREVoalhEeVQKWlVOMERtUkJVNjNjWjR
SUlXd2xWUTA9Ci0tLS0tRU5EIFBSSVZBVEUgS0VZLS0tLS0K
NODE_ENV=development

saml_federation_metadata=http://localhost.openrpa.dk/issue/FederationMetadata/2007-06/
↪FederationMetadata.xml
saml_issuer=uri:localhost.openrpa.dk
# saml_entrypoint=http://localhost.openrpa.dk/issue
# saml_baseurl=http://localhost.openrpa.dk:1880/

allow_personal_nodered=false
allow_user_registration=true
```

(continues on next page)

(continued from previous page)

```
# mongodb_url=mongodb://localhost:27017
# mongodb_db=openflow

# api_bypass_perm_check=true
update_acl_based_on_groups=true
multi_tenant=false

websocket_max_package_count=1048576

api_ws_url=ws://localhost.openrpa.dk
domain=localhost.openrpa.dk
protocol=http
amqp_url=amqp://localhost
aes_secret=7TXsxf7cn9EkUqm5h4MEWGjzkxkNCk2K
auto_create_users=true
auto_create_domains=
skip_history_collections=audit,jslog
```

And if the user is using **Windows 10**, he must allow Powershell Scripts to run by changing the Execution Policy using the following command in a Powershell Prompt with escalated privileges.

```
Set-ExecutionPolicy Bypass --Force
```

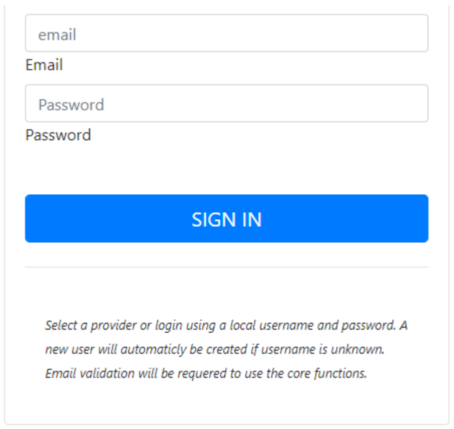
Finally, the user can run `OpenFlow` by pressing `Ctrl+Shift+D`, selecting `OpenFlow` in the dropdown box and clicking the **Play** button.

The user can now access OpenFlow by using a browser and navigating to the `http://localhost.openrpa.dk` URL.

To properly install and configure Node-RED, please refer to the **Installing & First Run** section in the Node-RED chapter.

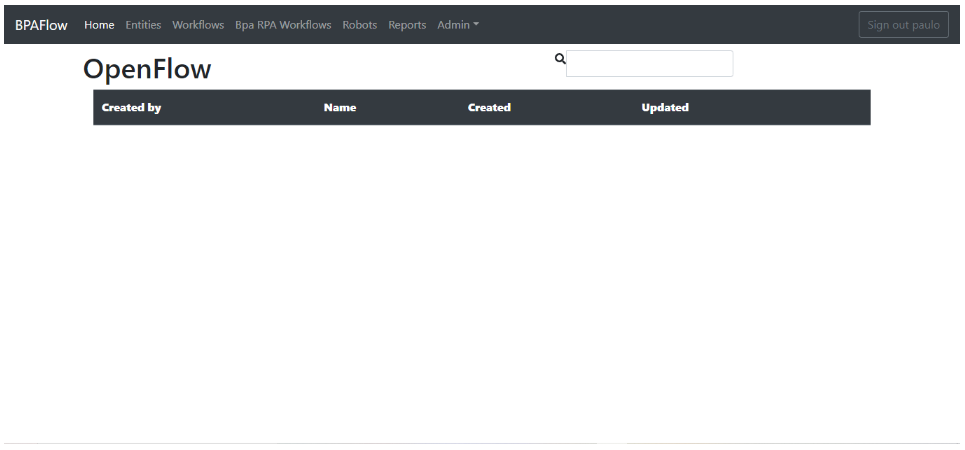
2.3 Accessing for the first time

Upon entering the OpenFlow page in the URL - if set to work locally, the URL is `http://localhost.openrpa.dk` - otherwise it will be provided by the working environment. If none of those are set the user is still able to use OpenFlow's own demo environment, found at `https://demo1.openrpa.dk`. It is also needed to authenticate yourself with the credentials provided. And after signing in, the user is redirected to the home page of OpenFlow, as seen below.



Select a provider or login using a local username and password. A new user will automaticly be created if username is unknown. Email validation will be requered to use the core functions.

Fig. 1: **Authentication page.**



Created by	Name	Created	Updated

Fig. 2: **OpenFlow's home page.**

2.4 Creating a new user

To create a new user first click the Admin dropdown and the Users option.

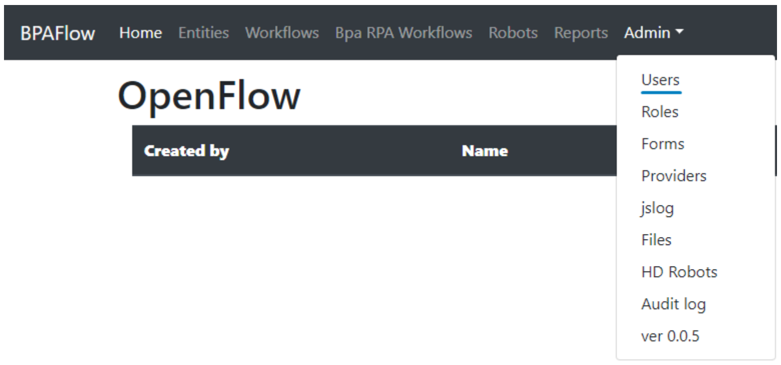
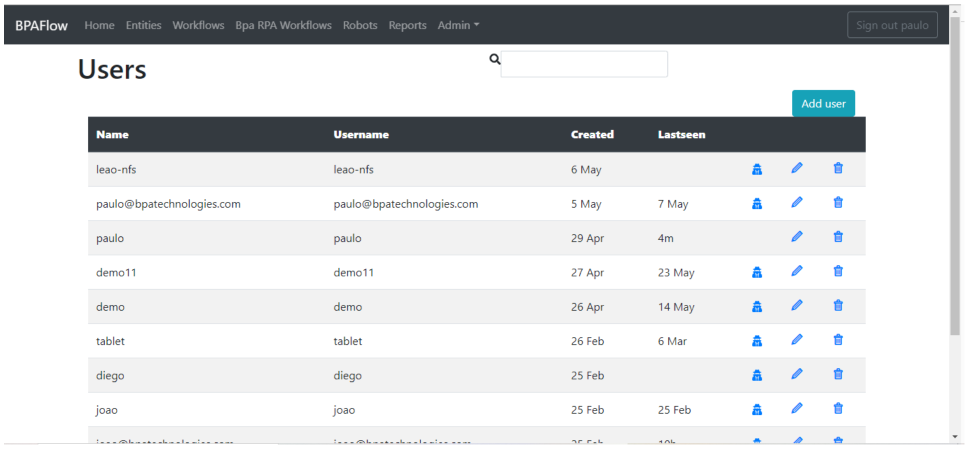


Fig. 3: **Admin dropdown.**

Now click the navy blue `Add user` button.



Name	Username	Created	Lastseen
leao-nfs	leao-nfs	6 May	
paulo@bpatechnologies.com	paulo@bpatechnologies.com	5 May	7 May
paulo	paulo	29 Apr	4m
demo11	demo11	27 Apr	23 May
demo	demo	26 Apr	14 May
tablet	tablet	26 Feb	6 Mar
diego	diego	25 Feb	
joao	joao	25 Feb	25 Feb

Fig. 4: **Users page.**

Now it is needed to fill the given forms and click save.

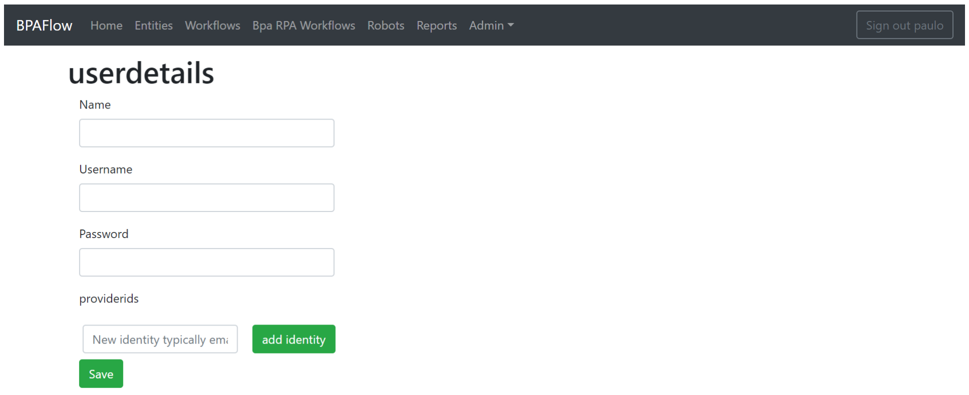


Fig. 5: **User creation page.**

2.5 Roles

2.5.1 What are Roles?

Coming soon - Work in progress

2.5.2 RPARole (Pool)

Coming soon - Work in progress

2.5.3 Granting permissions (admin, nodered, user)

Coming soon - Work in progress

2.6 Workflows Tab (Node-RED)

2.6.1 What is it?

Workflows in Node-RED are mainly called 'Flows'. These flows are made to connect different services, applications and logic functions. When grouped together they can be used to process data and output the results to a given service or file.

At the Workflows page (eg. `http://localhost.openrpa.dk/#/Workflows`) in OpenFlow, it is possible to remotely invoke these flows.

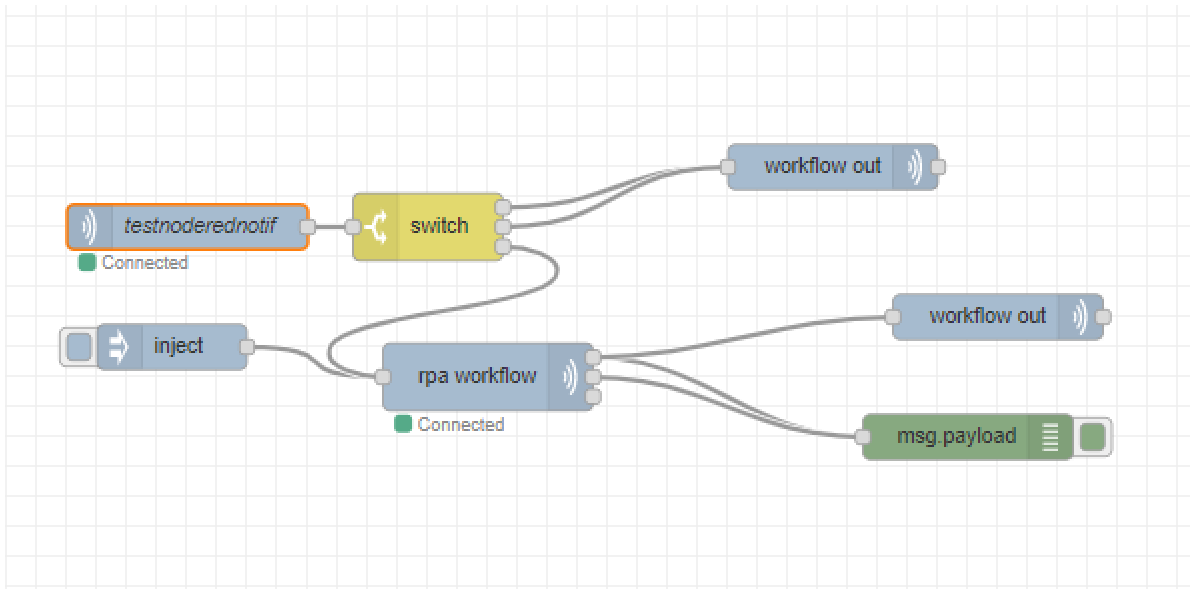


Fig. 6: **Node-RED subflow example.**

2.6.2 Invoking

Workflows can be started through all three applications: OpenRPA, OpenFlow and Node-RED. Invoking a workflow does not change it's content, it merely executes it remotely. Think of it as a read-only procedure.

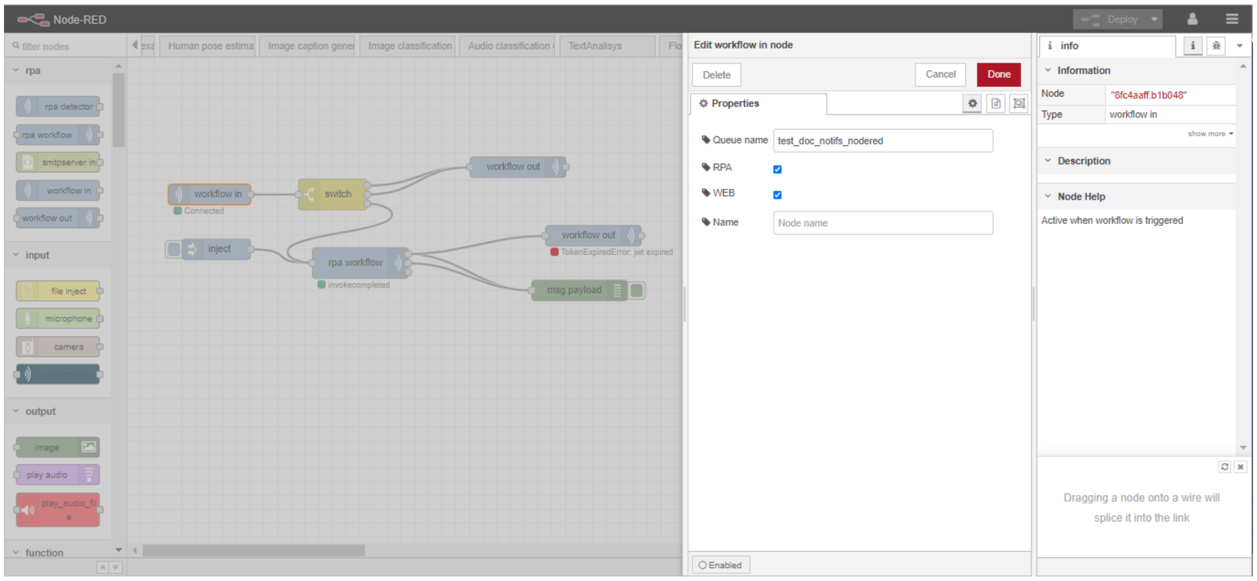
- **Deploying a Workflow to Node-RED**

  Upon accessing Node-RED, flow tabs are shown containing all the flows available for deployment. To execute a Node-RED flow, simply click the starting node inside a Node-RED subflow. Everytime a change is made to a Node-RED flow or subflow it is required to deploy it again so the changes are applied.

  If it is desired to invoke a workflow through OpenFlow according to the settings set on the Node-RED application, a few more steps are required.

  1. Enter the Node-RED page and select the desired flow tab where you want to create the subflow containing the invoke call to OpenFlow.
  2. Create a subflow with a `Workflow In` node as its starting point.
  3. Click twice the `Workflow In` node corresponding to the workflow you want to create.
  4. Name it by setting up the `Queue name` input field.
  5. If desired, route it to send a form back to the user by using `Workflow Out` (more about Forms below).
  6. Check one or both of the checkboxes: RPA - to enable this node to be invoked from OpenRPA - and WEB - to enable invoking from OpenFlow.
  7. Deploy the current flow.

Note: The user must have the `nodered users` role enabled in order to able to invoke Node-RED flows from within OpenFlow.



Queue name	test_doc_notifs_nodered
RPA	✓
WEB	✓
Name	Node name

Node	"8fc4aaff.b1b048"
Type	workflow in

Active when workflow is triggered

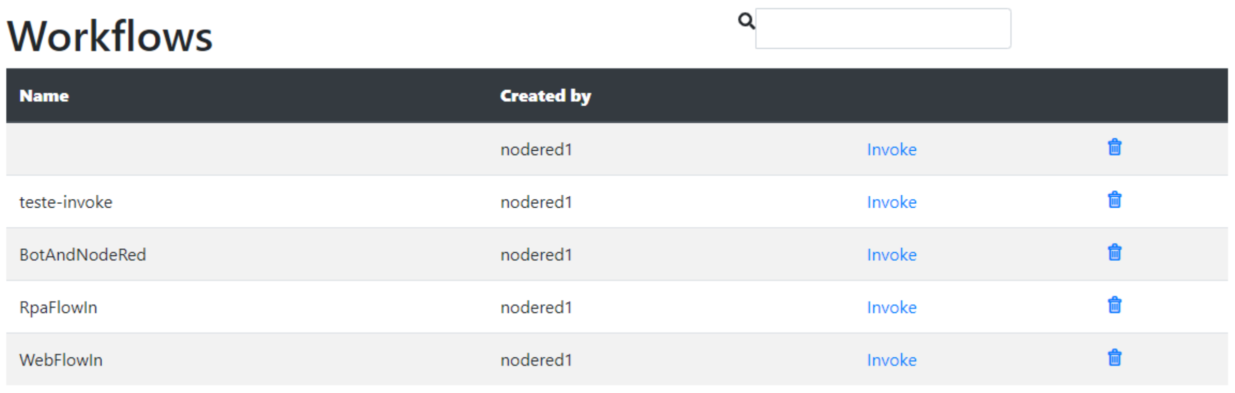
Fig. 7: **Enabling Node-RED invoking from OpenFlow.**

- **Invoking a Node-RED workflow in OpenFlow**

  After the steps above are complete, go to the Workflows page (eg. `http://localhost.openrpa.dk/#/Workflows`) and directly invoke the desired workflow. A form page will then open to choose the input parameters, if any.

  The forms are either automatically generated by OpenRPA, according to the variables preset - more on that on OpenRPA's Variables section - or manually generated by the user on the Forms page (eg. `http://localhost.openrpa.dk/#/Forms`).

  Upon clicking submit on a form, the parameters are sent to the robot for processing and execution of the given workflow.



Workflows

Name	Created by		
	nodered1	Invoke	🗑
teste-invoke	nodered1	Invoke	🗑
BotAndNodeRed	nodered1	Invoke	🗑
RpaFlowIn	nodered1	Invoke	🗑
WebFlowIn	nodered1	Invoke	🗑

Fig. 8: **Workflows page.**

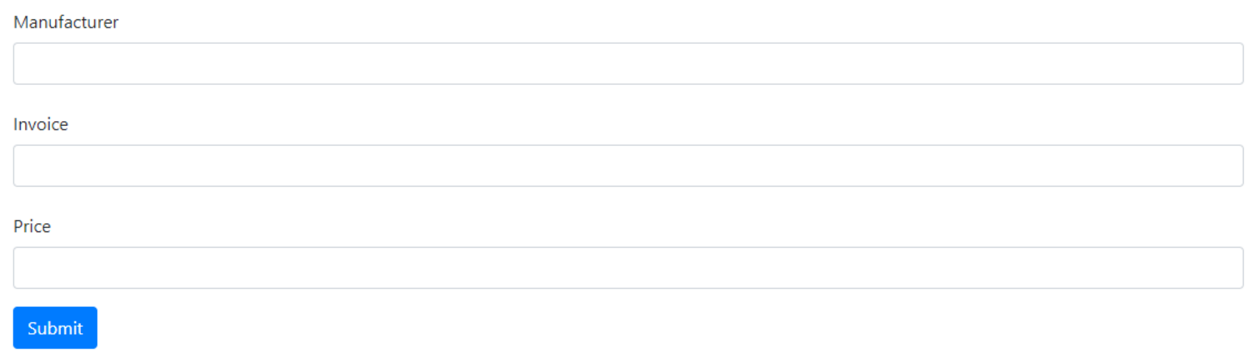


Fig. 9: **Form example.**

2.7 RPA Workflows Tab (OpenRPA)

2.7.1 What is it?

Workflows in OpenRPA and OpenFlow are the same thing, an algorithm or a sequence of steps that execute a meaningful task. The difference is that when you invoke a workflow in OpenFlow, it creates an instance of that workflow. By accessing the Workflows tab, you may invoke workflows remotely, meaning, the stack in OpenFlow will send a message to the available agent to process and execute the given workflow.

From OpenFlow you can create forms, grant permissions to a given workflow and most importantly invoke it.

OpenFlow automatically manages the workflow repository. When properly connected, by saving a workflow inside OpenRPA, it will also automatically appear inside the Workflows webpage.

To download a workflow, simply go to the RPA Workflows link and click download. After dowloading the .XAML file, you may share it with others or import it into your OpenRPA client.

2.7.2 Invoking

- **Methods for Invoking**

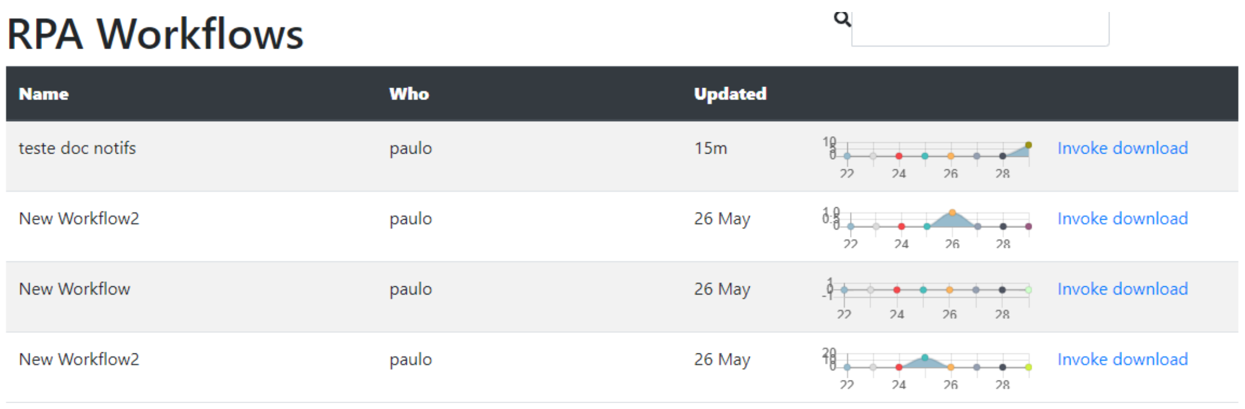
  Here we discuss the methods for invoking a workflow using OpenFlow.

- **Invoking through OpenFlow's RPA Workflows Page**

  To invoke a workflow through OpenFlow, simply go to RPA Workflows page (eg. http://localhost.openrpa.dk/#/Workflows) and click Invoke, another page for the specific workflow will be opened where all the forms needed to be filled are going to be presented. Simply fill them and click Invoke again. The input data is then sent to the chosen robot/agent and it will start processing the workflow.

  Data processing is bi-directional: input parameters are sent to a robot/agent and the workflow output will also be returned. That means that you can make many workflows calling different applications. Think of it as a message, messages are sent, read and replied to. Nothing prevents that message from being sent, read or replied to multiple times.

Note: For the user to invoke a workflow using OpenFlow, the user must have the proper permissions. See more at OpenRPA's chapter Granting permissions to users/roles.


RPA Workflows

Name	Who	Updated		
teste doc notifs	paulo	15m		Invoke download
New Workflow2	paulo	26 May		Invoke download
New Workflow	paulo	26 May		Invoke download
New Workflow2	paulo	26 May		Invoke download

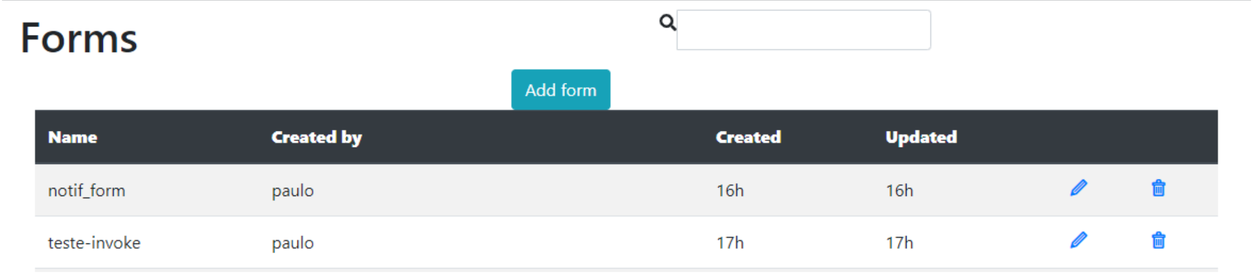
Fig. 10: **RPA Workflows page.**

2.8 Forms

Forms are an user-friendly way of passing input to a workflow by creating dynamic OpenFlow's webpages. There are two ways of generating a form, one of them is through OpenFlow's automatically generated forms which are created upon saving a Workflow into its repository and the other one is manually creating a form and connecting it to a Node-RED workflow.

2.8.1 Creating a Form

Creating a form is rather easy and simple. Go to the Forms page, where all forms are listed, available under the Admin dropdown or on the Forms page (eg. http://localhost.openrpa.dk/#/Forms) and click the Navy blue `Add form` button.


Forms

Name	Created by	Created	Updated
notif_form	paulo	16h	16h
teste-invoke	paulo	17h	17h

Fig. 11: **OpenFlow's Forms page.**

Now at the Forms edit page, there are many basic from which you can choose. For synthetic purposes, we are only going to discuss the most used one here: Text Field Component.

Drag the Form from the Basic category into the Form workspace. Immediately after, a window containing all the parameters to configure the form will appear.

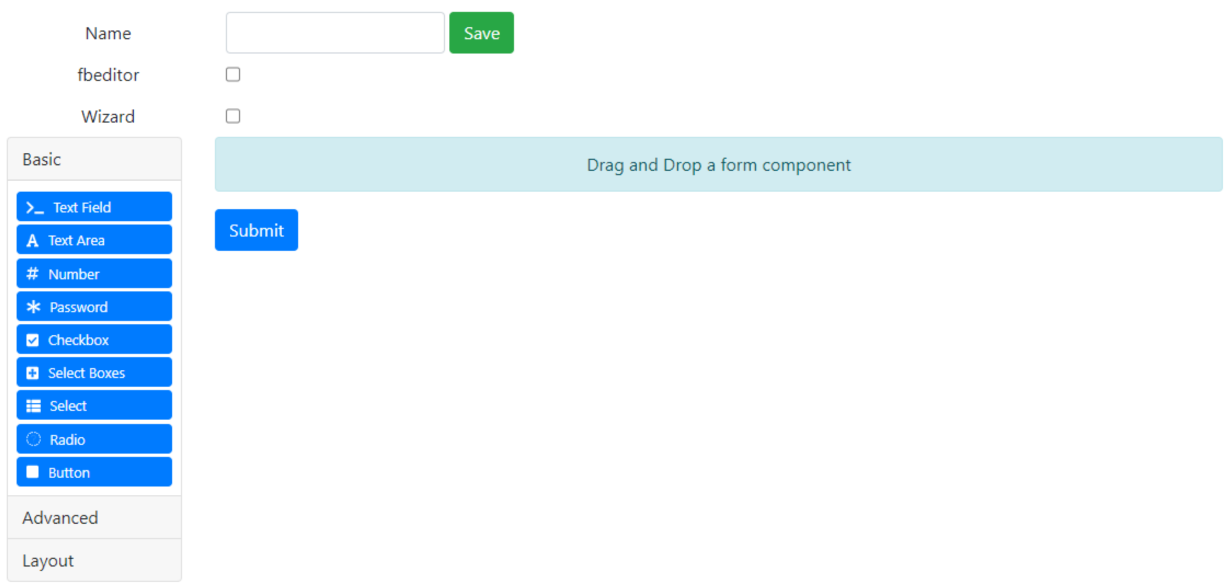
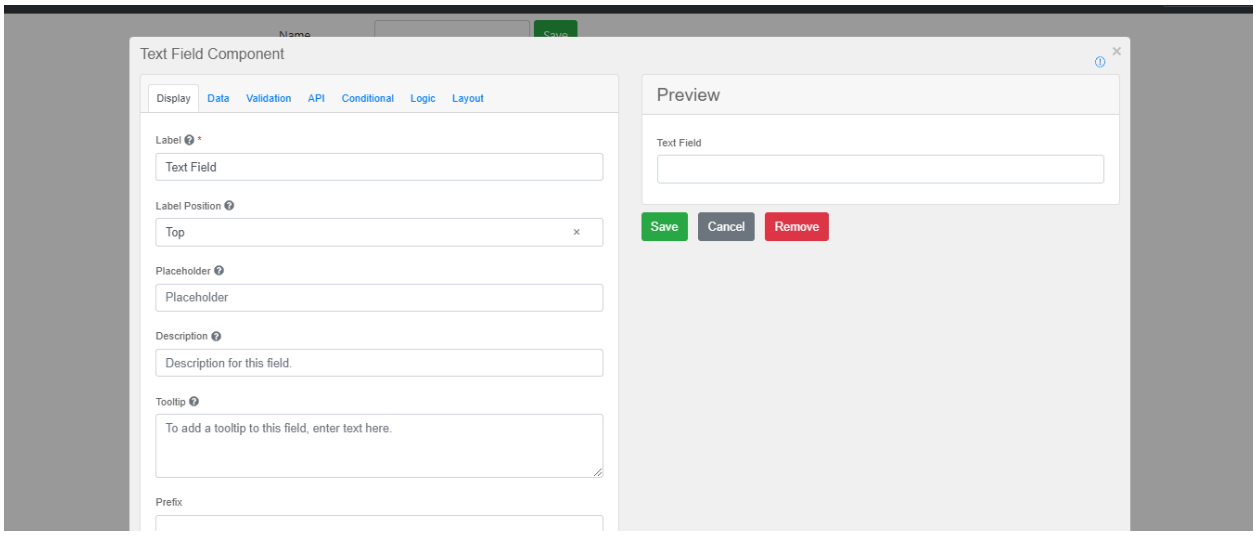


Fig. 12: **OpenFlow's Forms creation page.**



Label *
Text Field
Label Position
Top
Placeholder
Placeholder
Description
Description for this field.
Tooltip
To add a tooltip to this field, enter text here.
Prefix

Text Field

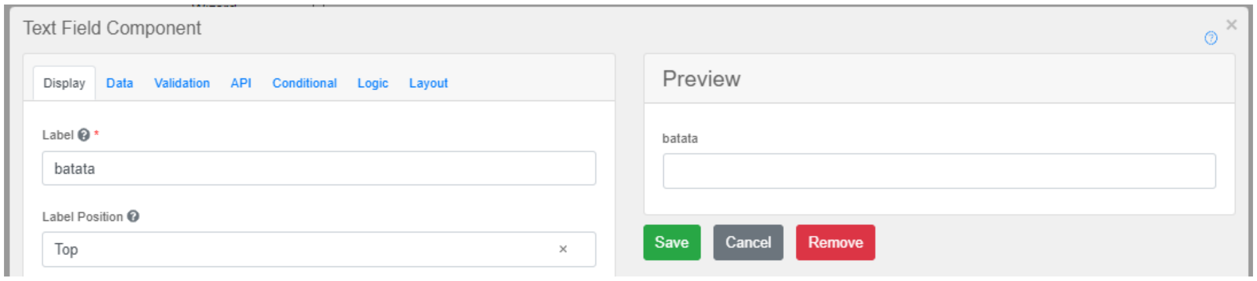
Fig. 13: **OpenFlow's Forms starter configuration tab.**

2.8.2 Configuring the Form

Below are the steps needed to properly configure a form, in our case TextField Form.

- **Changing Form Label**

  To change the Form's label, ie. the title which will appear for the end-user, simply click the Display tab and change the input form titled **Label**, the changes are shown real-time.



Label *

batata

Label Position

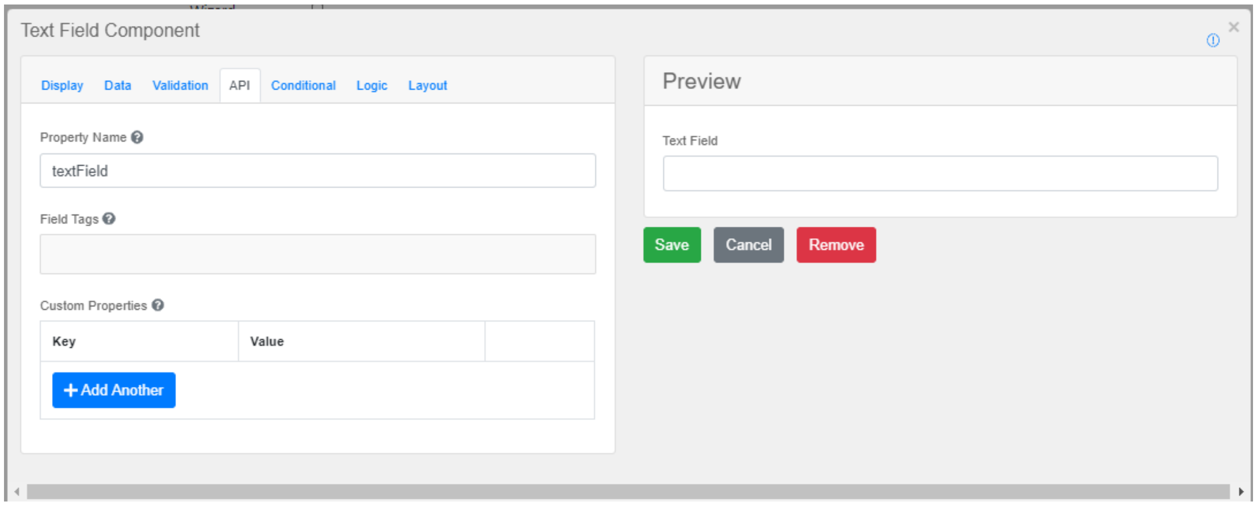
Top

batata

Fig. 14: **OpenFlow's Form Display configuration tab.**

- **Assigning Input Variable**

  To assign the input form to a variable configured inside the OpenRPA workflow you've mapped to OpenFlow, simply go to the API tab and insert the name of the variable inside **Property Name** and press save. Now the next time this workflow is called, a new parameter will appear.



Property Name

textField

Field Tags

Custom Properties

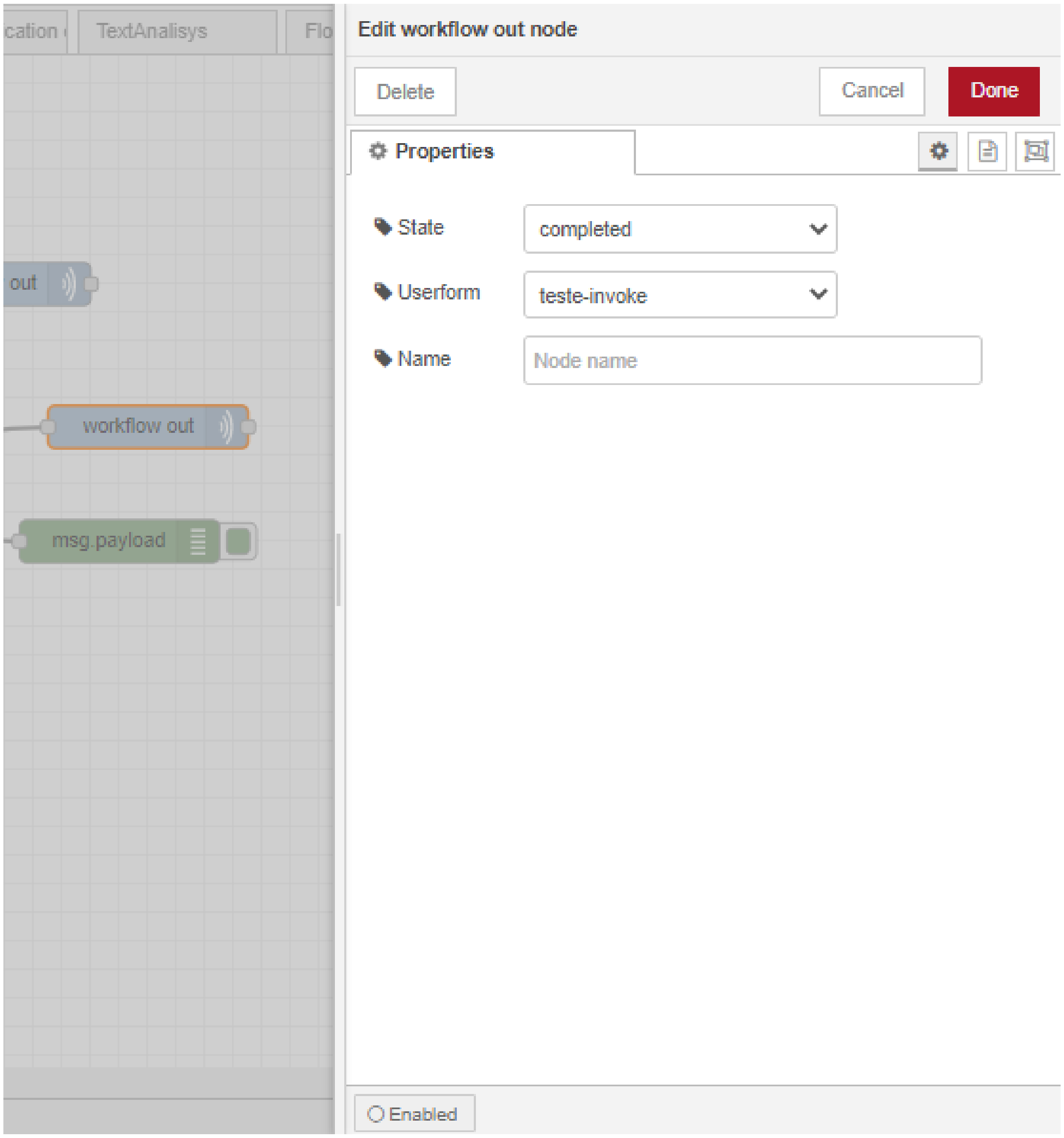
Key	Value

Text Field

Fig. 15: **OpenFlow's Form API configuration tab.**

- **Assigning Form to Node-RED Workflow**

  Now it is required that you assign a Form to a Node-RED workflow. To do that, go to Node-RED and click twice on the `Workflow Out` node you want to assign the form to. The `Edit workflow out node` tab appears and you must select the form you've just created in the `Userform` dropdown.



State: completed

Userform: teste-invoke

Name: Node name

Fig. 16: **Assigning Form to Node-RED workflow.**

2.8.3 Basic Forms

The Basic forms contain the most basic forms of inputs.

Text Field

Coming soon - Work in progress

Text Area

Coming soon - Work in progress

Number

Coming soon - Work in progress

Password

Coming soon - Work in progress

Checkbox

Coming soon - Work in progress

Select Boxes

Coming soon - Work in progress

Select

Coming soon - Work in progress

Radio

Coming soon - Work in progress

Button

Coming soon - Work in progress

2.8.4 Advanced Forms

The Advanced forms contain the most advanced forms of inputs.

Email

Coming soon - Work in progress

URL

Coming soon - Work in progress

Phone Number

Coming soon - Work in progress

Tags

Coming soon - Work in progress

Address

Coming soon - Work in progress

Date / Time

Coming soon - Work in progress

Day

Coming soon - Work in progress

Time

Coming soon - Work in progress

Currency

Coming soon - Work in progress

Survey

Coming soon - Work in progress

Signature

Coming soon - Work in progress

2.8.5 Layout Forms

The Layout forms contain view-related forms of input.

HTML Element

Coming soon - Work in progress

Content

Coming soon - Work in progress

Columns

Coming soon - Work in progress

Field Set

Coming soon - Work in progress

Panel

Coming soon - Work in progress

Table

Coming soon - Work in progress

Tabs

Coming soon - Work in progress

Well

Coming soon - Work in progress

2.9 Entities - MongoDB

2.9.1 What are entities?

Coming soon - Work in progress

2.9.2 Usages

Coming soon - Work in progress

2.9.3 Collections

Coming soon - Work in progress

Chapter 3

OpenRPA

3.1 What is OpenRPA?

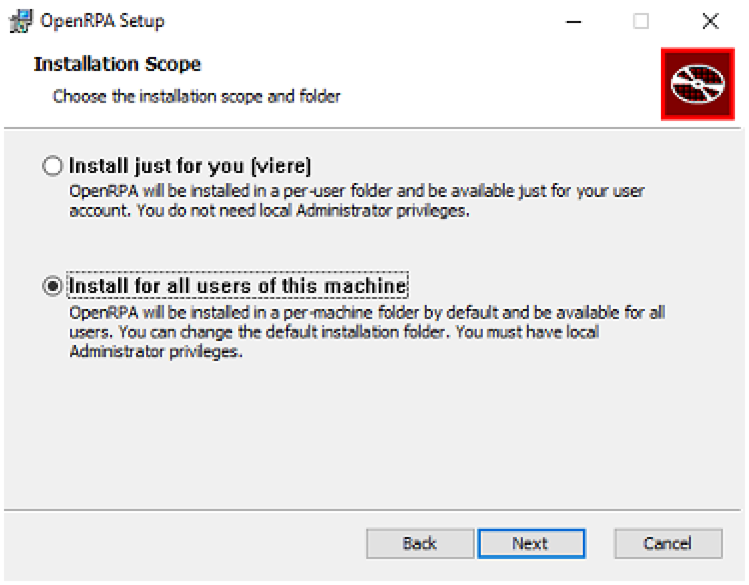
OpenRPA is where the actual automation happens, inside it you create the workflows and invoke them, run all the activities needed to complete your task and deploy them with the assistance of OpenFlow and Node-RED. The framework also offers integration with other tools that are essential such as message brokers and repository management tools and OpenRPA also include other features within itself, such as Image/OCR, Browser Navigation, and many others.

3.2 Installing & First Run

To install OpenRPA you first need to download the installer executable provided here (`https://github.com/open-rpa/openrpa/releases/latest/download/OpenRPA.msi`). Given that the stack OON is open source, the user may also run OpenRPA directly from the source code by using Visual Studio 2019.

3.2.1 Installing OpenRPA

Installing Universally or User-Only


OpenRPA Setup

Installation Scope

Choose the installation scope and folder

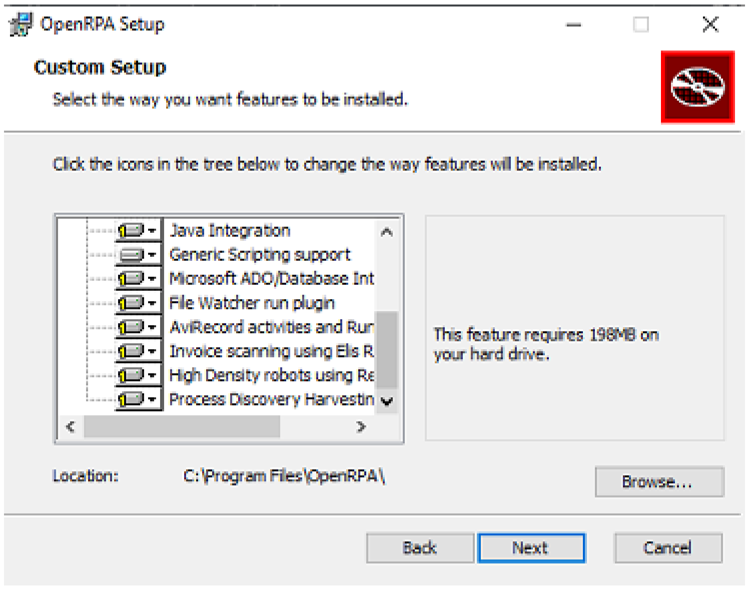
Install just for you (viere)
OpenRPA will be installed in a per-user folder and be available just for your user account. You do not need local Administrator privileges.

[✓ Install for all users of this machine]
OpenRPA will be installed in a per-machine folder by default and be available for all users. You can change the default installation folder. You must have local Administrator privileges.

Fig. 1: **OpenRPA's installation scope.**

The installer first provides the option to install OpenRPA user-only or universally. It is recommended to install for all users in the machine since a local installation will prevent anyone else using the machine from executing it. After selecting the desired option, click Next.

Installing OpenRPA libraries


OpenRPA Setup

Custom Setup

Select the way you want features to be installed.

Click the icons in the tree below to change the way features will be installed.

- Java Integration
- Generic Scripting support
- Microsoft ADO/Database Int
- File Watcher run plugin
- AviRecord activities and Rur
- Invoice scanning using Elis R
- High Density robots using Re
- Process Discovery Harvestin

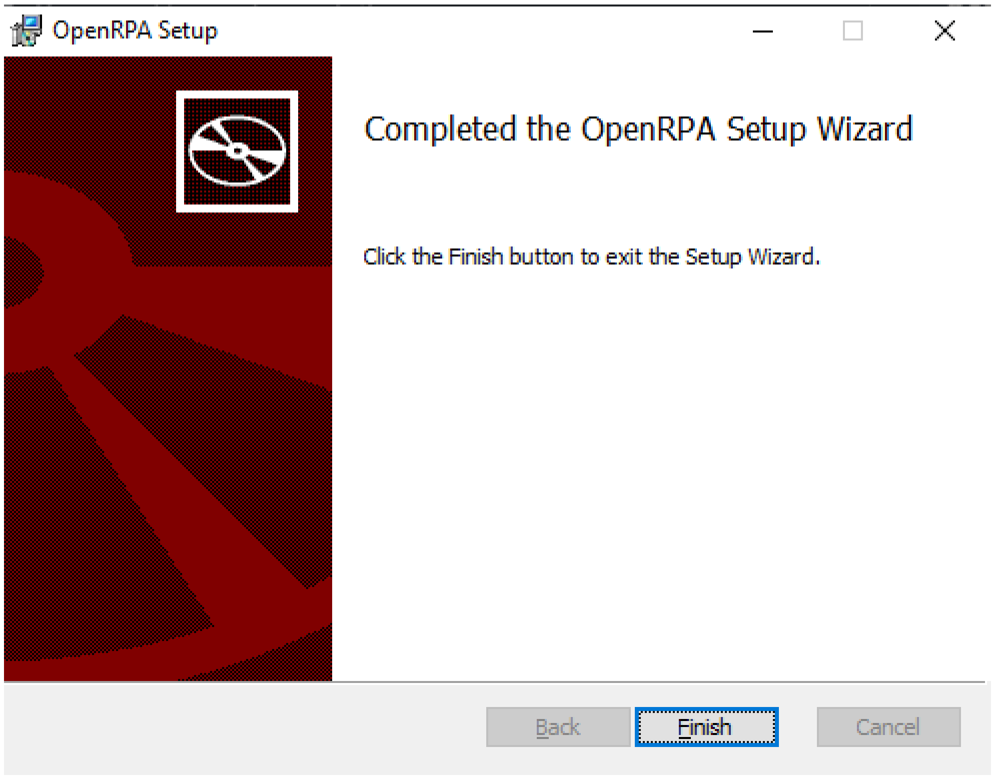
This feature requires 198MB on your hard drive.

Location: C:\Program Files\OpenRPA\

Fig. 2: **OpenRPA's installation libraries.**

There is a bundle of libraries from OpenRPA available for installation. It is recommended to install all of them. After selecting them, click Next once again and Install so the installation can be concluded.

If everything goes well the installation finishes and the window on **Fig. 3** appears.


OpenRPA Setup

Completed the OpenRPA Setup Wizard

Click the Finish button to exit the Setup Wizard.

Fig. 3: **OpenRPA finished installation.**

3.3 Settings.json

The `settings.json` file is the default JSON configuration file for OpenRPA. It is located inside `C:\Users\{YOUR_USER}\Documents\OpenRPA` and contains most configuration parameters for OpenRPA which are not configurable through it's GUI.

3.3.1 wsurl

The parameter `wsurl` corresponds to the websocket URL which you will be using to connect to the OpenFlow orchestrator. If you do not want to set a specific websocket, you may leave this parameter as-is.

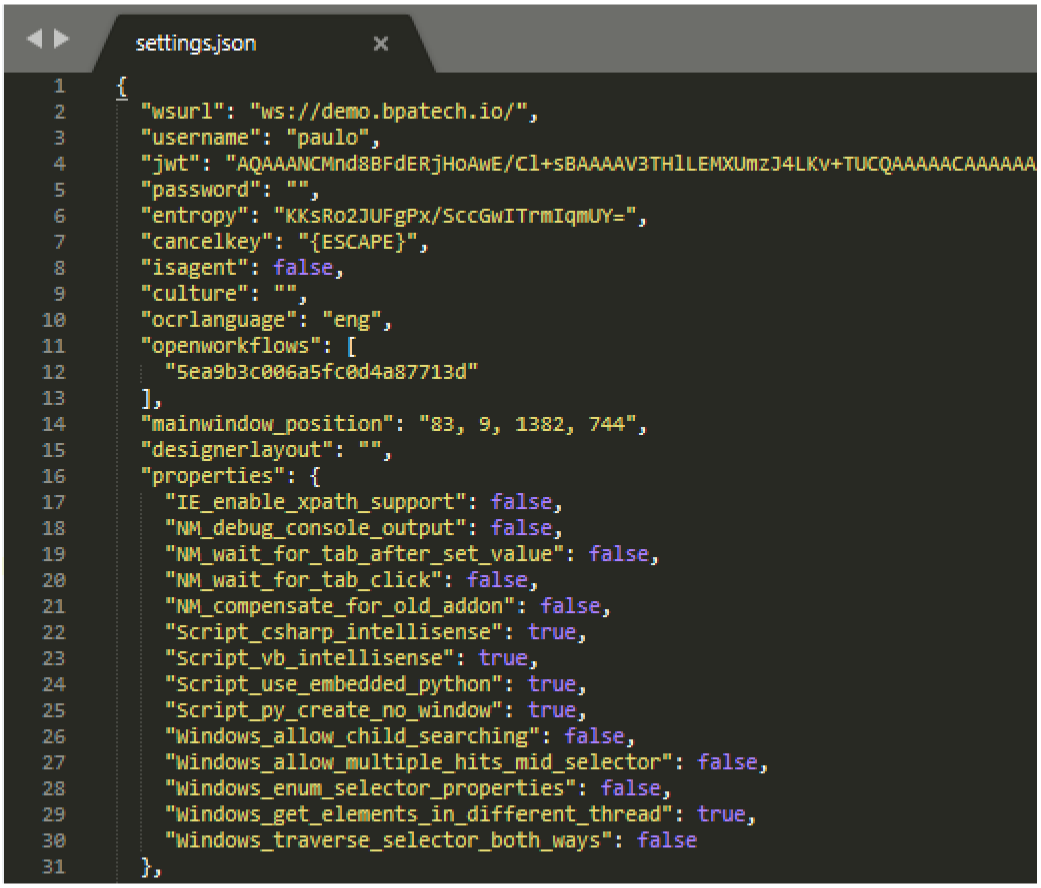
If you are not connected to any websocket, you can only see the workflows you have cached upon your last access to a repository. Or, if you have not ever connected to a websocket, you can only see workflows you have built.

Note: If you do not have any websocket set, you are going to obligatorily run OpenRPA locally and will lose the benefits of the OON stack integration. It is preferable to at least use the demo OpenFlow websocket provided by OpenRPA, which is located at `wss://demo1.openrpa.dk`.

3.3.2 isagent

If this parameter is set to true, the client machine and, consequently, the user no longer has access to view or edit the workflows. This also means that OpenRPA simply becomes an agent/run-time service.

If you do not know what this means, leave it as-is.


settings.json

```
{
  "wsurl": "ws://demo.bpatech.io/",
  "username": "paulo",
  "jwt": "AQAAANCMnd8BFdERjHoAwE/Cl+sBAAAAV3THlLEMXUmzJ4LKv+TUCQAAAAACAAAAAA
  "password": "",
  "entropy": "KKsRo2JUFgPx/SccGwITrmIqmUY=",
  "cancelkey": "{ESCAPE}",
  "isagent": false,
  "culture": "",
  "ocrlanguage": "eng",
  "openworkflows": [
    "5ea9b3c006a5fc0d4a87713d"
  ],
  "mainwindow_position": "83, 9, 1382, 744",
  "designerlayout": "",
  "properties": {
    "IE_enable_xpath_support": false,
    "NM_debug_console_output": false,
    "NM_wait_for_tab_after_set_value": false,
    "NM_wait_for_tab_click": false,
    "NM_compensate_for_old_addon": false,
    "Script_csharp_intellisense": true,
    "Script_vb_intellisense": true,
    "Script_use_embedded_python": true,
    "Script_py_create_no_window": true,
    "Windows_allow_child_searching": false,
    "Windows_allow_multiple_hits_mid_selector": false,
    "Windows_enum_selector_properties": false,
    "Windows_get_elements_in_different_thread": true,
    "Windows_traverse_selector_both_ways": false
  },
```

3.4 Plugins

3.4.1 Chrome, Firefox

Coming soon - Work in progress

3.4.2 AVI Recorder

Coming soon - Work in progress

3.5 Interface

3.5.1 Activities/Toolbox

The toolbox is where all the possible activities in OpenRPA are contained. Imagine it as a box of building tools where all the instruments you'd normally use are, such as a screwdriver or a driller. The toolbox is essentially where you'd reach out for getting the tools for, take it as an example, fixing a faucet.

The activities in OpenRPA are separated by categories which clearly represent what they are used for.

Scope

It refers to the reach of the objects contained within a sequence. If you have some knowledge of programming and scope, you can skip this part. First, let's think of it as sets. In this example, we are going to use variables since arguments function as if they were globally scoped.

Per the code block below, one may observe that sequence **B** has access to only variables x, **-** but since **B** is inside the scope of **A**, it makes sequence **B** to have access to all variables {n, p, x, y}. In simple words, sequences can be used to organize variable scopes and to group activities, just like a developer would normally do in a programming language using functions to avoid monolithic code.

```
Sequence A {

        variables = n, p

        Sequence B {

                variables = x, y

        }
}
```

3.5.2 Designer

The designer is where the main process of desigining and implementing a task happens inside OpenRPA. It is where you drag and drop activities and add and nest sequences to fulfill your implementation choices, as well as set properties and variables pertaining to your workflow business process. It contains five components: Toolbox & Snippets sidebar, Workspace, Properties box, Output bar and Connection bar.

Toolbox & Snippets sidebar

This is the place where all activities lie as well as some code snippets which teach the user to make use of the tool for various purposes.

To use an activity, simply drag it into the `Workspace` sequence you want to add it to.

The activities are divided in 17 different sections.

1. **OpenRPA**

   Here lie the most general activities which are concerned to the most common tasks used inside OpenRPA, such as UI actions and invoking other workflows.

2. **OpenRPA.AviRecorder**

   Activities concerned with the **AviRecorder** plugin.

3. **OpenRPA.Database**

   Database operations such as **Execute** and **Update**.

1. **OpenRPA.ElisRossum**

   Rossum AI activities, for more on that please refer to the Rossum page (`https://rossum.ai/`).

2. **OpenRPA.Forms**

   Invoke **OpenRPA Forms** and **Show Notification.** For more on that please refer to the **Forms** section in this chapter.

3. **OpenRPA.IE**

   IE-navigation activities.

4. **OpenRPA.Image**

   Activities for manipulating images through OCR or other means.

5. **OpenRPA.Java**

   Activity for finding a Windows Java element.

6. **OpenRPA.NM**

   Chrome-navigation activities.

7. **OpenRPA.Office**

   Manipulation of UI and data inside **MS Office** applications.

8. **OpenRPA.OpenFlowDB**

   Manipulating data inside OpenFlow's MongoDB database.

9. **OpenRPA.SAP**

   Coming up soon - work in progress

10. **OpenRPA.Script**

    Coming up soon - work in progress

11. **OpenRPA.Utilities**

    Coming up soon - work in progress

12. **OpenRPA.Windows**

    Coming up soon - work in progress

13. **System.Activities**

    Coming up soon - work in progress

14. **System.Activities.Core.Presentation**

    Coming up soon - work in progress

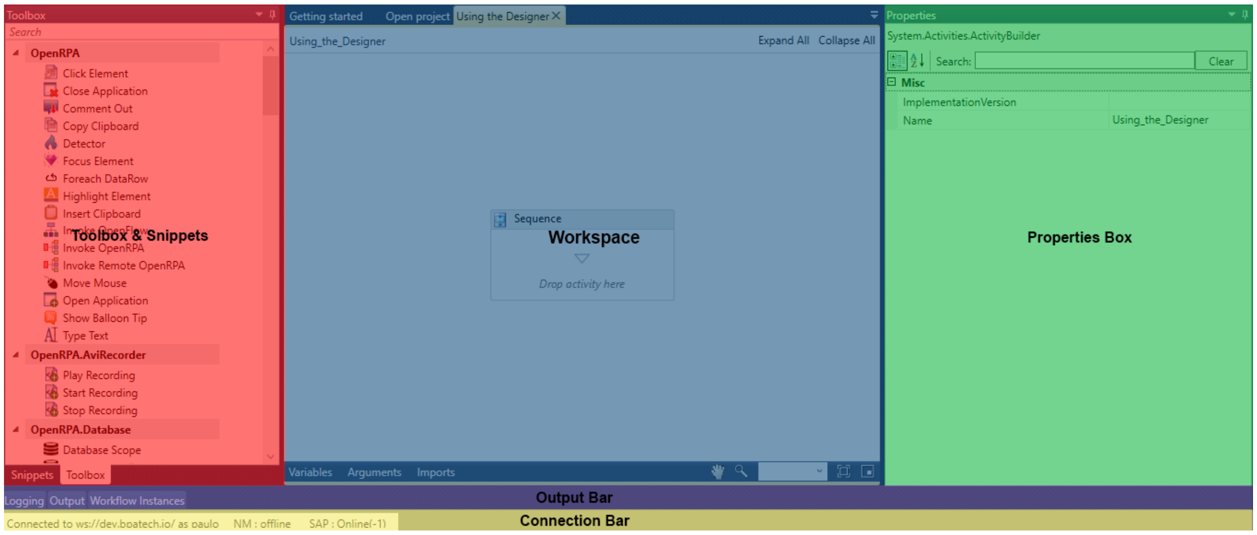
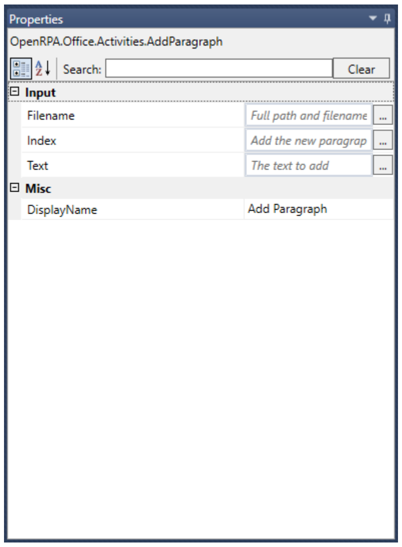


Fig. 4: **OpenRPA Designer components.**

3.5.3 Property Box

Every activity in OpenRPA has it's given property box, containing parameters passed that alter the mode of functioning of the given activity. Inside the property box there are also contained sections which are pretty much selfexplanatory and refer to the given parameters they cause changes upon, such as **Input** or **Misc.**



Input

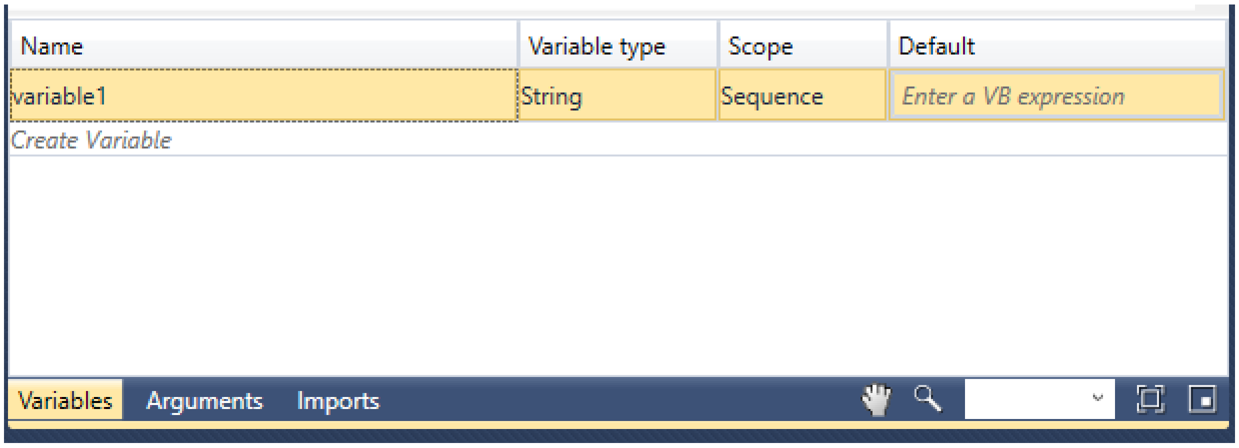
Filename	*Full path and filename*
Index	*Add the new paragrap*
Text	*The text to add*

Misc

DisplayName	Add Paragraph

Fig. 5: **Properties box.**

3.5.4 Variables, Arguments



Name	Variable type	Scope	Default
variable1	String	Sequence	Enter a VB expression
Create Variable			

Fig. 6: **Variables & Arguments box.**

Variables

Variables, as their very name states - are changeable values assigned to aliases inside the execution of a sequence. These aliases can be used in mathematical or computational expressions. For example, if you want to save a variable named "price" to assign the price of a product along a sequence. You can use the assign activity and an expression to assign it's value.

Upon inserting an activity with an input box, if you mention a non-existent variable and want to easily create it you can press `Ctrl+K` and a dropdown will appear with many types of variable to be selected.

Variables in OpenRPA are statically typed and their type can be selected from the **variable type** field inside the argument box.

Arguments

Arguments are simply variables that can be passed to other sequences or external services and have global scope. They act exactly as variable, except for that single characteristic. If you pass an argument to a sequence created within another sequence, the argument is still valid within the outer sequence. If you pass a variable, that does not happen.

It is important to notice that both variables and arguments can have a **default value**, which are assigned to both if a value isn't necessarily assigned during the execution of the sequence. It is also required for arguments with the **isRequired** property checked to have an 'In' value, ie. already passed or having a default.

Similarly to Variables, Arguments in OpenRPA are statically typed and their type can be selected from the **variable type** field inside the argument box.

Output (In - In/Out - Out) Output refers to the way a variable or argument is handled during run-time. They are classified below.

- In

  In this case the argument is received from external service and is only given an input value that can't be returned after the execution of the workflow

- In/Out

In this case the argument can both be given an input value and be changed and outputted after the execution of the workflow

- Out

  In this case the argument returns a value to an external service, and it can't be changed after the execution of the workflow

Imports

Imports are all the modules used by the sequence.

3.5.5 Toolbox Detailed

OpenRPA

Here belong the activities located inside the OpenRPA toolbox.

`Click Element`

Clicks a given position inside an application. You must first select the application with the **Get Element** activity, which will be discussed later.

Properties Parameters

`Animate Mouse` - If set to `True`, the cursor will appear changing it's position throughout execution.

`Displayname` - Title of the activity inside the sequence.

`Double Click` - If set to `True`, the cursor clicks twice inside the window.

`Element` - The application selected with the **Get Element** activity.

`Key Modifiers` - If set, the corresponding keys act as if pressed during execution.

`Mouse Button` - Set to `1` if the Left Mouse button is to be clicked, `2` if it is the Right Mouse button.

`Post Wait` - How much time OpenRPA should wait for this activity to finish after completing its execution.

`Virtual Click` - Try to do a **Virtual Click** instead of actually clicking the mouse.

`X Offset` - The horizontal offset inside the window to be clicked. Units are in pixels. It also allows for negative values.

`Y Offset` - The vertical offset inside the window to be clicked. Units are in pixels. It also allows for negative values.

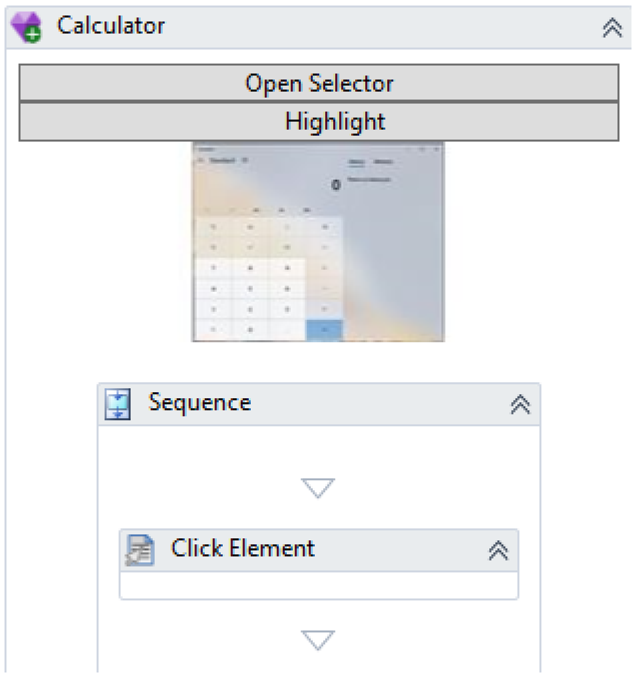


Fig. 7: **OpenRPA Click Element.**

`Close Application`

Close a given application selected with the **Get Element** activity.

Properties Parameters

`Displayname` - Title of the activity inside the sequence.

`Force`- If set to `True`, kills the application.

`Selector` - Contains all the data pertaining to the application selected.

`Timeout` - Time until the activity ceases if not successfully able to close the application.

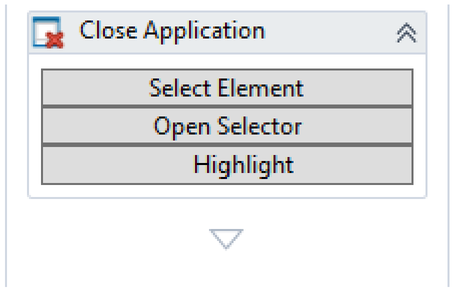


Fig. 8: **OpenRPA Close Application.**

`Comment Out`

Every activity put inside this one, which resembles a sequence, is ignored by the execution of the workflow. This activity is useful for remarking something out of the sequence.

Properties Parameters

`Displayname` - Title of the activity inside the sequence.

`Copy Cliboard`
Coming soon - Work in progress

`Detector Activity`
Detector is the main `Event Listener` inside OpenRPA - refer to Event listener (https://www.computerhope.com/jargon/e/event-listener.htm) if you don't know what that means. Simply put, it is an activity that waits for something to happen either before the Workflow continues processing and executing its functions.

There are currently five types of different event listeners. Here only one is going to be discussed: `KeyboardSequence`.

`Foreach DataRow`
Iterates through a DataRow inside an Excel Data Table or Data View.[2] All the activities inside this one will be looped over each row.

Properties Parameters

`Data Table` - The data table used to gather the data.

`Data View` - The data view used to gather the data.

`Displayname` - Title of the activity inside the sequence.

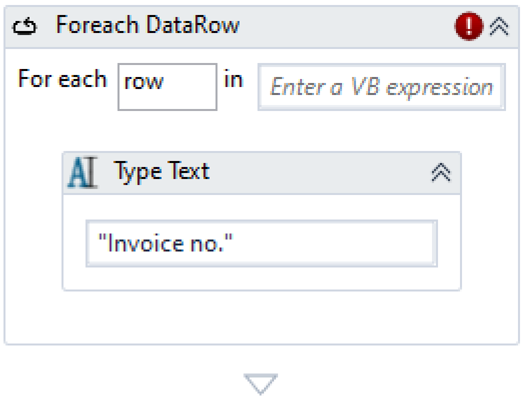


Fig. 9: **OpenRPA Foreach DataRow.**

[2] - What is the difference between DataView and DataTable? (`https://stackoverflow.com/questions/7382932/what-is-the-difference-between-dataview-and-datatable`)

`Highlight Element`
Coming soon - Work in progress

`Insert Clipboard`
Coming soon - Work in progress

`Invoke OpenFlow`
Allos the user to invoke Workflows which have been configured with OpenFlow/Node-RED, ie. available at the Workflows page (eg. `http://localhost.openrpa.dk/#/Workflows`).

Properties Parameters

`Displayname` - Title of the activity inside the sequence.

`Wait until Completed` - If set to `True`, OpenRPA waits until the Workflow invoke call is finished - either success or error. If successful, the message containing the data (as designed in the flow at Node-RED) is returned to the OpenRPA robot.

If set to `False`, however, OpenRPA no longer has access to the returned message - communication now is one-way - and therefore all logic that would otherwise depend on it must then be redesigned. Think of it as a try/catch exception block that does not return any exception message. [3]

`Workflow` - The workflow which is to be invoked.

Note: The workflow is invoked by the user currently logged in OpenRPA/OpenFlow. It means that the user must have access to the desired workflow which is to be invoked, otherwise it won't even be displayed in the workflows dropdown.

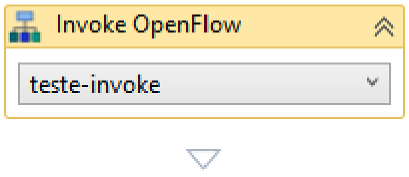


Fig. 10: **OpenRPA Invoke OpenFlow.**

`Invoke OpenRPA`

Allows the user to invoke RPA Workflows saved in OpenFlow's repository, those available at the RPA Workflows page (eg. `http://localhost.openrpa.dk/#/RPAWorkflows`).

The **Map Variables** button allows the currently set variables and arguments to be passed to the workflow invoke call. In order to do that, after choosing your workflow from the dropdown, create the arguments in the **Variables & Arguments Box** and click the **Map Variables** button in the activity.

Properties Parameters

`Displayname` - Title of the activity inside the sequence.

`Wait until Completed` - If set to `True`, OpenRPA waits until the Workflow invoke call is finished - either success or error. If successful, the message containing the data (as designed in the flow at Node-RED) is returned to the OpenRPA robot.

If set to `False`, however, OpenRPA no longer has access to the returned message - communication now is one-way - and therefore all logic that would otherwise depend on it must then be redesigned. Think of it as a try/catch exception block that does not return any exception message.

`Workflow` - The workflow which is to be invoked.

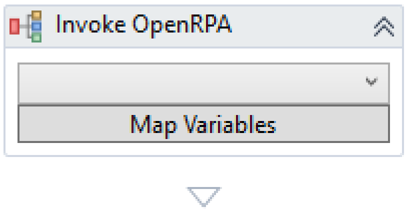


Fig. 11: **OpenRPA Invoke OpenRPA.**

`Invoke Remote OpenRPA`

Allows to remotely invoke RPA Workflows, saved in OpenFlow's repository, to be executed at other OpenRPA clients.

The **Map Variables** button allows the currently set variables and arguments to be passed to the workflow invoke call. In order to do that, after choosing your workflow from the dropdown, create the arguments in the **Variables & Arguments Box** and click the **Map Variables** button in the activity.

Properties Parameters

`Displayname` - Title of the activity inside the sequence.

`Target` - OpenFlow's hash string identifying the target which will be activated to execute the workflow.

`Wait until Completed`

If set to `True`, OpenRPA waits until the Workflow invoke call is finished - either success or error. If successful, the message containing the data (as designed in the flow at Node-RED) is returned to the OpenRPA robot.

If set to `False`, however, OpenRPA no longer has access to the returned message - communication now is one-way - and therefore all logic that would otherwise depend on it must then be redesigned. Think of it as a try/catch exception block that does not return any exception message.

`Workflow` - OpenFlow's hash string identifying the workflow which will be executed.

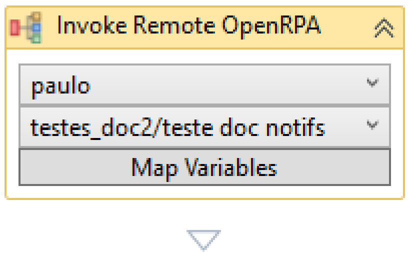


Fig. 12: **OpenRPA Invoke Remote OpenRPA.**

`Move Mouse`

Moves the cursor to a given position inside an application. The application must be selected by using the **Get Element** activity.

Properties Parameters

`Animate Mouse` - If set to `True`, the cursor will appear changing its position throughout execution. Good for showing what is happening in a presentation.

`Displayname` - Title of the activity inside the sequence.

`Element` - The application selected with the **Get Element** activity.

`Post Wait` - How long OpenRPA should wait for this activity to finish after completing its execution.

`Virtual Click` - If set to true, does a **Virtual Click** instead. Virtual clicking allows clicks on not focused and even hidden applications, but sometimes there might be limitations based on the application. As a rule of thumb, always try to first automate using Virtual Clicks and if it does not work then set it to `False` for futrher debugging/testing.

`X Offset` - The horizontal offset inside the window which the cursor will be moved to. Units are in pixels. It also allows for negative values.

`Y Offset` - The vertical offset inside the window which the cursor will be moved to. Units are in pixels. It also allows for negative values.

Note: This activity requires a Mouse IO. If it fails to find one, it will return an error. If it fails to find one, it will return "Could not send keyboard input. Error Code: 5". This may happen if OpenRPA Client was launched from inside a Windows Remote Desktop and the session was disconnected. To avoid this, please use HDRobots extension, do not disconnect/close the RDP session or make the session perssistent by using other techniques (ie: running a VNC server)

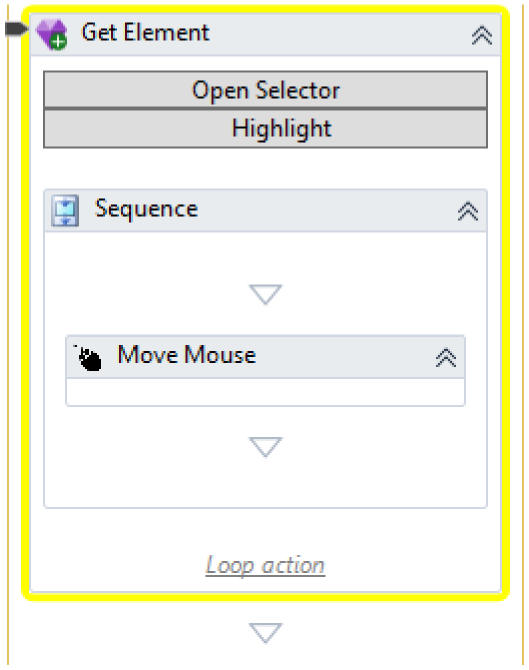


Fig. 13: **OpenRPA Move Mouse.**

`Open Application`

Searches for an application, based on the selector obtained from the **Select Element** button, and starts that application. This activity also works similarly to Get Element, where you can drop other activities that will be executed inside it.

Clicking **Highlight** will red-highlight the first element that matches the criteria on the selector. Useful for checking if the selector is properly set up.

The image shows an example containing a **Show Balloon Tip** activity.

Properties Parameters

`Animate Move` - Visibly move the window of the application which will be opened, instead of instantly moving it.

`Check Running` - Check if application is already running, add explanation from OpenRPA's properties.

`Displayname` - Title of the activity inside the sequence.

`Height` - If set, moves the application to this vertical offset.

`Result` - Save the result of the execution to a variable. By default it saves the application element itself.

`Selector` - Contains all the data used to identify application.

`Timeout` - Time until the activity ceases if not successfully able to open the application.

`Width` - If set, moves the application to this horizontal offset.

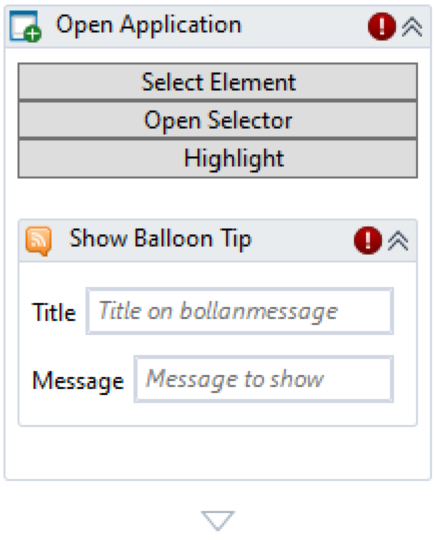


Fig. 14: **OpenRPA Open Application.**

Show Balloon Tip

Shows a message similar to the **Show Notification** activity, near the system tray.

Properties Parameters

`Duration` - How long the message appears.

`Message` - Message which is shown.

`Title` - Title of the balloon tip.

`Displayname` - Title of the activity inside the sequence.

`NotificationType` - Different notification types: information, error or warning.

Note: Refrain from using too many UI elements (such as this activity or Show Notification). UI elements are run single-threaded, which means that each time they are executed, they briefly stop workflow execution.

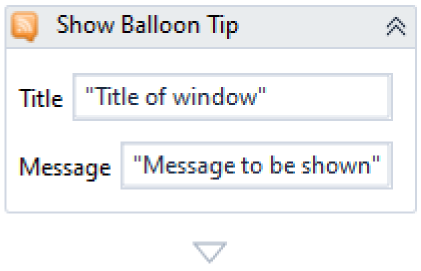


Fig. 15: **OpenRPA Show Balloon Tip.**

Type Text

Sends key strokes from the text contained inside **Hint** to the window currently focused. Instead of memorizing every key stroke, it is recommended to use the Recorder as it will map every key stroke automatically.[1]

The key stroke syntax should be "{{Modifier1, Key1} {Modifier2, Key2} {...} {ModifierN, KeyN}}", reminding the user that Modifiers, which would be auxiliary keys - such as `Ctrl`, `Alt`, `Shift` - are not obligatory.

The user can also send **variables** or **hardcoded text**, such as sending "{{Modifier1, Key1} SOME TEXT " + desired_variable + "{Enter}".

Properties Parameters

`Click Delay` - Delay between each keystroke.

`Displayname` - Title of the activity inside the sequence.

`Text` - Displayname inside the workflow designer.

Note: This activity requires a Keyboard IO. If it fails to find one, it will return "Could not send keyboard input. Error Code: 5". This may happen if OpenRPA Client was launched from inside a Windows Remote Desktop Session and the session was disconnected. In order to avoid this, please use HDRobots extension, do not disconnect/close the RDP session or persist the session using other techniques (ie. running VNC).

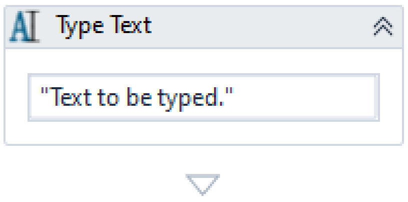


Fig. 16: **OpenRPA Type Text.**

[1] - Overview of TypeText Activity (`https://openrpa.openrpa.dk/pages/typetext-syntax`)

OpenRPA.AviRecorder

Coming soon - Work in progress

OpenRPA.Database

Coming soon - Work in progress

OpenRPA.Forms

Coming soon - Work in progress

OpenRPA.IE

Here belong the activities located inside the OpenRPA.IE toolbox.

`Get Element`
This activity is similar to OpenRPA.Get Element. It selects an element - or elements - filtered with the IE selector.

Clicking **Highlight** will red-highlight the first element that matches the criteria on the selector. Useful for checking if the selector is properly set up.

Properties Parameters

`Displayname` - Title of the activity inside the sequence.

`Elements` - the selected IE elements.

`From` - Allows search only within an element found within any of the IEElement's instances.

`LoopAction` - ?

`MaxResults` - Maximum elements to be found.

`MinResults` - Number of minimum results. If this value is higher than 0 and this activity fails to find an element, it throws an exception (`ElementNotFound`). If you want to check whether an elements exists or not, set this value to 0 and test for 'item.Length() == 0' inside the sequence.

`Selector` - Contains all the data to identify the IE application.

`Timeout` - Time until the activity ceases if not successfully able to find an element.

`Wait for ready` - If true, waits until the page is fully loaded.

Note: The user can also use **XPath Selector** to select elements inside the page. XPath Selectors use the following syntax:

- – `*` Matches any element node
- @* - Matches any attribute node
- node() - Matches any node of any kind

So, for example, if the user wanted to select all the child element nodes of a **bookstore** node, he'd use `/bookstore/*`. To select all elements in a document, he'd use `//*`. Now if he'd like to select all **title** elements which have **at least one** attribute of any kind, he'd use `//title[@*]`. To concatenate XPath expressions, the user can use the `| operator`. So, if he'd like to select all **title AND price** elements of all book elements, he'd use the expression `//book/title | //book/price`.[1]

[1] - XPath Syntax (`https://www.w3schools.com/xml/xpath_syntax.asp`).

Note: As well as the **XPath Selector**, the user can also use the **CSS Selector**. CSS Selectors use the following syntax:

- **.{CLASS}** - Matches all elements with that class.
- **#{CLASS}** - Matches all elements with that id.
- ** * ** - Matches all elements.
- **{ELEMENT}** - Matches all <p> elements.

The syntax is pretty similar to that of XPath Selectors. For more on CSS Selectors, refer to CSS Selectors (`https://www.w3schools.com/cssref/css_Selectors.asp`).

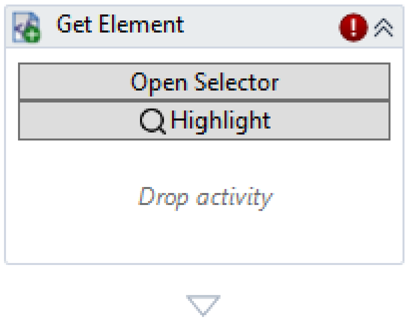


Fig. 17: **OpenRPA.IE Get Element.**

`Open URL`

Opens an URL in the currently selected InternetExplorer.Application instance. If there is none, it will automatically start a new Internet Explorer instance.

Properties Parameters

`Browser` - Which instance from the currently selected IE Elements will be used to open the desired URL.

`DisplayName` - Title of the activity inside the sequence.

`NewTab` - If set to `True`, it opens the desired URL in a new tab.

`Url` - The URL which the browser will navigate to.

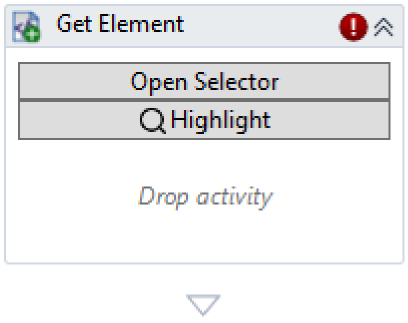


Fig. 18: **OpenRPA.IE Open URL.**

OpenRPA.Image

Here belong the activities located inside the OpenRPA.Image toolbox.

Note: Activities that use OCR to recognize text inside images must first have the language which OCR will use first on the **Settings tab**.

`Get Color`

Gets color at the specified coordinates inside an element provided by Image.GetElement.

Properties Parameters

`Displayname` - Title of the activity inside the sequence.

`Element` - Element from which the colors will be acquired.

`Result` - Variable to which to save the color.

`X Offset` - The horizontal offset inside the element from which to capture the color. Units are in pixels. It also allows for negative values.

`Y Offset` - The vertical offset inside the element from which to capture the color. Units are in pixels. It also allows for negative values.

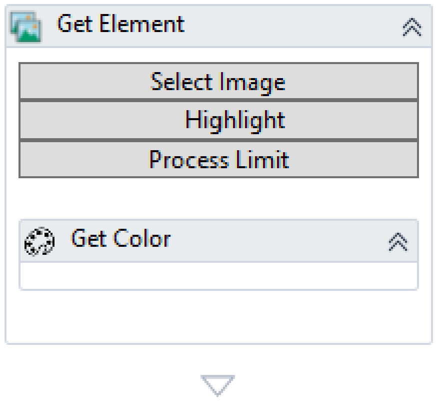


Fig. 19: **OpenRPA.Image Get Color.**

`Get Element`

Locates an image in screen or in a specific application window and creates an element from it. Use the **Select Image** button to select the are you want to capture or use the recorder.

The Recorder at first tries to create elements from controls in Windows Applications, Java Apps or Web elements, but as a last resort it will then make use of Image.GetElement to create elements by capturing the smallest unique images inside the are you drag while recording.

Clicking the **Highlight** button will red-highlight the first element that matches the criteria on the selector. It is useful for checking if the selector is properly set up.

Properties Parameters

`CompareGray` - If set to `True`, compare the image in grayscale to find the desired element. This parameter also saves performance.

`Displayname` - Title of the activity inside the sequence.

`Elements` - Element from which the image will be gathered.

`From` - Allows search only within an element found within any of the ImageElement's instances.

`Limit` - Limit the number of elements.

`LoopAction` - Loop the activity until something is found.

`MaxResults` - Number of maximum results.

`MinResults` - Number of minimum results. If this value is higher than 0 and this activity fails to find an element, it throws an exception (`ElementNotFound`). If you want to check whether an image exists or not, set this value to 0 and test for 'item.Length() == 0' inside the sequence.

`Processname` - Limits the element's process to a specific processname.

`Threshold` - Threshold used to find a matching image.

`Timeout` - Time until the activity ceases if not successfully able to find an element.

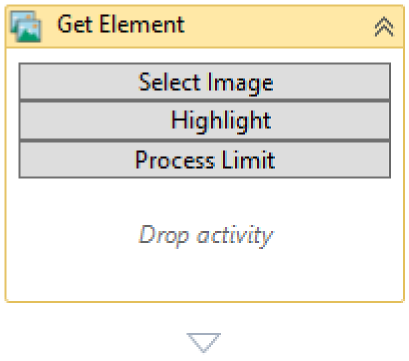


Fig. 20: **OpenRPA.Image Get Element.**

`Get Image`

Gets an image from the screen relative to another image or element found with Image.GetElement.

You must first use the **Image.Get Element** activity to select the image which this activity will act upon, then use the **Select Image** button to specify the relative area within the screen you want to capture. That is, the selection **Get Image** does is relative to another image selected with the **Image.Get Element** activity.

Clicking **Highlight** will green-highlight image selected relative to the other one, which will appear blue-highlighted. This is useful for checking if the selector is properly set up.

Properties Parameters

`Displayname` - Title of the activity inside the sequence.

`Element` - Element from which the image will be gathered.

`Height` - Height of the image to be captured.

`OffsetX` - The horizontal offset inside the element from which to capture the image. Units are in pixels. It also allows for negative values.

`OffsetY` - The vertical offset inside the picture from which to capture the image. Units are in pixels. It also allows for negative values.

`Result` - Variable to which the image element will be saved.

`Width` - Width of the image to be captured.

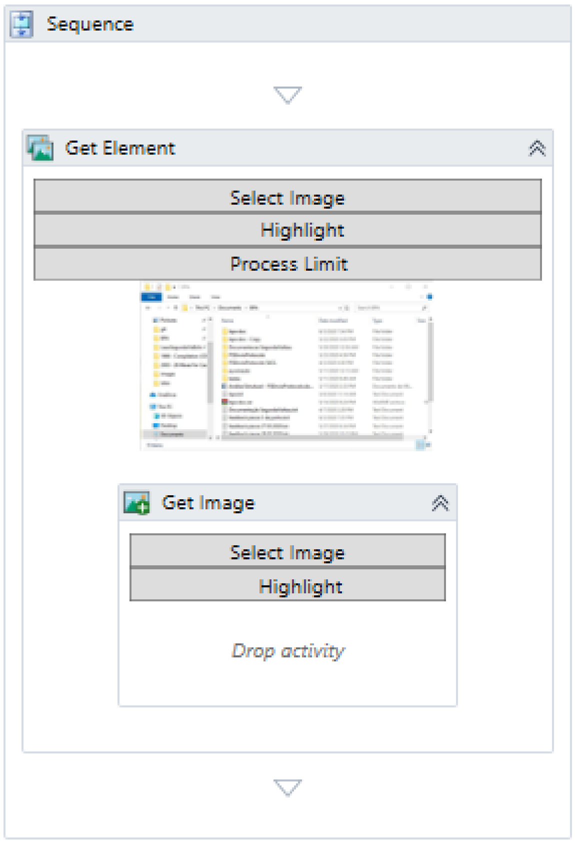


Fig. 21: **OpenRPA.Image Get Image.**

`Get Text`

Gets text from an image using OCR, can be limited to a specific word or sentence. It can also be used to split or limit text gathered from other image-related activities.

The `Limit to` or `WordLimit` parameter is used to limit the output of the OCR processing. It is useful if you want to limit the output of `Get Text` to a given set of words.

Properties Parameters

`CaseSensitive` - Differentiate between lowercase and uppercase letters.

`Displayname` - Title of the activity inside the sequence.

`Element` - Element from which the image will be gathered.

`Result` - Variable to which the text will be saved.

`WordLimit` - Max. numbers of words to gather.

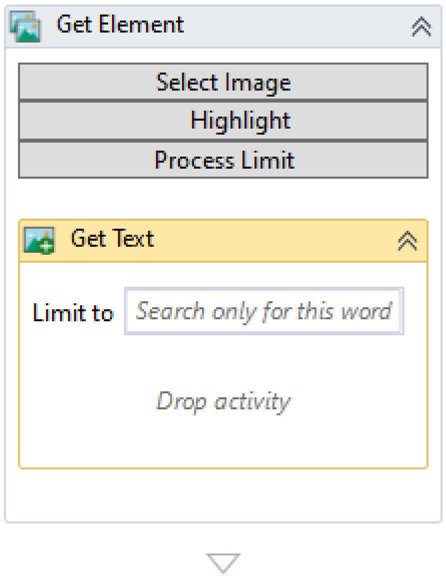


Fig. 22: **OpenRPA.Image Get Text.**

`Load From File`

Loads an image from a file and returns it as an ImageElement.

It also uses OCR Image Recognition automatically to capture the text inside the screenshot, and saves it inside the result's `ImageElement.value` attribute. The extensions allowed are: `gif`, `jpg`, `jpeg` and `png`, that is, only image extensions.

Properties Parameters

`Displayname` - Title of the activity inside the sequence.

`Filename` - Filename of the file that will be used to acquire the image from.

`Result` - Variable to which to save the ImageElement.

Note: Note: OCR Image Recognition is done automatically upon the image and you may access the resulting text by accessing `ImageElement.value` property.

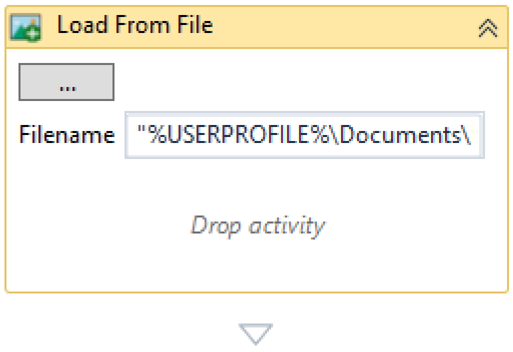


Fig. 23: **OpenRPA.Image Load From File.**

`Take Screenshot`

Takes a screenshot of the entire screen or of a specific element found by using Image.GetElement and returns it as an ImageElement.

Properties Parameters

`Displayname` - Title of the activity inside the sequence.

`Element` - Name of the element from which the screenshot will be captured.

`Height` - Height of the screenshot.

`Result` - Variable to which to save the screenshot.

`Width` - Width of the screenshot.

`X` - The horizontal offset inside the picture from which to capture the screenshot.

`Y` - The vertical offset inside the picture from which to capture the image.

Note: Note: OCR Image Recognition is done automatically upon the image and you may access the resulting text by accessing `ImageElement.value` property.

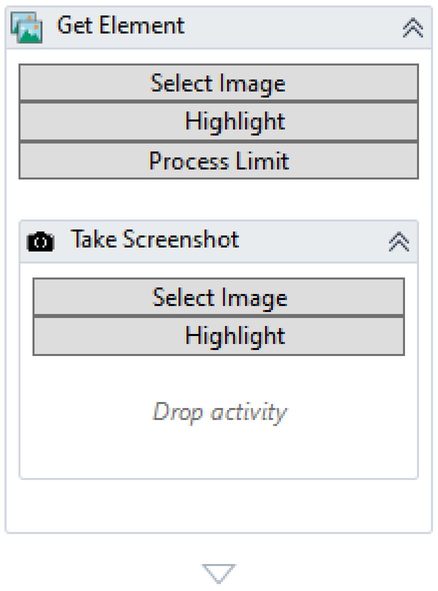


Fig. 24: **OpenRPA.Image Take Screenshot.**

OpenRPA.Windows

Here belong the activities located inside the OpenRPA.Windows toolbox.

`Get Element`

It selects an element - or elements by using the **Open Selector** button or automatically through recording.

You can use the **Highlight** button to highlight the selected area inside the screen. Clicking the **Highlight** button will red-highlight the first element that matches the criteria on the selector. It is useful for checking if the selector is properly set up.

Properties Parameters

`Displayname` - Title of the activity inside the sequence.

`Elements` - Element from which the image will be gathered.

`From` - Allows search only within an element found within any of the Element's instances.

`MaxResults` - Number of maximum results.

`MinResults` - Number of minimum results. If this value is higher than 0 and this activity fails to find an element, it throws an exception (`ElementNotFound`). If you want to check whether an element exists or not, set this value to 0 and test for 'item.Length() == 0' inside the sequence.

`Selector` - Contains all the data pertaining to the application selected.

`Timeout` - Time until the activity ceases if not successfully able to find an element.

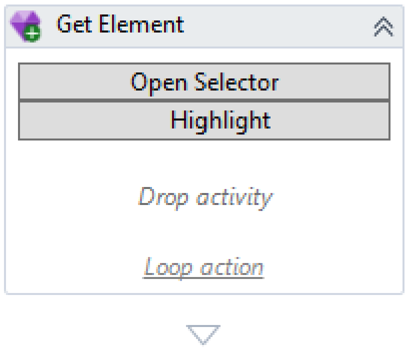


Fig. 25: **OpenRPA.Windows Get Element.

System.Activities

Here belong the activities located inside the OpenRPA.System.Activities toolbox.

`Delay Activity`

Creates a delay from an instance of an object from the `TimeSpan` class. You can simply type a length of time in the `HH:MM:SS`.

Properties Parameters

`Displayname` - Title of the activity inside the sequence.

`Duration` - The argument specifying the length of the delay. Can be a simple 00:00:00.00 {hh:min:ss.mili} or an object from the TimeSpan class.

Note: A **TimeSpan** value represents a time interval and can be expressed as a particular number of days, hours, minutes, seconds, and milliseconds. Because it represents a general interval without reference to a particular start or end point, it cannot be expressed in terms of years and months, both of which have a variable number of days.[1]

[1] - TimeSpan Struct (`https://docs.microsoft.com/en-us/dotnet/api/system.timespan?view=netcore-3.1`)

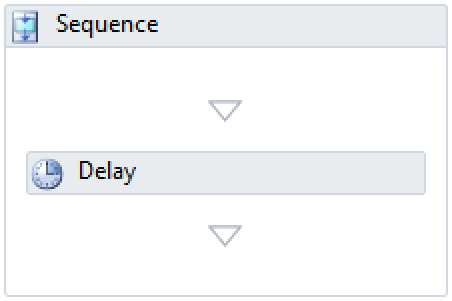


Fig. 26: **System.Activities Delay.**

`Do While`

Creates a loop that executes other activities dropped in it for at least once and repeatedly until **looping condition** no longer evaluates to `True`.

Properties Parameters

`Condition` - This is a VB Expression that represents the **looping condition**.

`Displayname` - Title of the activity inside the sequence.

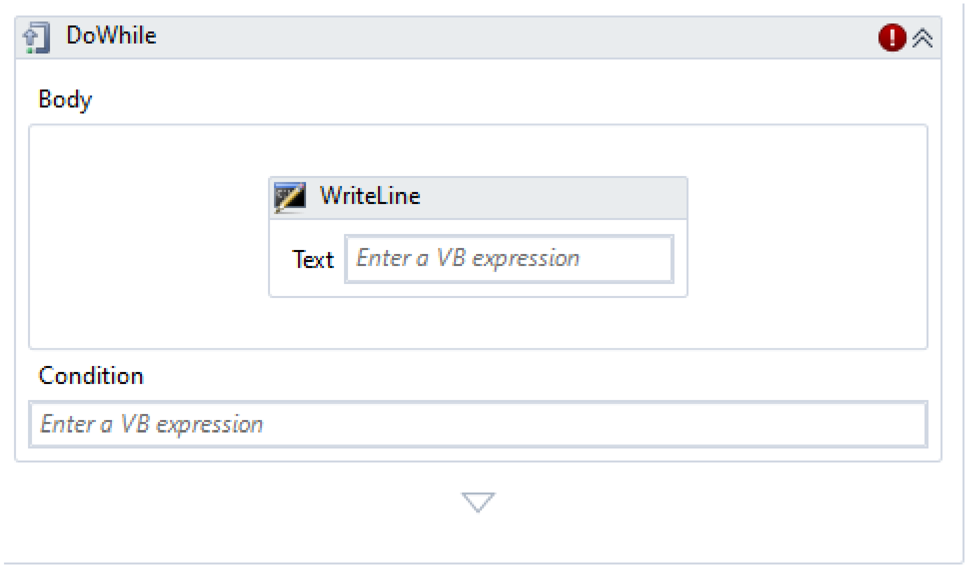


Fig. 27: **System.Activities Do While.**

`If Activity`

It is similar to conditionals in programming, if the **clause condition** is `True`, then it executes the activities in the `Then` block. Else, if it is `False` it executes the `Else` block.

Properties Parameters

`Condition` - This is a VB Expression that represents the **clause condition**.

`Displayname` - Title of the activity inside the sequence.

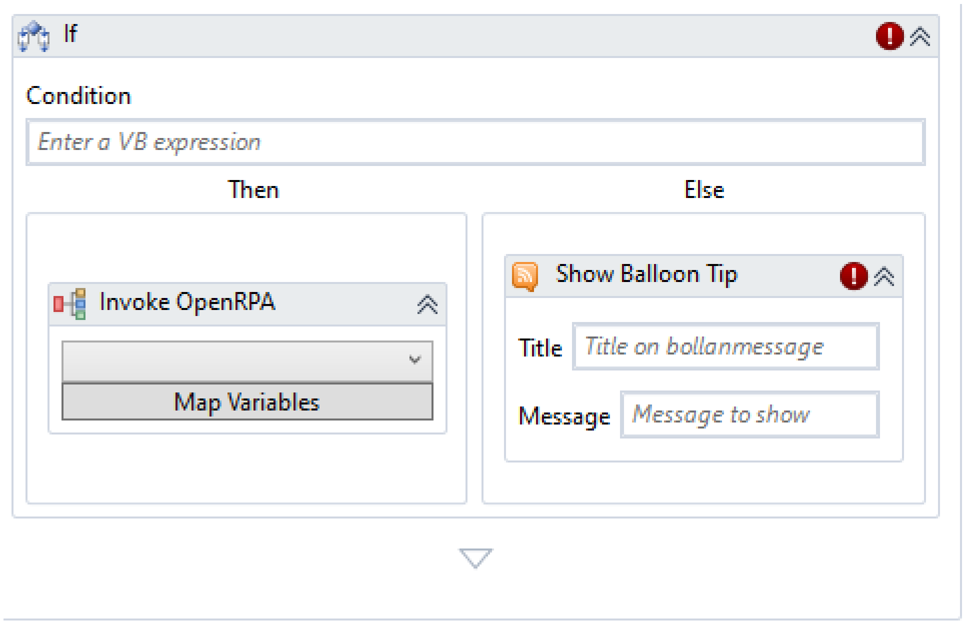


Fig. 28: **System.Activities If.**

`Sequence Activity`

This activity is used to nest sequences upon the main sequence. You can drop activities inside it just as you would do inside the main sequence. It is helpful to organize your code/workflow because you can expand or collapse them.

Properties Parameters

`Displayname` - Title of the activity inside the sequence.

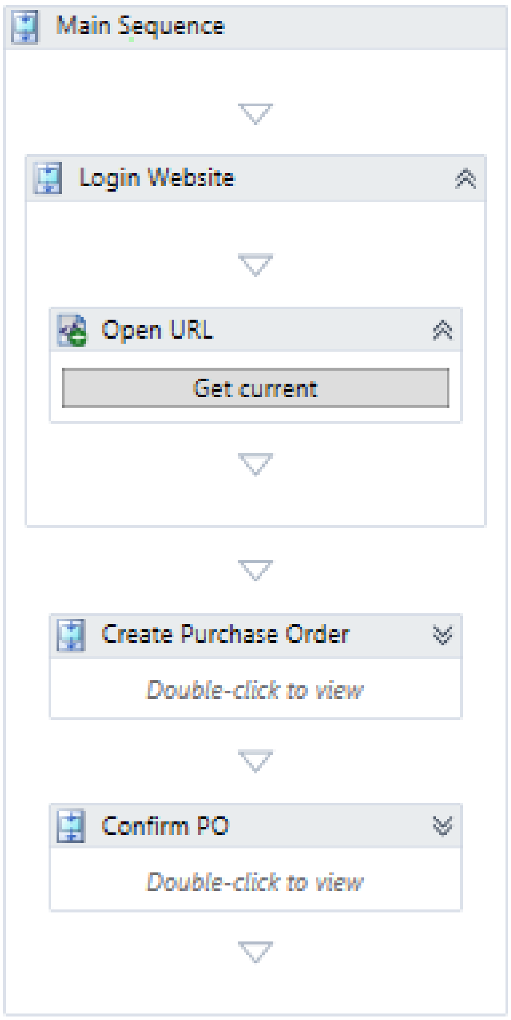


Fig. 29: **System.Activities Sequence.**

`Throw Activity`

Provides a method to terminate the execution of the workflow in a certain case and end it, displaying an error message. For people used to programming, that would be known as throwing an exception. Upon entering the exception in the `Exception` field, type `new` to exhibit all the available ones.

Properties Parameters

`Displayname` - Title of the activity inside the sequence.

`Exception` - The exception which will appear on the error message.

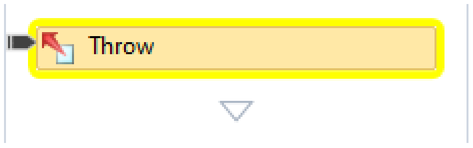


Fig. 30: **System.Activities Throw.**

`TryCatch Activity`

Tries to execute a sequence of activities within the `Try` block, if any of them causes an error, the error is catched and then the activities inside the `Catches` block are executed. At the end of execution, the `Finally` block is executed.

Properties Parameters

`Displayname` - Title of the activity inside the main outer sequence.

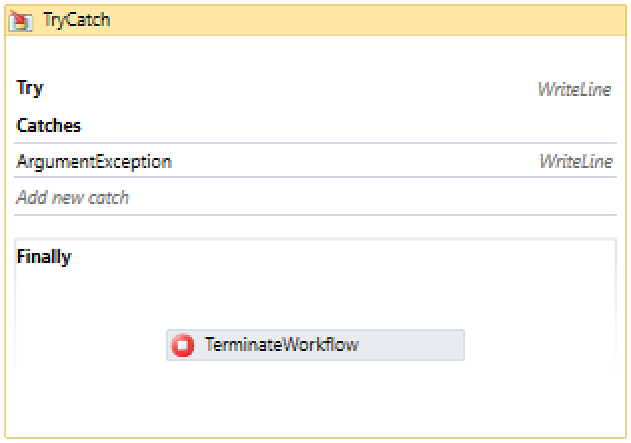


Fig. 31: **System.Activities TryCatch.**

`While Activity`

Creates a loop that executes other activities dropped in it while the **condition** is fulfilled - ie. `False`. Once it is fulfilled, ie. `True`, the loop ceases and moves to the next activity. Differently from the **Do While** activity, it obligatorily checks whether the condition is `True` before entering the sequence.

Properties Parameters

`Condition` - This is a VB Expression that represents the **condition**.

`Displayname` - Title of the activity inside the sequence.

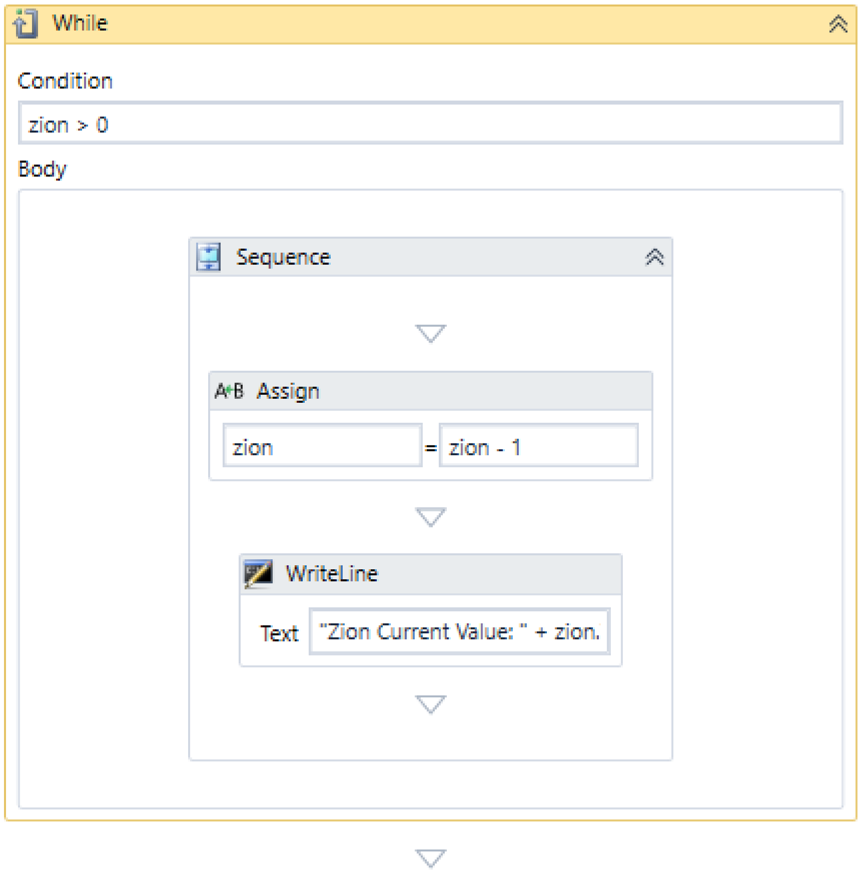


Fig. 32: **System.Activities While.**

`WriteLine Activity`

Writes output to the console according to the `Text` string.

Properties Parameters

`Displayname` - Title of the activity inside the sequence.

`Text` - The string which will be printed on the console.

`TextWriter` - **Optional** - The writer used to write the character stream.

Note: The TextWriter class is an abstract class. Therefore, you do not instantiate it in your code. The StreamWriter class derives from TextWriter and provides implementations of the members for writing to a stream. The following example shows how to write two lines that consist of string values to a text file by using the WriteLineAsync(String) method.[1]

---

[1] - TextWriter Class (`https://docs.microsoft.com/en-us/dotnet/api/system.io.textwriter?view=netcore-3.1`)

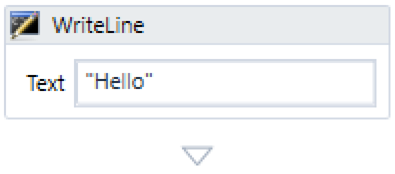
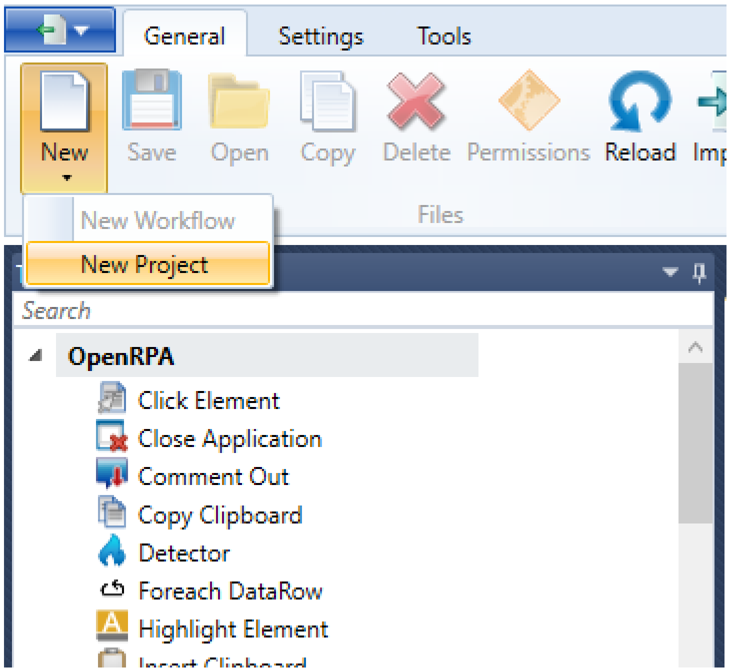


Fig. 33: **System.Activities Write Line.**

3.6 Creating a new Project

3.6.1 Create New Project

To create a new project on OpenRPA simply click in **New > New Project** at the Ribbon. It will then open an empty workflow named **New Workflow**.



3.6.2 Granting permissions to users/roles

To set the permissions for the current workflow or project, whichever is selected, simply click in the Permissions button at the Ribbon. It opens the **Permissions Window** with the names of the current users for the workflow opened inside the workspace. There are 4 types of permission available which will be discussed below.

Read

User has access to execute and view the content of the workflow.

Update

User has access to update and view the content of the workflow.

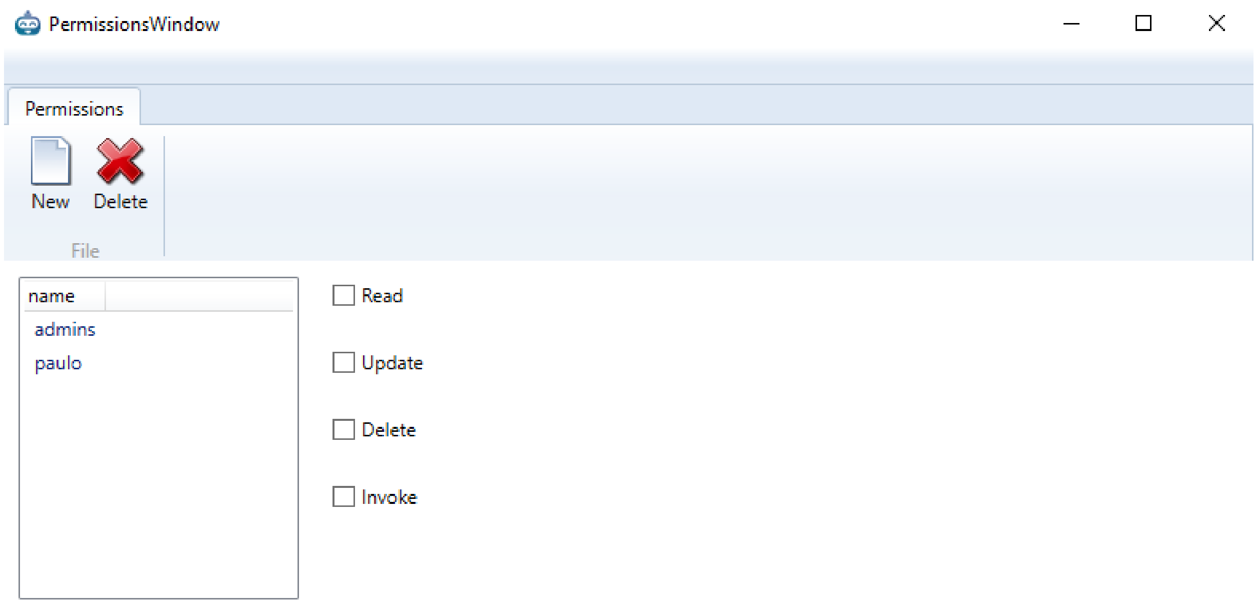
Delete

User has access to delete the workflow or group of workflows from the workspace.

Invoke

User has access to invoke the content of the workflow.

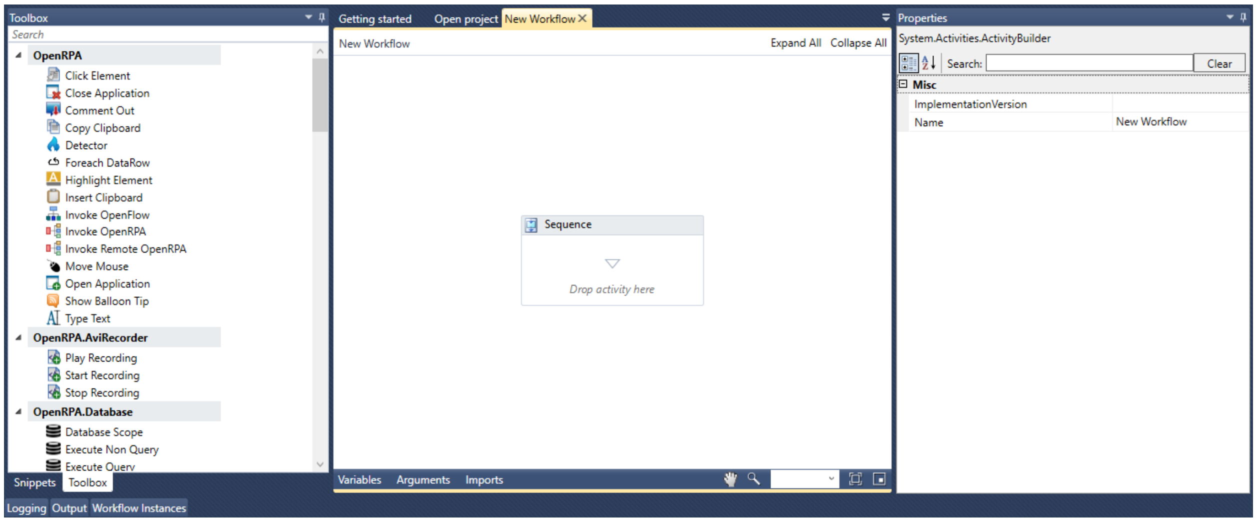
To set these permissions simply click in the user you want to grant or revoke permissions and click the according radio button. If the user is available at the OpenFlow repository but not in the current workflow, simply click the new button and type his username, a dropdown will come up with suggestions for the name of the user you are looking for. Select the user and click OK.



3.6.3 New Workflow

Immediately after clicking **New Project** inside the **New** ribbon, a new empty workflow appears on the screen along with a single sequence containing the caption *Drop activity here*. This is where you are going to add activities and build and implement the BPM process you've mapped.

If you are connected to a web socket, your workflows are automatically inside OpenFlow's repository. The OON stack takes care of handling the workflow storage, deploying the workflows and repository control versioning.



3.6.4 Using the Designer

Coming soon - work in progress

3.6.5 Running the Workflow

Coming soon - work in progress

3.7 Recorder

Coming soon - Work in progress

3.8 Selector

Coming soon - Work in progress

3.8.1 Returning multiple elements

Coming soon - Work in progress

3.8.2 Custom CSS

Coming soon - Work in progress

3.9 Detectors

Detector is the main `Event Listener` inside OpenRPA - refer to Event listener (https://www.computerhope.com/jargon/e/event-listener.htm) if you don't know what that means. Simply put, it is an activity that waits for something to happen either before the Workflow continues processing and executing its functions.

To set a **Detector**, you must first define it inside the Detector's settings (`Settings inside the main Ribbon -> Detectors -> Add Detector`).

There are also different parameters which can be set for each type of detector which are going to be further discussed.

Screenshots on How to set up Detectors

After that, you go back to your sequence, define a **Detector activity** inside it and select the Detector you want to use as your `Event Listener`. The `Detector` will automatically start the process for which it is defined, if it is inside a `State Machine` or continue it's process, if inside a `Flowchart` or `Sequence`.

3.9.1 FileWatcher

The `FileWatcher` detector checks for file changes inside a given path.

You can set a `Name` for the detector.

`Path` refers to the absolute path the folder in which the files are located.

`File filter` is set to filter any files with a specific given filename.

3.9.2 KeyboardDetector

The `KeyboardSequence` detector listens for a specific - or sequence of - key strokes.

You can set a `Name` for the detector.

`Process restriction` refers to the process upon which this won't be listened to if their interface is the main screen.

`Keys` are the keys which enable the Detector. To set them, use the button `Set keys`.

3.10 Running Workflows from Command Line

Coming soon - Work in progress

3.11 Workflow Examples

3.11.1 Web Automation

In this example, the RPA Input Forms Challenge (`http://www.rpachallenge.com/`) will be solved.

As recommended, the Recorder will be used to facilitate the designing process.

The steps are listed below.

1. Download the `Excel file`.

2. Read the `Excel file`.
3. Map Start button.
4. Loop the DataSet rows.
5. Find the respective handle for each WebElement input field.
6. Fill all the required fields inside the webpage.
7. Map Submit button.

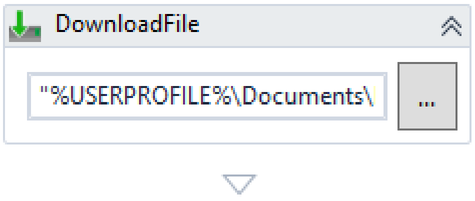
Download the Excel file

To download the `Excel file`, use the **DownloadFile** activity, found inside the `OpenRPA.Utilities` toolbox.

Drag the activity to the main sequence inside the Designer.

Click the `...` button in it to select the path of file to which the archive will be saved or assign the `LocalPath` parameter inside the Properties tab of the activity.

Paste the URL from the archive which you want to fetch. It is useful to remind that, since the URL will be acquired by OpenRPA as a `string`, it must be contained within `"` quotes.



Read the Excel file

The ReadExcel activity will read from an `Excel file` and save it into a variable.

All the steps here are visibly explained at the **How to read from an ``Excel file``**.

How to read from an Excel file

Drag the **ReadExcel** activity, found inside the `OpenRPA.Utilities` toolbox, to the main sequence.

Select the `Excel file` which will be read by clicking the `...` button and find the filepath in explorer where the sheet you want to read from is found.

Then, by clicking the `...` button, the user will be prompted by an explorer window and there select the `Excel file` that should be used.

The **ReadExcel** activity will read this `Excel file` and save it into a variable.

As this variable does not exist yet, it must first be created. A reminder that variable must be obligatorily set with the type **System.Data.DataSet**. Simply click in the `DataSet` input field inside the **Properties tab**, type the name of the variable you want to use and press `Ctrl+K`.

Now the user may either create the variable inside the **Variables & Arguments box** or follow the note below.

Note: While creating a new variable in the **Properties box**, after typing the variable name and with the caret still inside the input field, by pressing `Ctrl+K` OpenRPA will automatically create this new variable with the correct type required for the activity.

Note: A `DataSet object` corresponds to the whole sheet. Each `DataTable` (eg. `Sheet1`) accordingly corresponds to each sheet inside that `Excel file`. The `DataSet` class is the object temporarily - the in-memory cache - created to store all the data consumed from this `Excel file`. For more on the `DataSet` class, refer to the link to Microsoft's Documentation on the DataSet Class (`https://docs.microsoft.com/en-us/dotnet/api/system.data.dataset?view=netcore-3.1`).

Note: A `DataTable object` corresponds to one table of in-memory data contained in a `DataSet object` from the class equally named. When you read an `Excel file`, it is automatically assigned to a `DataSet` class and all the `Sheets` are saved into `DataTable` classes - numbered from 0 to the quantity of sheets inside the `Excel file`. Here it is reminded that `DataTable` objects are case sensitive.

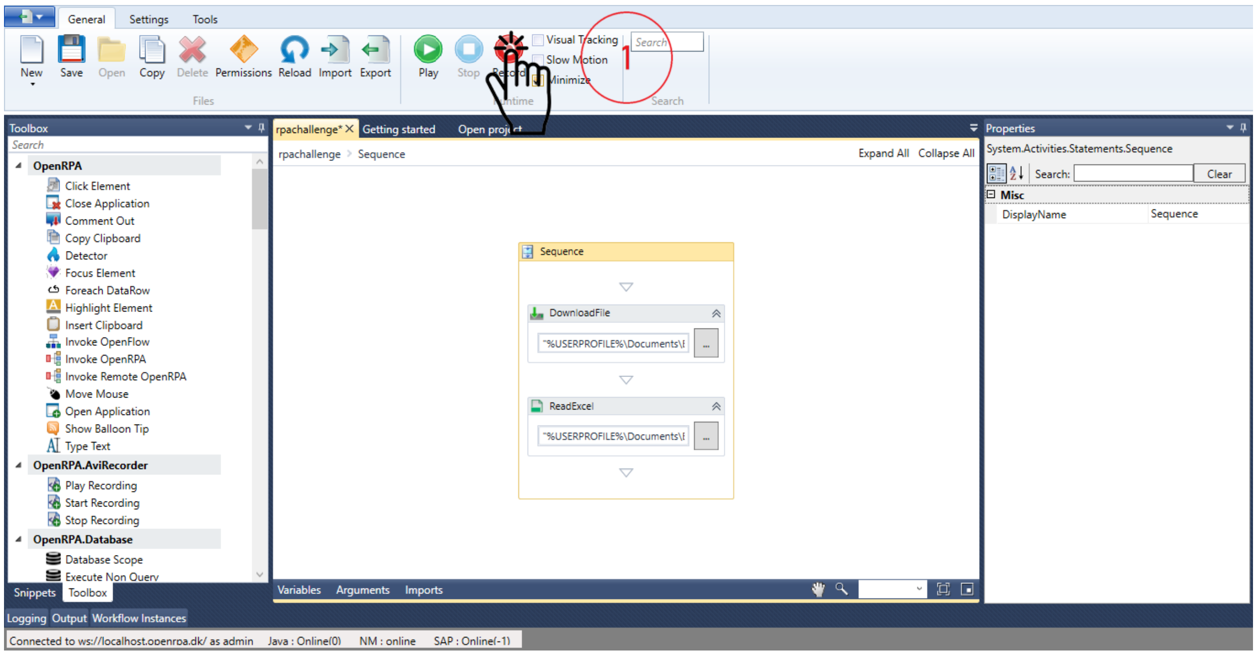
Map the Start button

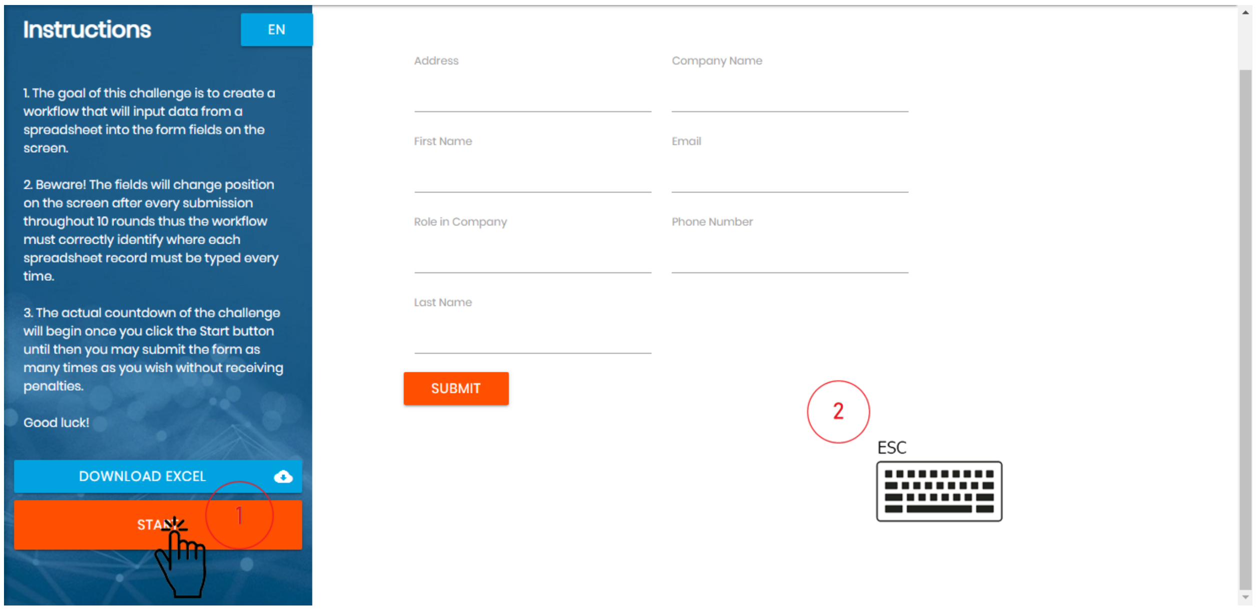
The challenge starts by clicking the **Start button**.

To map the process of starting the challenge, simply click the white triangle corresponding to the space below the **Read Excel** activity, start the Recorder and click the **Start** button inside the browser, as shown in the steps below.

In order to cease recording, simply press the `ESC` button.

It is reminded here that ending the recording is obligatory, else it will record all activities you execute endlessly.





Instructions EN

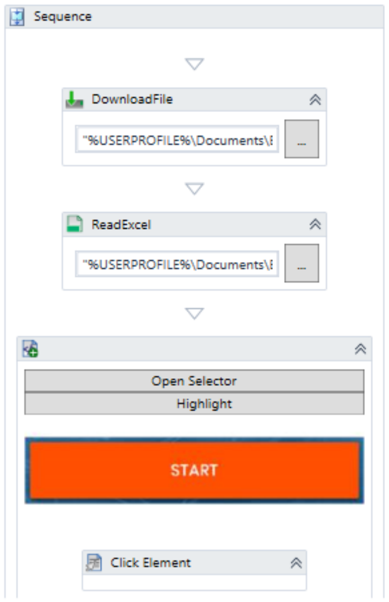
1. The goal of this challenge is to create a workflow that will input data from a spreadsheet into the form fields on the screen.

2. Beware! The fields will change position on the screen after every submission throughout 10 rounds thus the workflow must correctly identify where each spreadsheet record must be typed every time.

3. The actual countdown of the challenge will begin once you click the Start button until then you may submit the form as many times as you wish without receiving penalties.

Good luck!

After successfully mapping the **Start** button, your workflow should look like the last image on the next page.



Loop the DataSet rows

Now it is time to populate the input forms inside the page with the data we've gathered from the `Excel file`.

Drag the **Foreach DataRow** activity, found inside the `OpenRPA` toolbox. Then inside the **Foreach DataRow** activity, drag a **Sequence** activity, from the `System.Activites` toolbox.

As stated before, each sheet inside the `DataSet` assigned from the **Read Excel** activity is saved in-memory as a collection inside the `Tables` attributes.

Each sheet inside the `Tables` attribute is acquired accordingly to the sheets inside the `Excel file`, these sheets are numbered starting from 0. As an example, if you were to add a row to a `DataTable` inside that

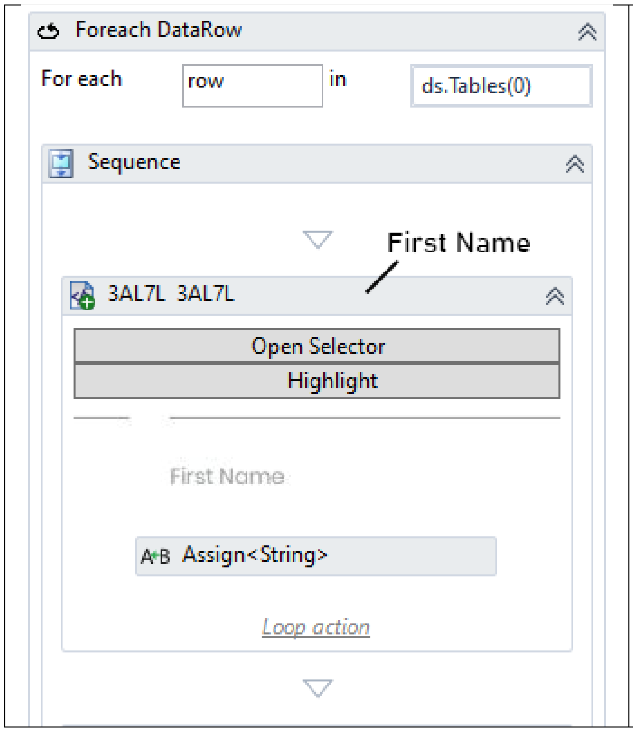
`Excel file`, you'd first assign it to a `DataTable class` object and use the activity to add that row to the table.

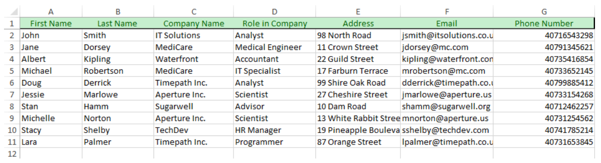
You can either acquire each row by assigning a single variable to the one you want to capture, numbered starting from 0, or you can use the **Foreach DataRow** to automatically loop through each row accordingly. Since in our case the `DataSet` is assigned to the `ds` variable, that is the one which will be used to retrieve the `Table` from. Since there is only one sheet, the `ds.Tables(0)` is used to assign the `Table` from where the **Foreach DataRow** will retrieve the values.

After assigning the `DataTable` where the rows are going to be retrieved from, it is noted here that the columns are mapped to attributes inside the variable assigned to the left-hand side of the **Foreach DataRow** activity - in our case `row` to return or set the value of a `DataColumn`.

The **Foreach DataRow** loop basically iterates through every value in the row (`First Name`, `Last Name`, `Company Name`, ...) and assign it to the `XPath` selector item inside the webpage where each field belongs. This causes the value inside the RPA Input Forms Challenge (`http://www.rpachallenge.com/`) page to be populated.

Note: A `DataRow object` refers to each row mapped from the `DataTable`. By using the `DataRow` object and its properties and methods the user is able to retrieve and evaluate; insert, delete and update the values inside the `DataTable`. The `DataRowCollection` also represents the actual `DataRow` objects inside the `DataTable`. Each column mapped from it can be retrieved by using its index as an attribute of the `DataRow` class object, if you'd like to retrieve the 1st column inside the row, you'd use `row(0)`. For more on the `DataRow` class, refer to the link to Microsoft's Documentation on the DataRow Class (`https://docs.microsoft.com/en-us/dotnet/api/system.data.datarow?view=netcore-3.1`).





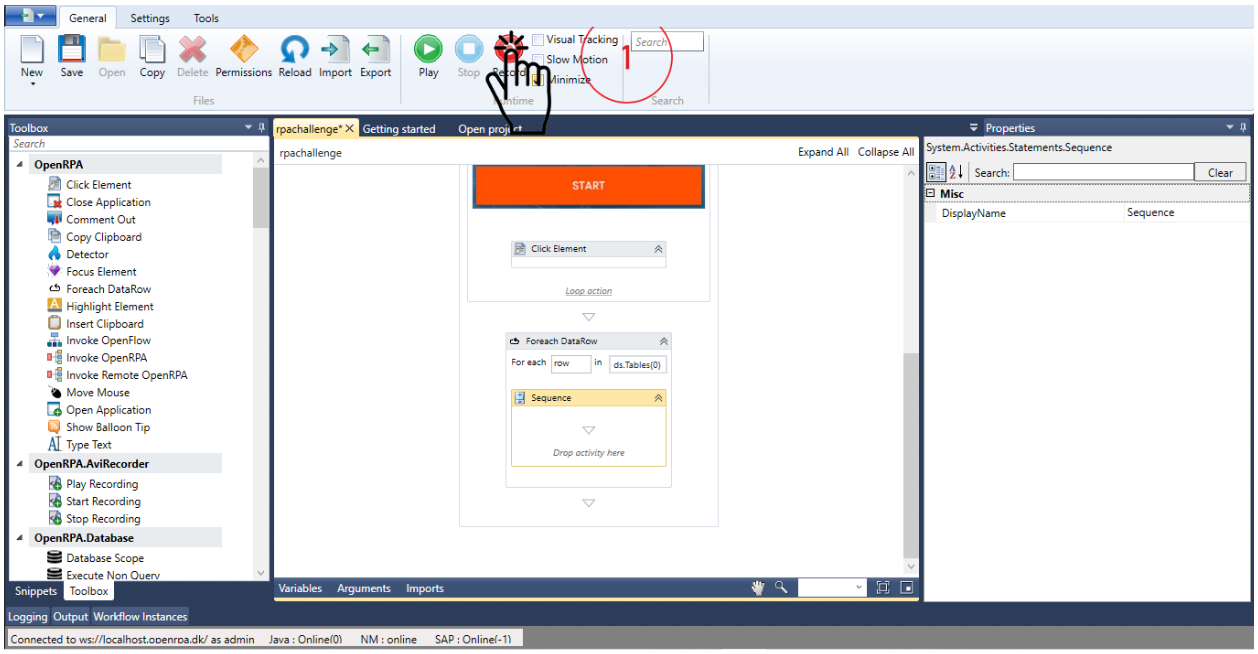
	A	B	C	D	E	F	G
1	First Name	Last Name	Company Name	Role in Company	Address	Email	Phone Number
2	John	Smith	IT Solutions	Analyst	98 North Road	jsmith@itsolutions.co.u	40716543298
3	Jane	Dorsey	MediCare	Medical Engineer	11 Crown Street	jdorsey@mc.com	40791345621
4	Albert	Kipling	Waterfront	Accountant	22 Guild Street	kipling@waterfront.con	40735416854
5	Michael	Robertson	MediCare	IT Specialist	17 Farburn Terrace	mrobertson@mc.com	40733652145
6	Doug	Derrick	Timepath Inc.	Analyst	99 Shire Oak Road	dderrick@timepath.co.u	40799885412
7	Jessie	Marlowe	Aperture Inc.	Scientist	27 Cheshire Street	jmarlowe@aperture.us	40733154268
8	Stan	Hamm	Sugarwell	Advisor	10 Dam Road	shamm@sugarwell.org	40712462257
9	Michelle	Norton	Aperture Inc.	Scientist	13 White Rabbit Stree	mnorton@aperture.us	40731254562
10	Stacy	Shelby	TechDev	HR Manager	19 Pineapple Bouleva	sshelby@techdev.com	40741785214
11	Lara	Palmer	Timepath Inc.	Programmer	87 Orange Street	lpalmer@timepath.co.u	40731653845
12							

Find the respective handle for each WebElement input field

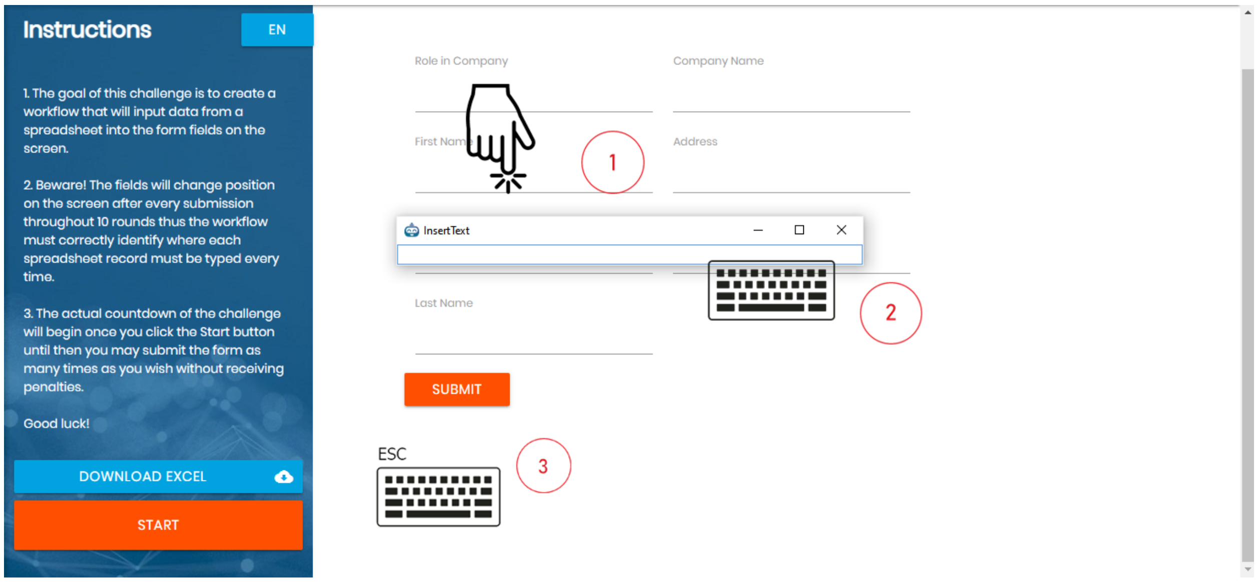
By using the Recorder and clicking on each input field found in the RPA Input Forms Challenge (`http://www.rpachallenge.com/`) page, it is possible to easily find the best handles to each WebElement being clicked.

As this workflows deals with WebElements manipulation, usually the best handles are Element.ID, Element.Name, Element.ClassName and so on. In our case, the `ng-reflect-name` attribute is used along with the `XPath` selector to provide a means of capturing a given element.

Once you start recording, click a single input field inside the RPA Input Forms Challenge (`http://www.rpachallenge.com/`) page. As usual, we recommend capturing the first field - that is - `First Name` and then proceeding to the others.



After clicking an input field, a prompt window will appear with the title `InsertText`, simply press `Enter`.



1. The goal of this challenge is to create a workflow that will input data from a spreadsheet into the form fields on the screen.

2. Beware! The fields will change position on the screen after every submission throughout 10 rounds thus the workflow must correctly identify where each spreadsheet record must be typed every time.

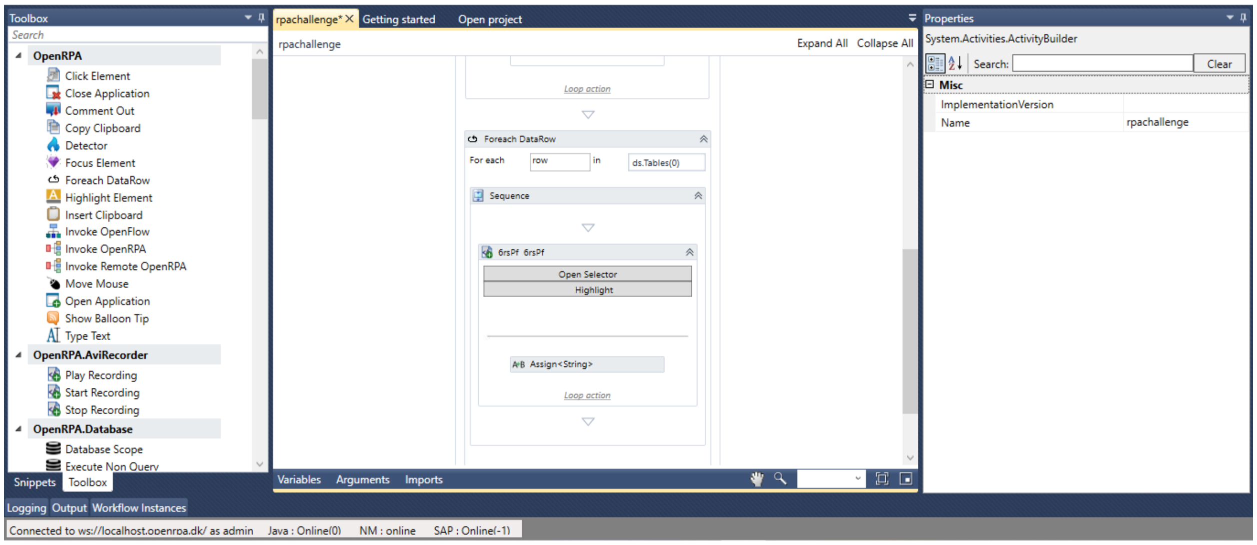
3. The actual countdown of the challenge will begin once you click the Start button until then you may submit the form as many times as you wish without receiving penalties.

Good luck!

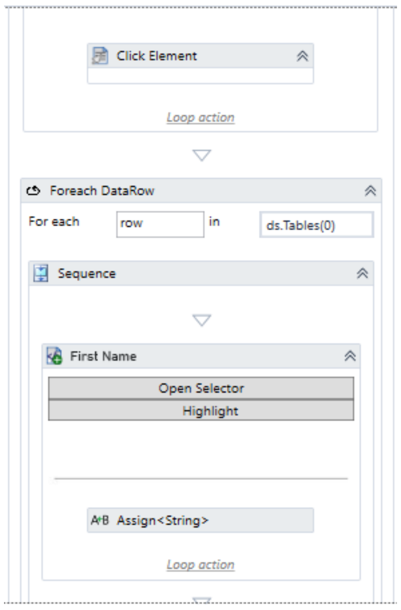
In order to cease recording of the current input field, simply press the `ESC` button.

It is reminded here that ending the recording is obligatory, else it will record all activities you execute endlessly.

Then, a new `OpenRPA.NM.GetElement` activity will appear inside the **Foreach DataRow** sequence.

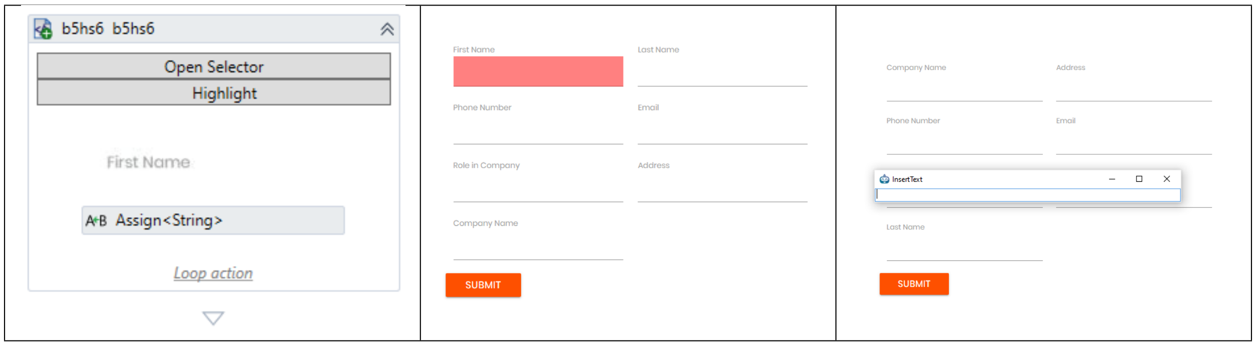


It is desirable that the user change the name of each activity to the corresponding input field. As shown in the image below.



Note: To change the name, or `DisplayName` - as set inside the activity's parameters, the user can either click twice in the name of the activity, type the desired name and press `RETURN` or set the parameter `DisplayName` inside the **Properties box**.

Note: When a WebElement is clicked at using the Recorder and OpenRPA determines that the WebElement is capable of receving input values, automatically a prompt will appear where the user can type string texts and then the Recorder will generate the activities as if it were wanted to fill these WebElements with the respective string texts provided. For now, the user may ignore these and fill the prompt with any text, these assigned values will be replaced so they get their values from the DataTable `ds`. The WebElements are automatically assigned to the `item` variable.



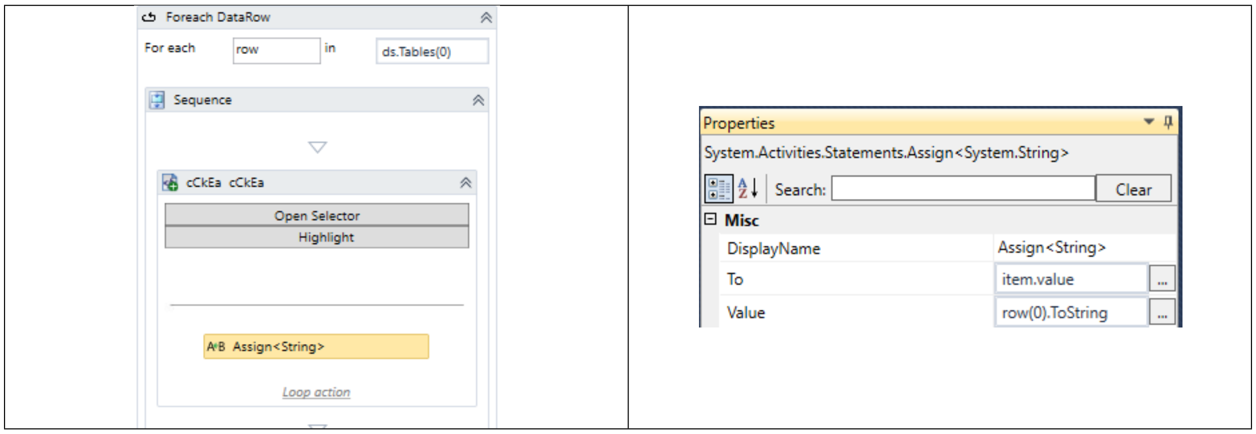
Fill all the required fields inside the webpage.

Now that the handles are mapped, assign the values of the rows, columns that were obtained in the Excel sheet.

The first input field is `First Name`, and the assignment can be done by using the **Assign** activity found in the `RPA.Utilies` toolbox.

In the left hand side, use the `item.Value` to access the property of the `WebElement` and in the righthand side use the desired input value - in the specific case of the `First Name` column, `row(0).ToString`.

`FirstName` is located in the first column, hence `index = 0`.



System.Activities.Statements.Assign<System.String>

DisplayName	Assign<String>
To	item.value
Value	row(0).ToString

Note: The **Assign** activity is used to reference a value to a variable. To do that, enter in the left-hand side of the activity the name, or alias, of the variable which you want to map a value to. If desired, you can press `Ctrl+K` to automatically create the variable or create it manually. Use the right-hand side to enter the value which will be assigned to the variable.

Note: The value of the respective column inside the row is a `DataCell` object, an object temporarily - the in-memory cache - created to store the column value inside the respective row. Furthermore, as the Excel value may be an object of non-string type, it's better to be safe and always cast it to a string. Since the need to cast - ie. change its type - to a `String` before assigning it to the `item.Value`, given that the DOM object returned by the `Selector` is of type `String` as well. This is done by using the `ToString` method.

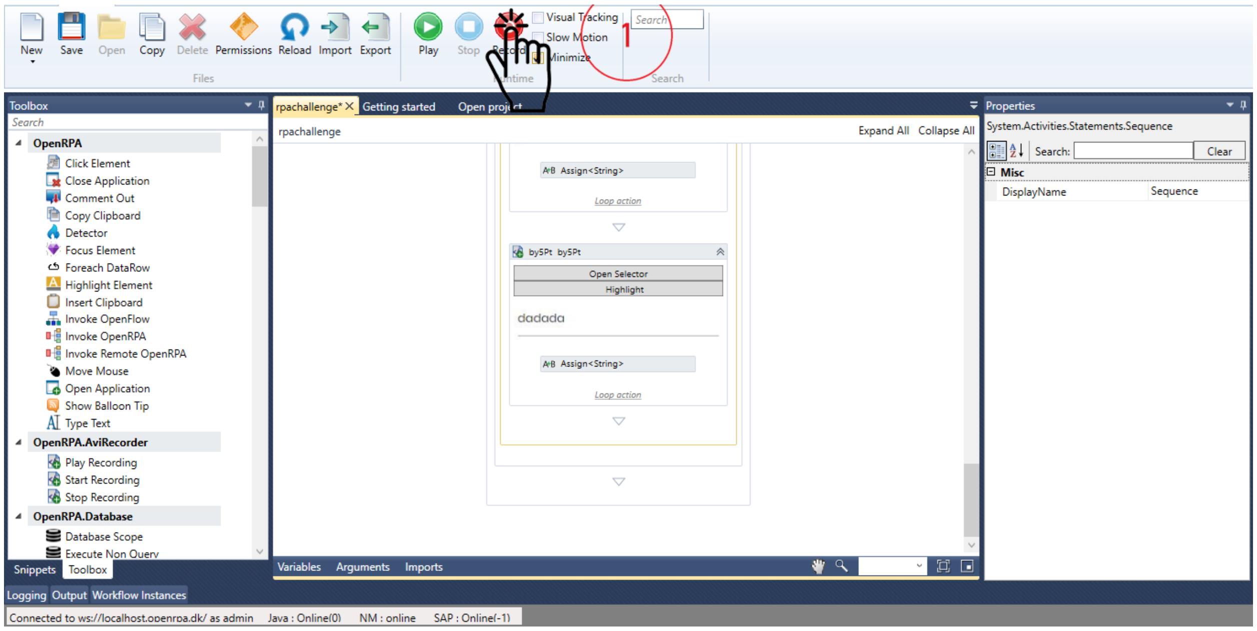
Map Submit button

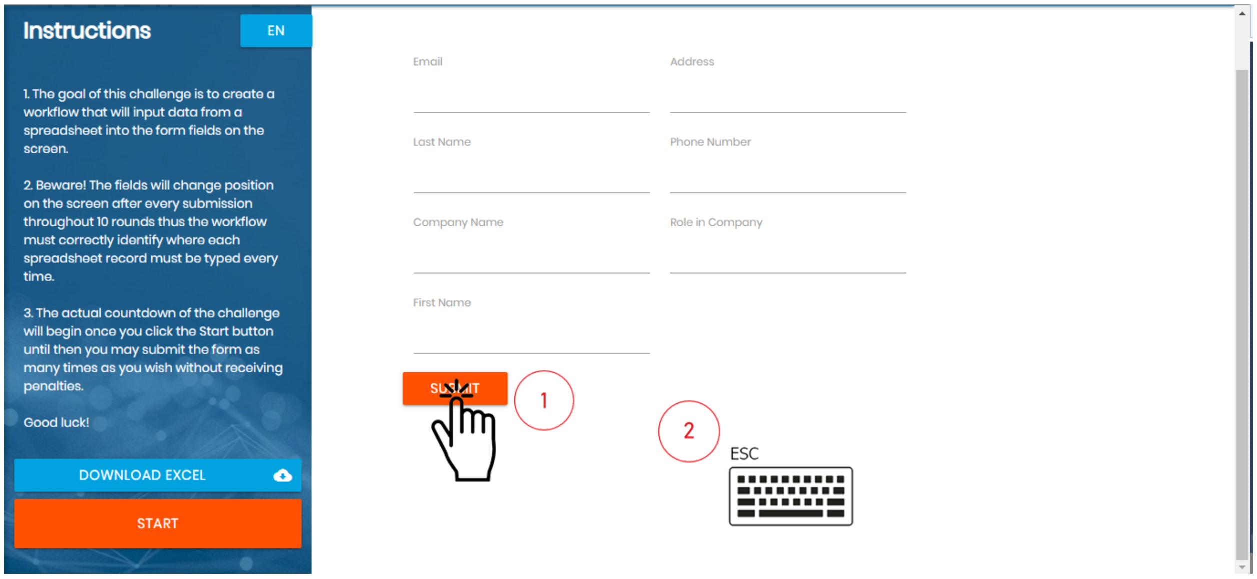
Each entry - corresponding to each row - must be entered after being filled by clicking the **Submit** button.

To map the process of entering each input - ie. clicking the **Submit button** -, simply click the white triangle corresponding to the space below the last **Assign** activity, start the Recorder and click the **Submit** button inside the browser, as shown in the steps below.

In order to cease recording, simply press the `ESC` button.

It is reminded here that ending the recording is obligatory, else it will record all activities you execute endlessly.





Instructions

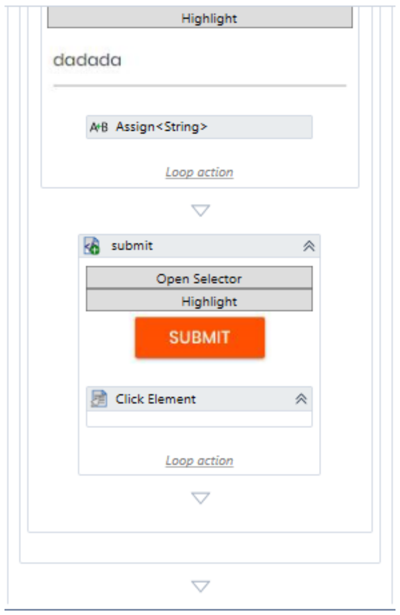
1. The goal of this challenge is to create a workflow that will input data from a spreadsheet into the form fields on the screen.

2. Beware! The fields will change position on the screen after every submission throughout 10 rounds thus the workflow must correctly identify where each spreadsheet record must be typed every time.

3. The actual countdown of the challenge will begin once you click the Start button until then you may submit the form as many times as you wish without receiving penalties.

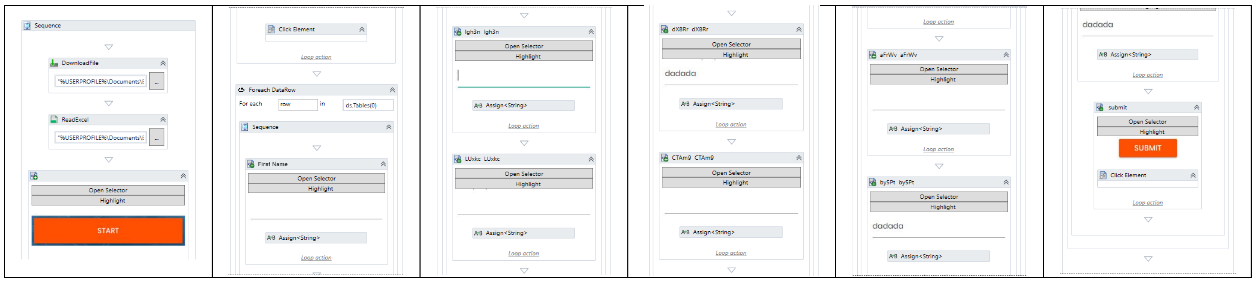
Good luck!

After successfully mapping the **Start** button, your workflow should look like the image below.



Workflow Finished

The finished workflow should look like the sequence below.



Finished RPA Challenge workflow.

3.11.2 Windows Automation

In this example, **Skype** will be used to exemplify how easy it is to automate Windows applications. Our goal is to ask the user for a contact info and then the robot will proceed to Skype and send a `.PDF file` to the specified recipient. Hence, the **OpenRPA** major features being exposed here are: Detector, Form, Selector and the Recorder.

This example is divided in 3 steps: The first steps refers to `Detector`, which will be used to check if a new file is created in an specific folder. The second step refers to `Form`, used to prompt the User about the recipient's contact info (`Email`, `Name` and `Username`) The final step is `Windows Automation (Skype app)/Using Selector`, where the user will record the procedure of sending a message and a file to the their Skype's contact.

If your main interest lies on **Windows UI automation**, you may skip directly to the Windows Automation step.

1. **Detector**
    - Create Detector in OpenRPA
    - Configure Detector node in Node-RED
    - Message Manipulation in Node-RED
    - Passing a variable from Node-RED to OpenRPA
2. **Form.**
    - OpenRPA Forms
3. **Windows Automation (Skype app)/Using Selector**
    - Record task
    - Configure Selectors

Detector

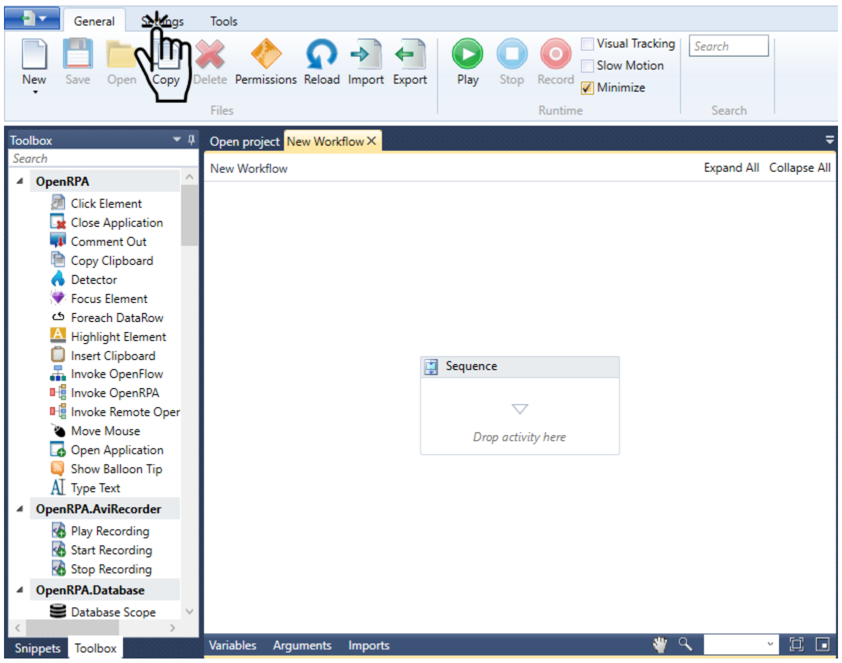
Detectors are synonyms to event-listeners. Alas, they wait for some specific condition to happen and then fire a message to `Node-RED` or even to `OpenRPA` itself.

In this section, the user will learn how to create a `FileWatcher Detector`, which will listen to file changes in a specific directory and will fire an event when a new file is created. For this example, the robot should be firing each time a new `PDF/.pdf file` is created.

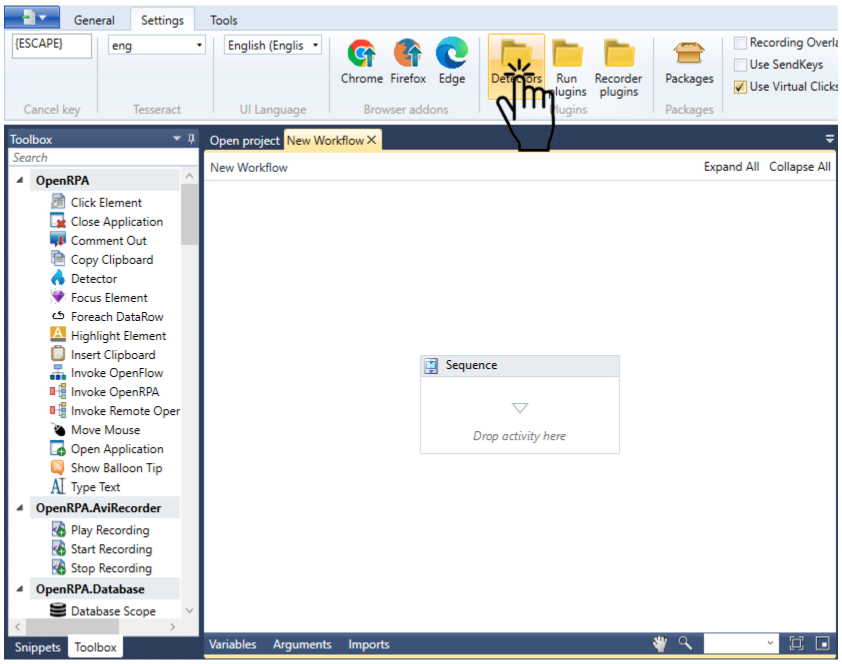
Create Detector in OpenRPA

First, it is needed to configure the `FileWatcher Detector`.

To do that, the user must first click the `Settings` bar inside OpenRPA's ribbon.

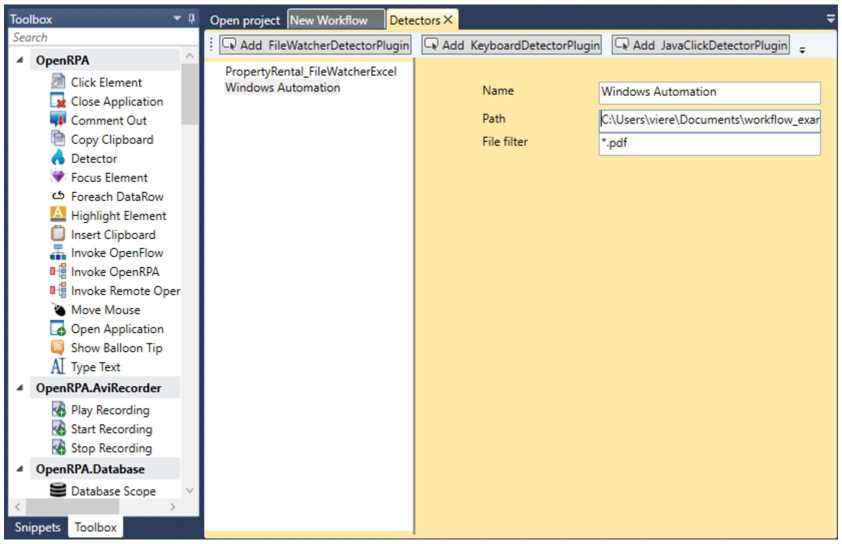


Then, click the Detectors button inside the ribbon.



After that, all the user needs is to click the desired detector inside the Detectors ribbon.

For this example, the `Detector` which will be used is the `FileWatcherDetectorPlugin`. Set the Detector's `Name` - in our case `Windows Automation` as well as the `Path` which it will listen to - `C:\Users\viere\Documents\workflow_example_files\windows_automation\input_files`, **no quotes.** Then set the `File Filter` to `*.pdf` - the `*` is a wildcard for the filename, that is, any filename will do.



Name: Windows Automation
Path: C:\Users\viere\Documents\workflow_exar
File filter: *.pdf

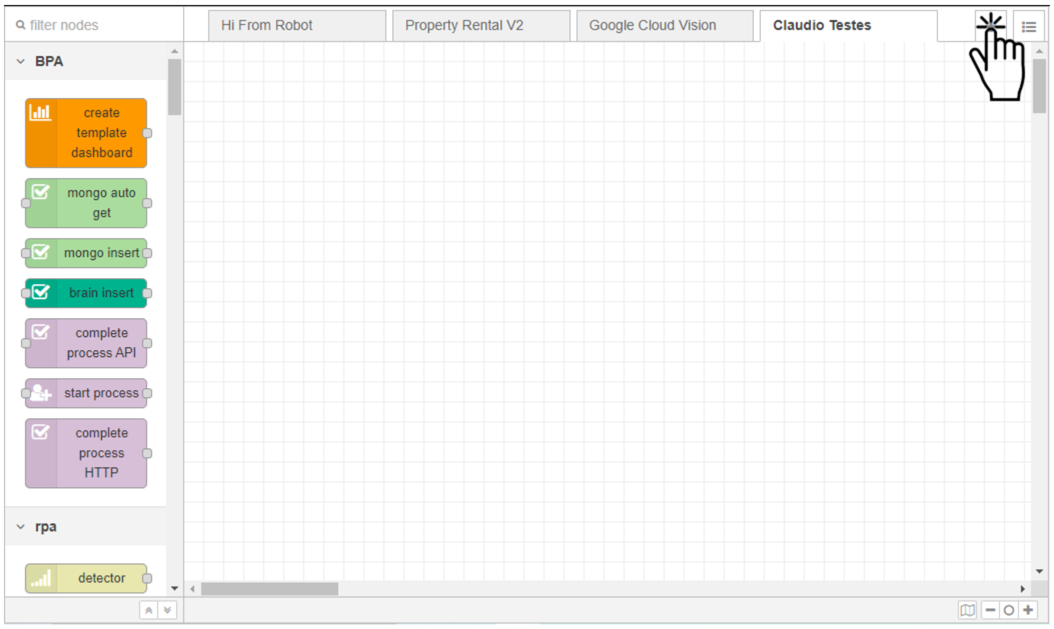
Note: If you can not find a `Detector`, please make sure it has been installed. In case of doubts, simply run the installer `.msi` again and make sure to include it during the installation.

Note: Wildcards (`https://ryanstutorials.net/linuxtutorial/wildcards.php`) are a set of building blocks that allow you to create a pattern defining a set of files or directories. As you would remember, whenever we refer to a file or directory on the command line we are actually referring to a string path. Whenever we refer to a path we may also use wildcards in that path to turn it into a set of files or directories..

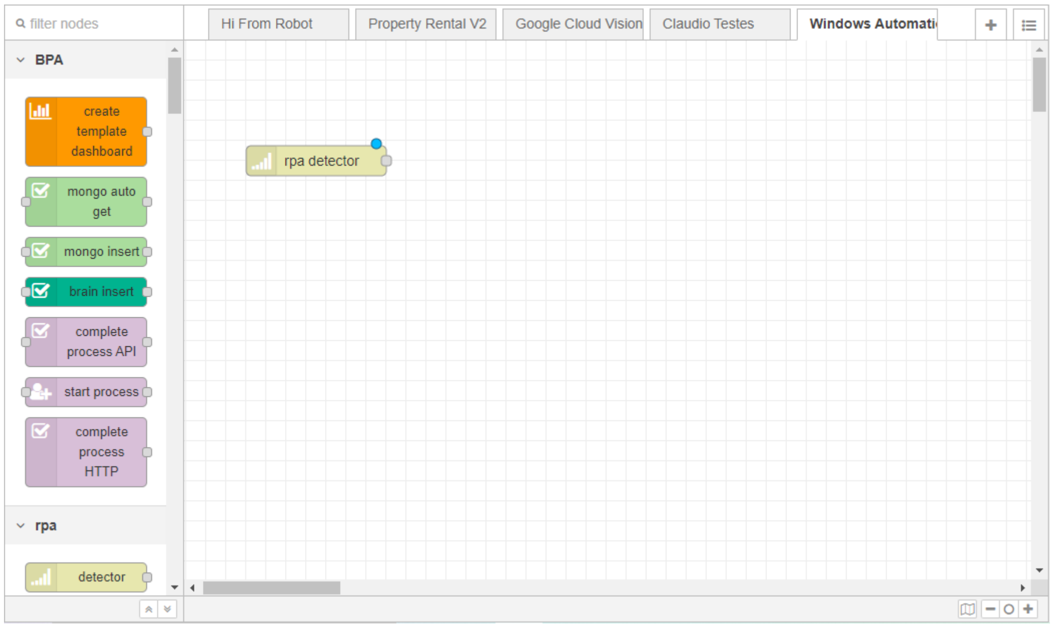
Configure Detector node in Node-RED

Now the user will configure, in Node-RED, a `Detector` node which will be the starting point for `NodeRED Messages` when the `Detector` is triggered.

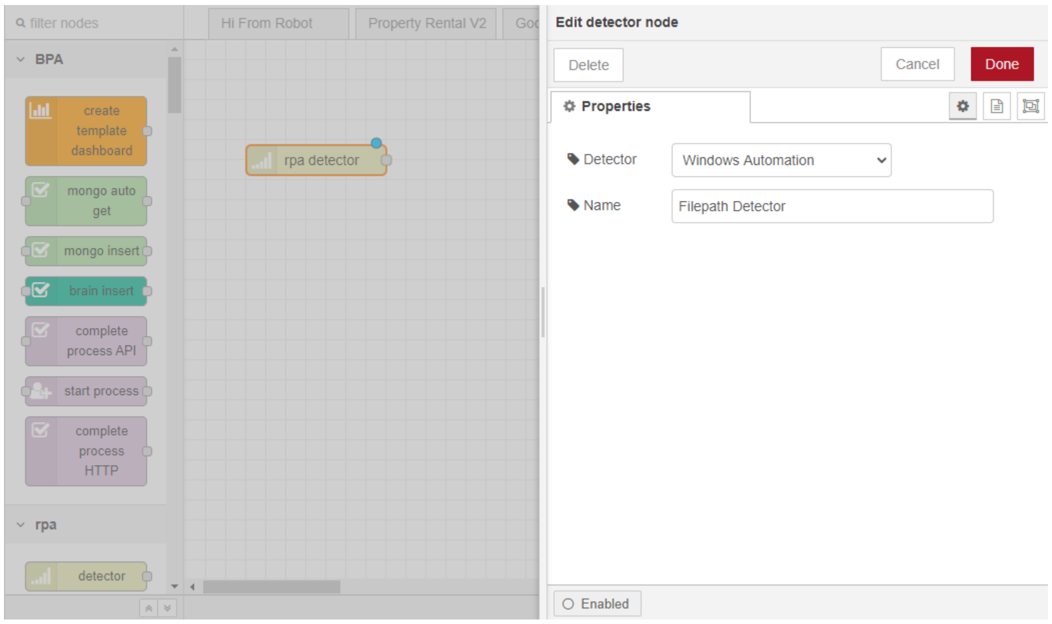
Create a new flow inside Node-RED. Our flow is being named `Windows Automation - Workflow Example`.



Now drag a `detector` node to the workspace.



Double-click the `detector` node to set its properties. Select the `Detector` in the dropdown field which will appear - in our case, `Windows Automation`. Additionally, you may also give a name to this node such as `Filepath Detector`, we recommend doing so to keep flows organized.

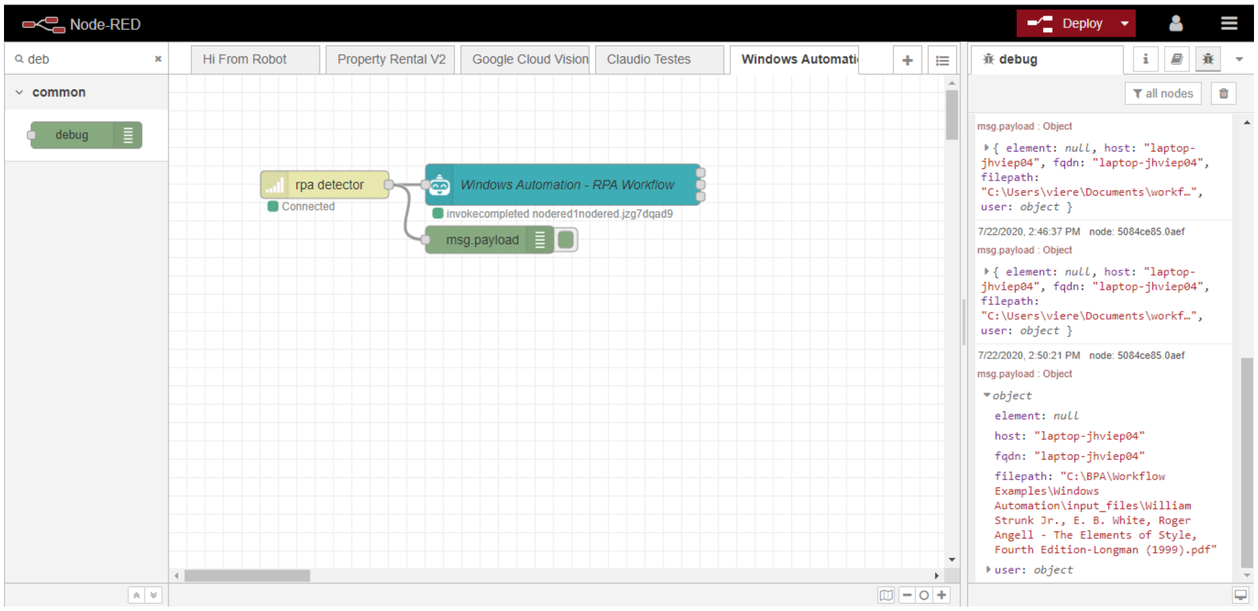

Edit detector node

Detector: Windows Automation

Name: Filepath Detector

Message Manipulation in Node-RED

When the `FileWatcher Detector` fires, some information will be passed in the `NodeRED Message`, such as the `filepath` - that is, the name of the file that has been created.



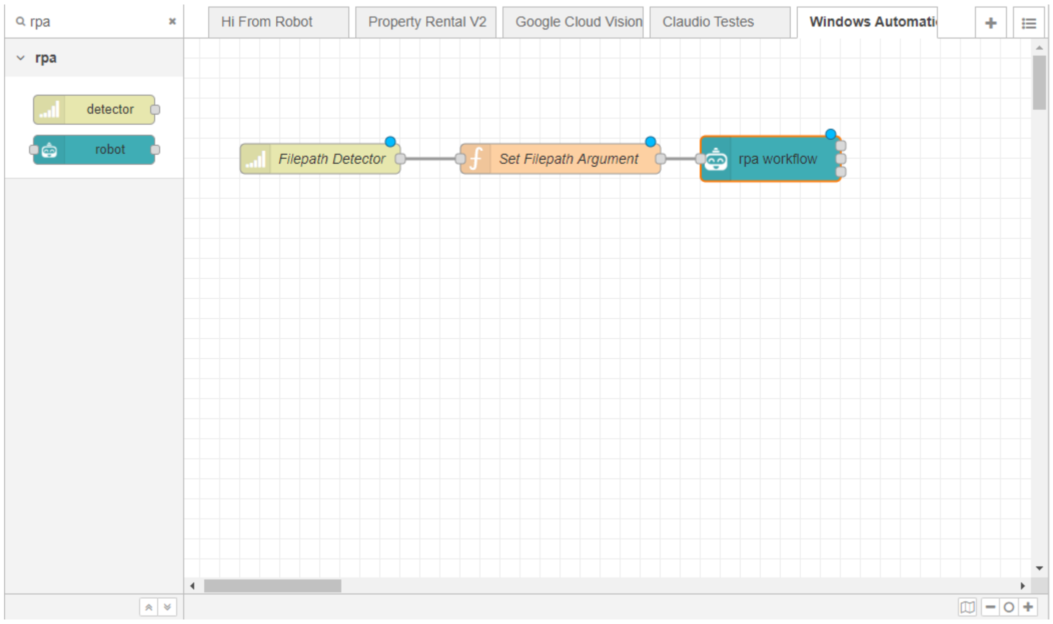
```
msg.payload : Object
▸{ element: null, host: "laptop-jhviep04", fqdn: "laptop-jhviep04", filepath: "C:\Users\viere\Documents\workf…", user: object }

7/22/2020, 2:46:37 PM   node: 5084ce85.0aef
msg.payload : Object
▸{ element: null, host: "laptop-jhviep04", fqdn: "laptop-jhviep04", filepath: "C:\Users\viere\Documents\workf…", user: object }

7/22/2020, 2:50:21 PM   node: 5084ce85.0aef
msg.payload : Object
▾object
  element: null
  host: "laptop-jhviep04"
  fqdn: "laptop-jhviep04"
  filepath: "C:\BPA\Workflow Examples\Windows Automation\input_files\William Strunk Jr., E. B. White, Roger Angell - The Elements of Style, Fourth Edition-Longman (1999).pdf"
 ▸user: object
```

It is desired to capture this data and then pass it back to the robot as `arguments` for when workflows are invoked. Remember that `arguments` should be passed to the **Robot** through `msg.payload` object. As our OpenRPA Workflow does not require any other `argument`, we can plug in the `FileWatcher Detector` node directly to the `robot` node.

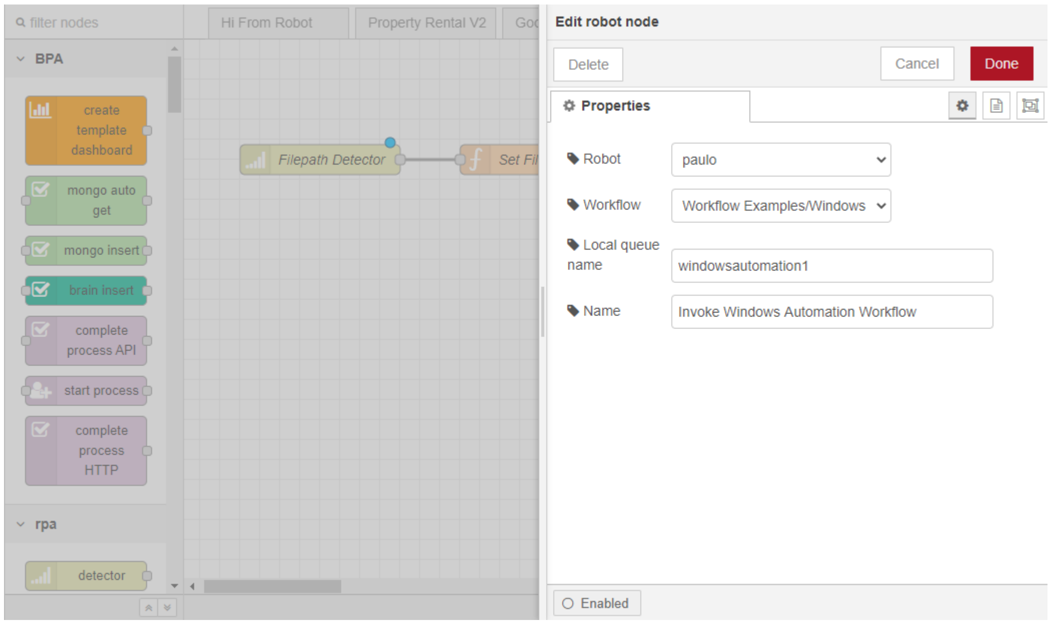
Invoking OpenRPA Workflows

Here lies the logic behind passing the variables to and executing the OpenRPA workflow which will be invoked upon the `Detector`'s positive signaling.

Drag a `robot` node to the workspace. This node enables us to invoke the `Windows Automation` workflow.

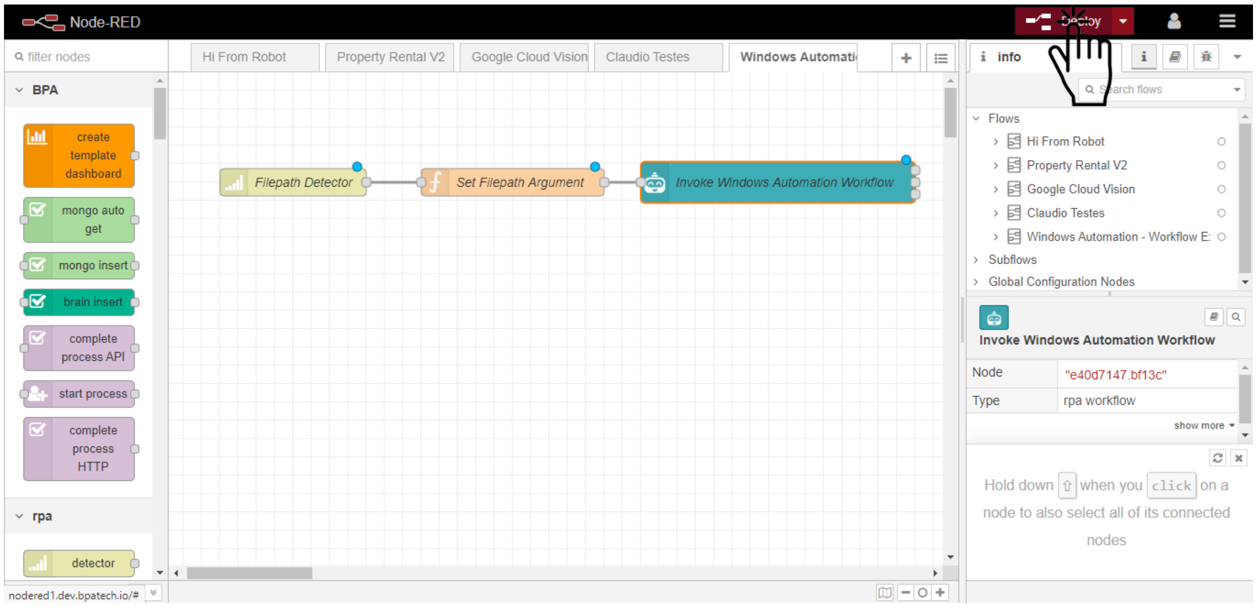


Once again, it is recommended to properly name the nodes, `Invoke Windows Automation Workflow`. Double-click and assign the `Robot (or agent)` whom shall will execute the workflow, and obviously, the name of the `Workflow` that should be executed (this example, `Windows Automation`). You may leave the field `Local Queue` blank.

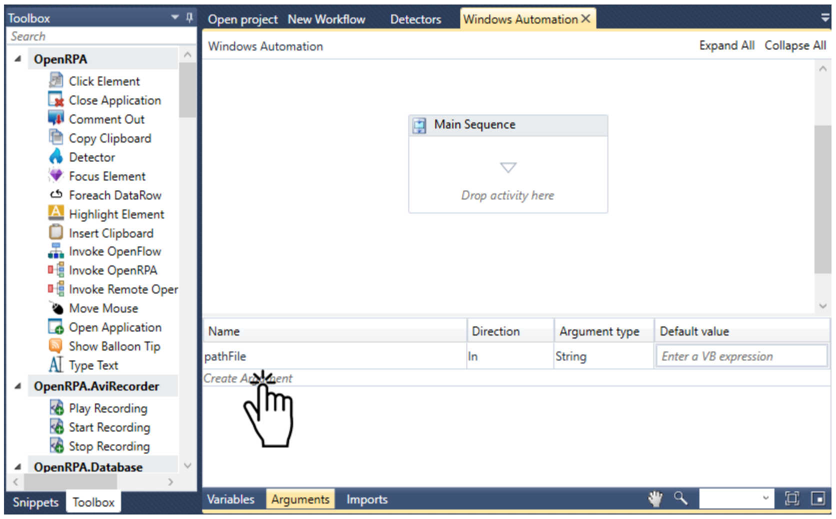


Robot: paulo
Workflow: Workflow Examples/Windows
Local queue name: windowsautomation1
Name: Invoke Windows Automation Workflow

Now, the user must deploy the flow created by clicking the red Deploy button at the top-left side of the screen.



After that, the user should define, inside the OpenRPA workflow, a new argument by clicking `Create Argument`. As you may have seen in the `debug` node, the parameter being passed is called `filepath` and therefore the `argument` should be the exact same. As the `argument` is an input, set its `Direction` to `In`.



Name	Direction	Argument type	Default value
pathFile	In	String	Enter a VB expression

Note: If the user wants to avoid using Node-RED completely, an infinite **While loop** activity (condition: true=true) with a **Try Block** is suggested along with the **Detector** activity containing the detector defined previously. The user may also use the **Assign** activity to assign a fixed `filepath` for the file which will be sent to the **Skype** contact. By using the loop, every time an iteration finishes, the robot is always back to the start waiting on the `Detector` to fire once more and in case anything extraordinary happens, the **Try Block** assures that the **Workflow** will keep running.

Form

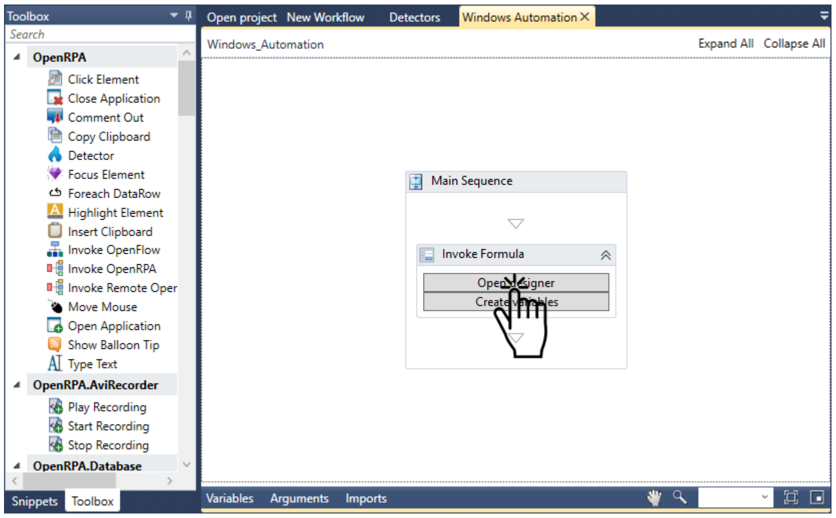
In this section, the user will learn how to create a `Form` to capture the `Email`, `Skype contact` and `Contact name` which will be used to pass the attachment and template message using the **Skype** application.

OpenRPA Forms

Here the user creates a dynamic WPF Form to gather the values mentioned above from the client.

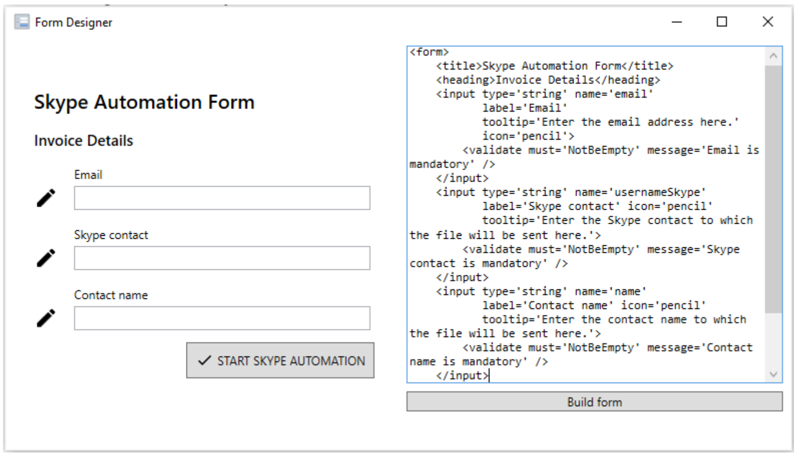
First drag a `Invoke Formula` snippet to the main sequence.

Now click `Open designer` to open the `Form Designer`.



Clear all example code and add the code snippet below to the designer code editor.

```
<form>
   <title>Skype Automation Form</title>
   <heading>Invoice Details</heading>
   <input type='string' name='email'
         label='Email'
         tooltip='Enter the email address here.'
         icon='pencil'>
      <validate must='NotBeEmpty' message='Email is mandatory' />
   </input>
   <input type='string' name='usernameSkype'
         label='Skype contact' icon='pencil'
         tooltip='Enter the Skype contact to which the file will be sent here.'>
      <validate must='NotBeEmpty' message='Skype contact is mandatory' />
   </input>
   <input type='string' name='name'
         label='Contact name' icon='pencil'
         tooltip='Enter the contact name to which the file will be sent here.'>
      <validate must='NotBeEmpty' message='Contact name is mandatory' />
   </input>
   <action name='submit' content='START SKYPE AUTOMATION' icon='check' validates='true
↪' ClosesDialog='true' />
</form>
```

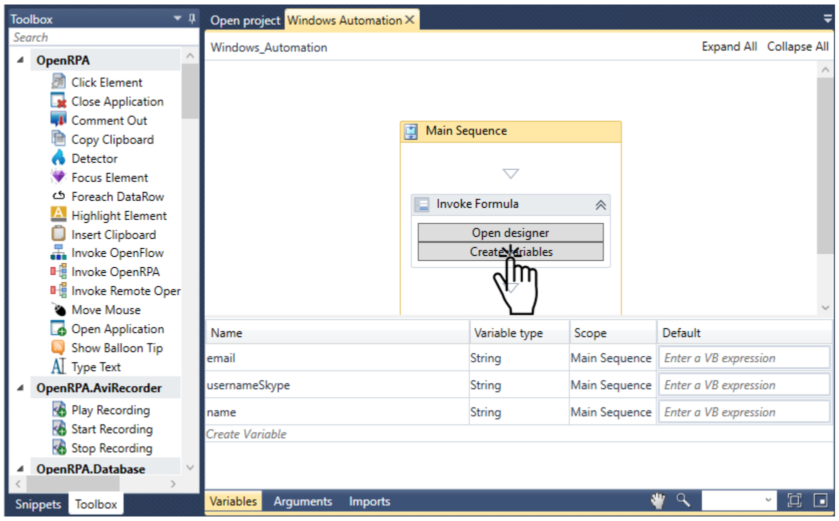


```
<form>
    <title>Skype Automation Form</title>
    <heading>Invoice Details</heading>
    <input type='string' name='email'
           label='Email'
           tooltip='Enter the email address here.'
           icon='pencil'>
        <validate must='NotBeEmpty' message='Email is
mandatory' />
    </input>
    <input type='string' name='usernameSkype'
           label='Skype contact' icon='pencil'
           tooltip='Enter the Skype contact to which
the file will be sent here.'>
        <validate must='NotBeEmpty' message='Skype
contact is mandatory' />
    </input>
    <input type='string' name='name'
           label='Contact name' icon='pencil'
           tooltip='Enter the contact name to which
the file will be sent here.'>
        <validate must='NotBeEmpty' message='Contact
name is mandatory' />
    </input>
```

If you're used to HTML or any other markup language you might understand what's happening behind the scenes here. If you don't, do not worry and refer to the **OpenRPA Forms** section.

By pressing the **Build form** button the user is able to build and see a preview of the WPF form in the left-side panel of the `Form Designer`.

After that, the user can press the **Create variables** button so all variables defined inside the `Form Designer` will be automatically created. Easy-peasy, isn't it?



Name	Variable type	Scope	Default
email	String	Main Sequence	Enter a VB expression
usernameSkype	String	Main Sequence	Enter a VB expression
name	String	Main Sequence	Enter a VB expression
Create Variable			

Note: If the user wants to avoid using `OpenRPA Forms`, he can simply use the **Assign** activity to assign values to the variables `email`, `usernameSkype` and `name`, which will be used further to send the attachment and template message to the desired client.

Note: The `Form Designer` is the place where the user can create his desired WPF forms to gather input from the client. At the right-side of it, a preview of the input form is shown. At the left-side of it, the user enters the code used to design the input form. After adding code to the `Form Designer`, always press the **Build form** button to save changes and visualize the preview.

Windows Automation (Skype app) / Using Selectors

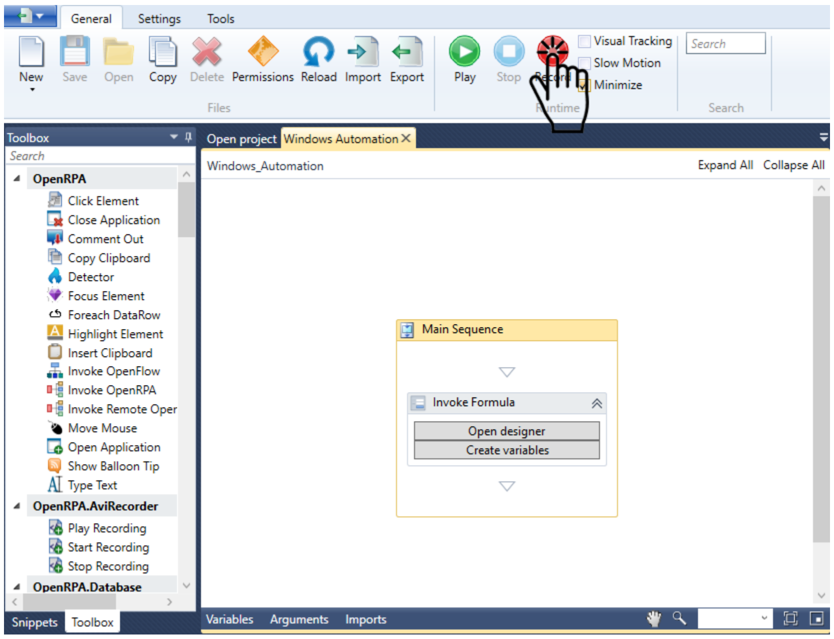
Now comes the greatest part of our journey so far in this workflow example. The built-in Recorder will be used to record the task that an user would normally do to effectively achieve the task described in this section's introduction.

Record task

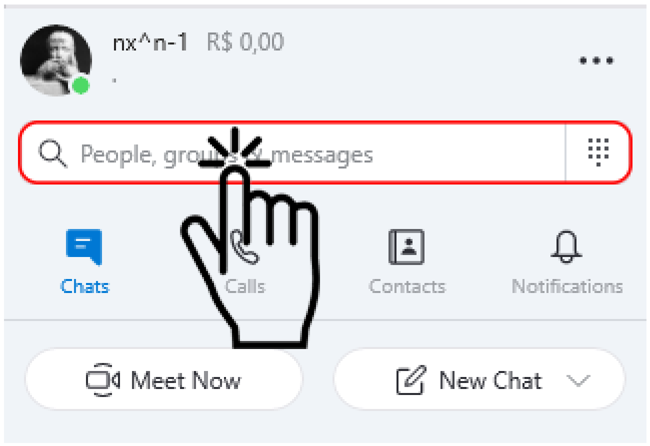
Here the user will record the task usually done, namely search for a contact inside the **Skype** application, select - or click - that contact and send a message containing the attachment.

The first step is to open the **Skype** application.

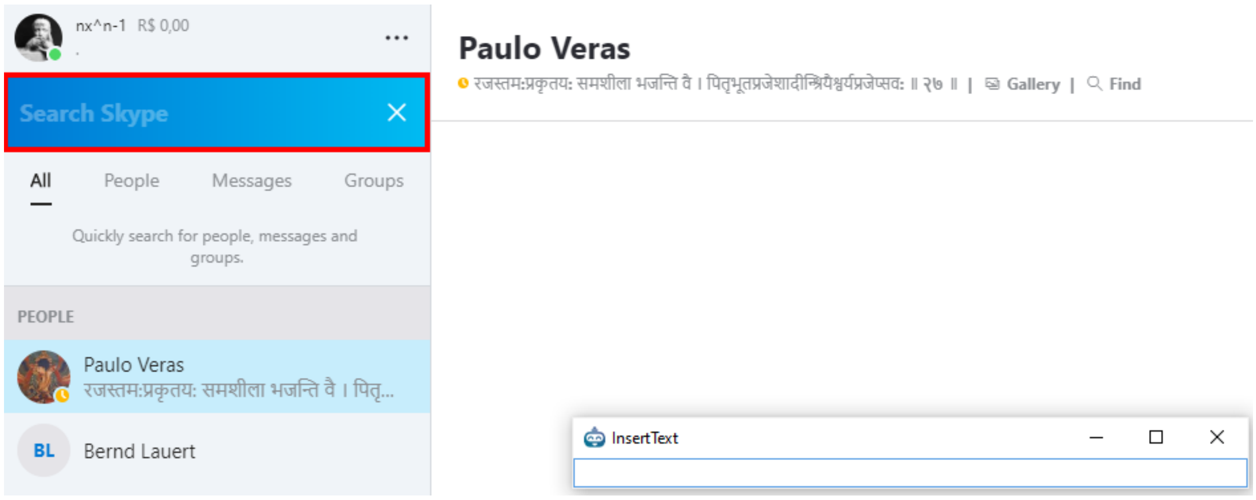
Now, with the **Skype** application open, the user will start the Recorder inside OpenRPA by clicking in the button inside OpenRPA's ribbon and execute the task as if done manually.

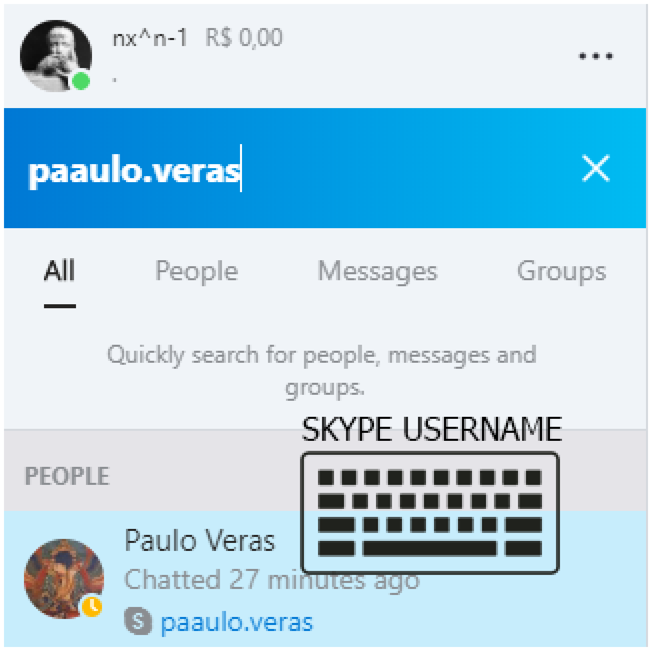


Now, with the Recorder started, the user must click the `Search` tab, containing the placeholder `People, groups & messages`.



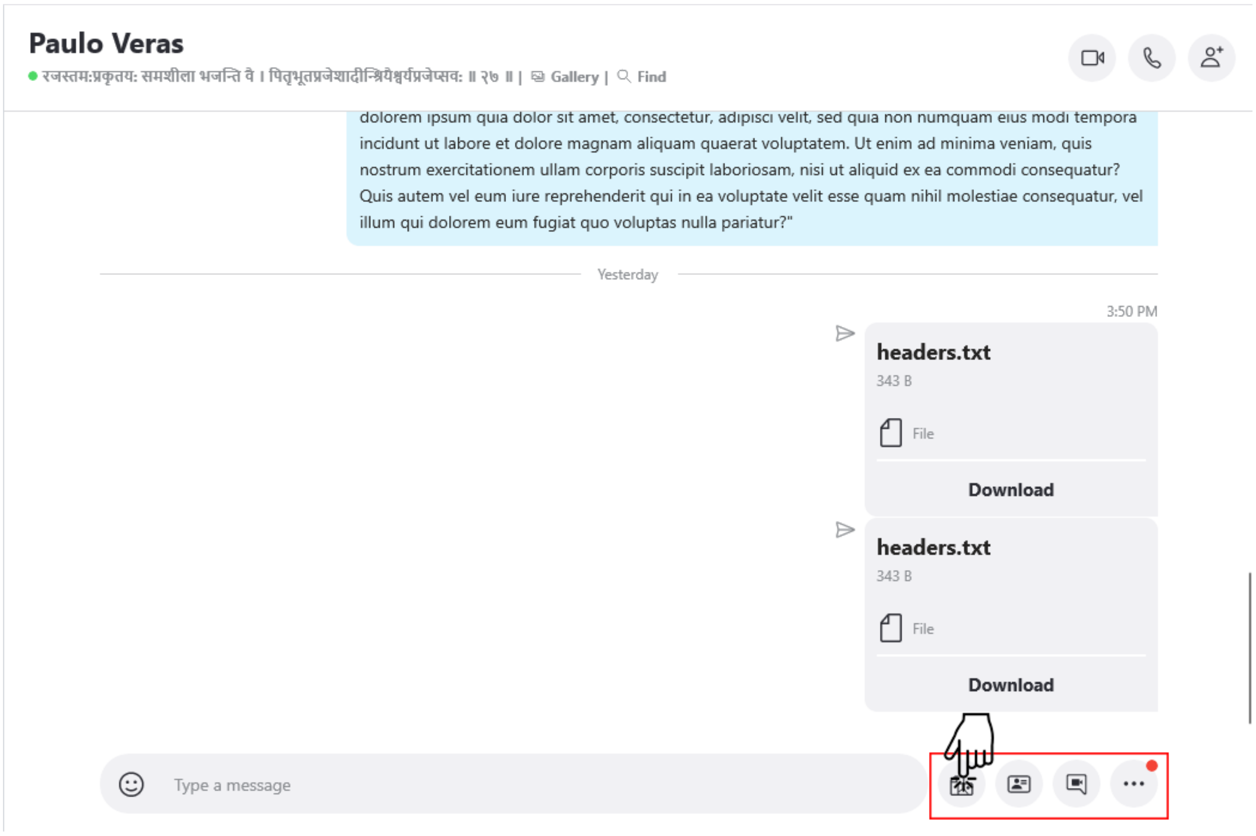
An input field will appear inside the `Search` tab containing the placeholder `Search Skype`. Click that input field, an `InsertText` input field will appear. The user should enter an existing contact which will be used as a dummy to assign the proper `Selector` further in this section. After waiting for **Skype**'s own autocomplete feature to find the user, the user shall click the desired contact.



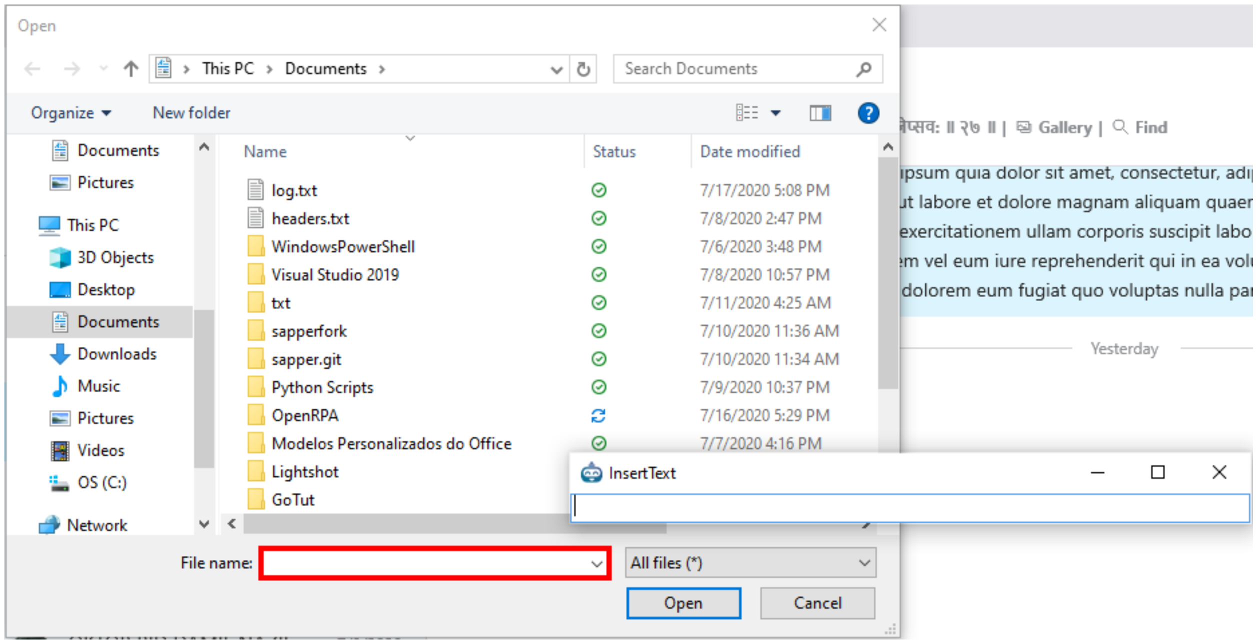


After clicking the desired contact, the `Messages` tab open will open for that specific contact.

The user must now click the `Add File` button in the lower right section.

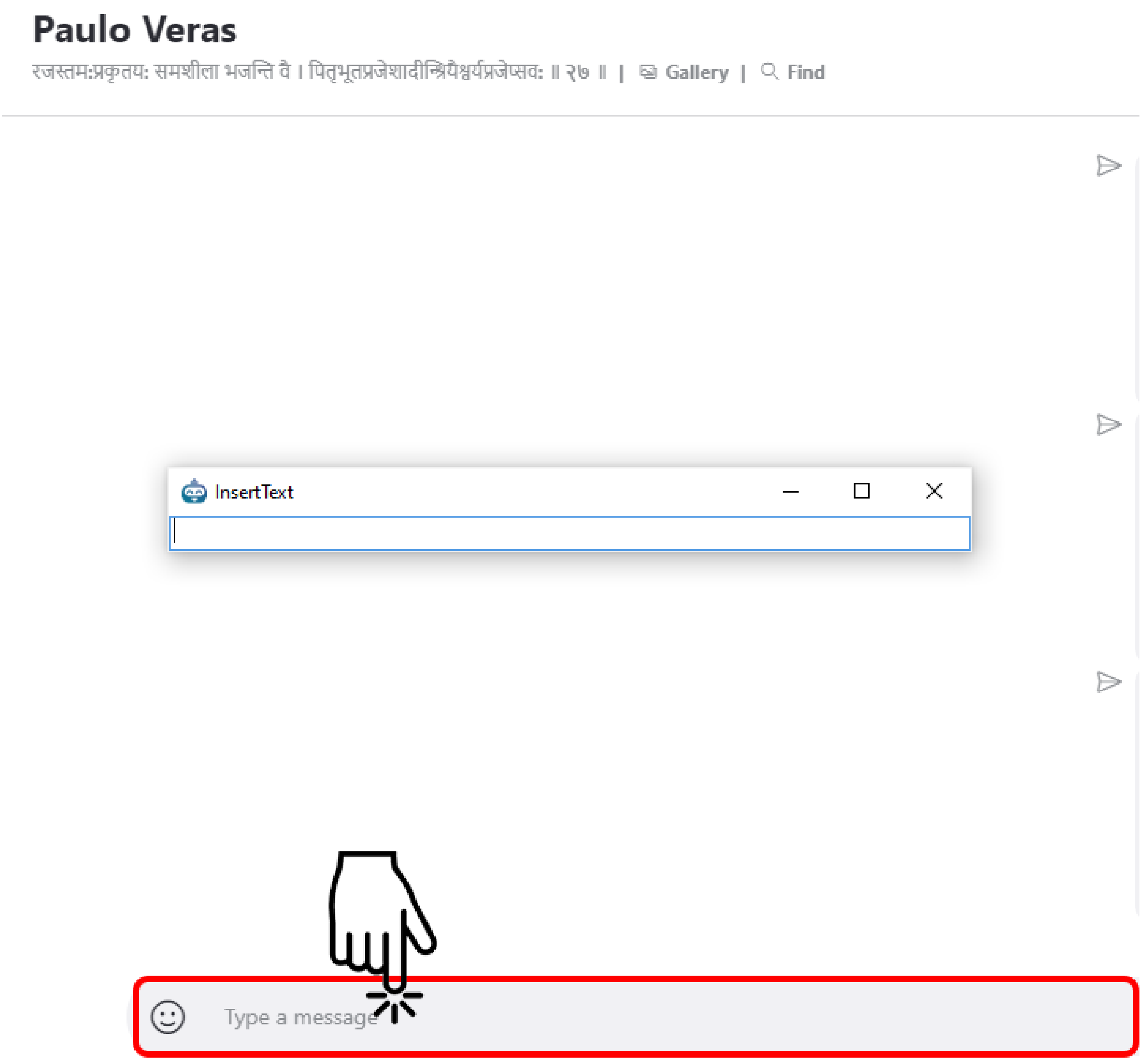

Paulo Veras
रजस्तम:प्रकृतय: समशीला भजन्ति वै । पितृभूतप्रजेशादीन्श्रियैश्वर्यप्रजेप्सव: ॥ २७ ॥ | Gallery | Find
dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?"
Yesterday
3:50 PM
headers.txt
343 B
File
Download
headers.txt
343 B
File
Download
Type a message

Now a file dialog appears, where the user can select the desired file, and an input dialog from OpenRPA titled `InsertText` where you can again insert dummy text. The `Selector` will act upon this dialog to choose the file which will be sent, as is seen in the next section. Then click the `Open` button.

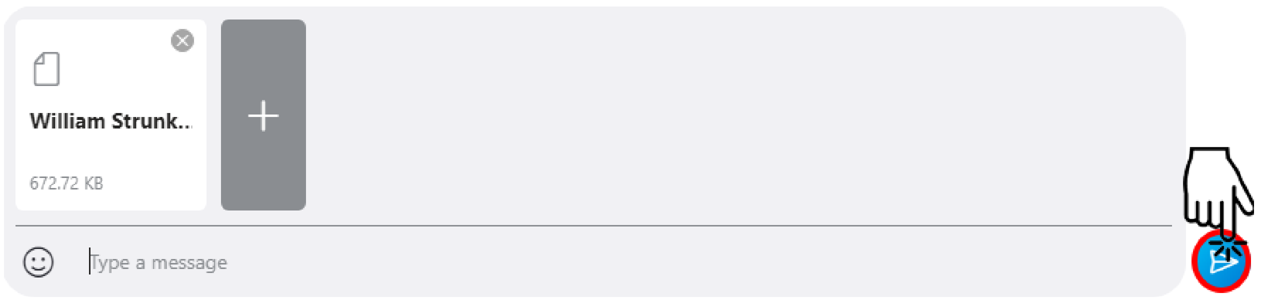


Name	Status	Date modified
log.txt	✓	7/17/2020 5:08 PM
headers.txt	✓	7/8/2020 2:47 PM
WindowsPowerShell	✓	7/6/2020 3:48 PM
Visual Studio 2019	✓	7/8/2020 10:57 PM
txt	✓	7/11/2020 4:25 AM
sapperfork	✓	7/10/2020 11:36 AM
sapper.git	✓	7/10/2020 11:34 AM
Python Scripts	✓	7/9/2020 10:37 PM
OpenRPA	⟳	7/16/2020 5:29 PM
Modelos Personalizados do Office	✓	7/7/2020 4:16 PM
Lightshot		
GoTut		

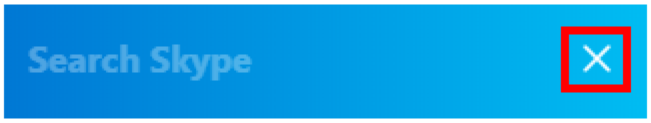
Now click the input field with the `Type a message` placeholder, another input dialog with the `InsertText` title will appear, you can again enter any dummy text.


Paulo Veras
रजस्तमःप्रकृतयः समशीला भजन्ति वै । पितृभूतप्रजेशादीन्श्रियैश्वर्यप्रजेप्सवः ॥ २७ ॥ | Gallery | Find
InsertText
Type a message

Then, the user shall click the blue **Send** button.



Finally, click the **X** button inside the `Search` tab to close the search dialogue and exit the Recorder by pressing the `ESC` button.

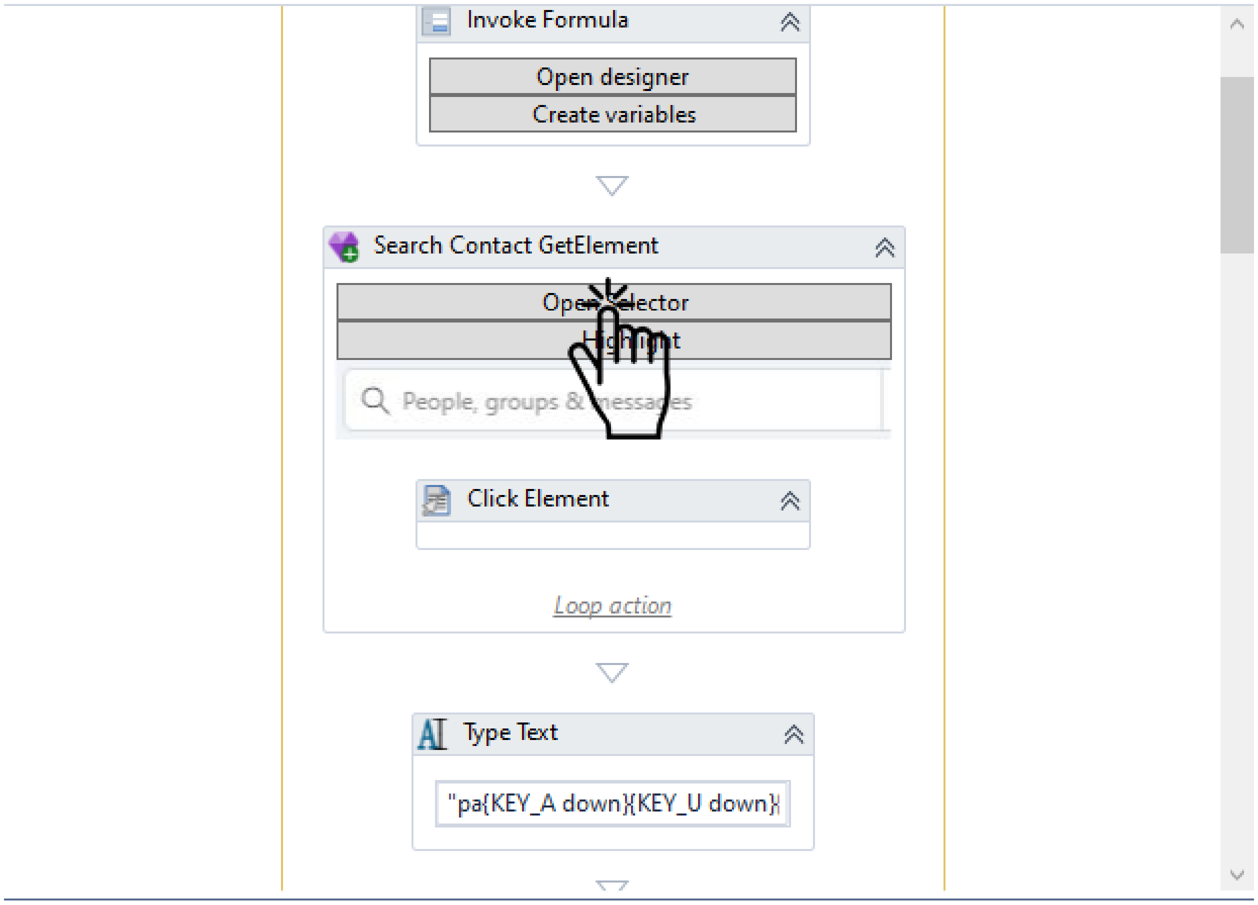


Configure Selectors

Now the `Selector` for the many **GetElement** activities that appeared inside OpenRPA's designer is configured.

First, configure the first `Selector` for the first element to be clicked inside the **Skype** application pane. That is, the one with the `People, groups & messages` placeholder. In our case, we changed its `DisplayName` from `Search for people, groups & messages` to `Search Contact GetElement`, to ease the viewing process by a 3rd-party which may be interested in the workflow.

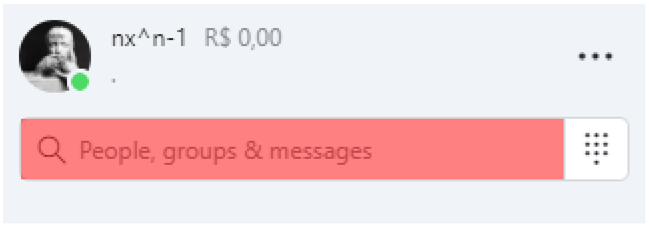
Click the **Open Selector** button inside the **GetElement** activity that was generated for that specific field.



In this image, the user notices a **Type Text** activity, which corresponds to the dummy username that was typed. The user can remove that activity from the workspace, since it'll be replaced with another activity in the following steps.

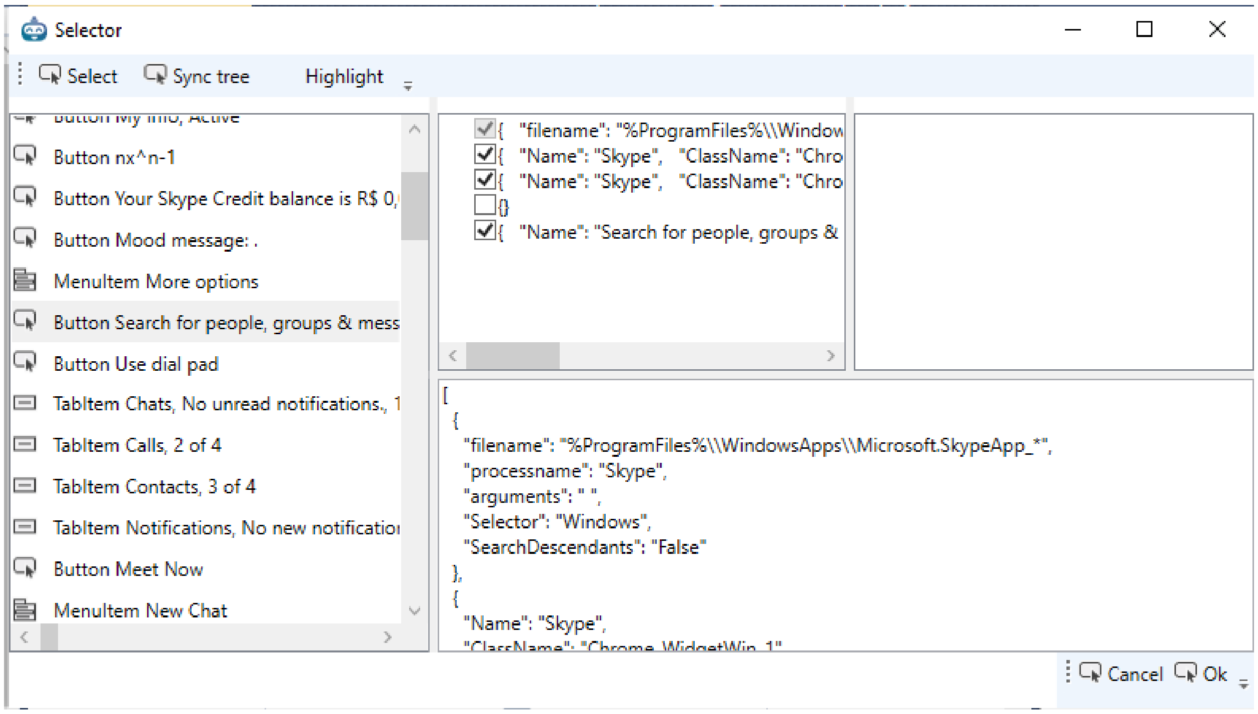
The user will notice that upon clicking the **Highlight** button inside the new window and with the **Skype** application window open, no elements inside the screen will be highlighted. Then, we must select the given element manually.

To do that, the user can first double-click the element `Pane Skype` -> `Document Skype XXXXXXXXXX` -> `Button Search for people, groups & messages` and right-click this last element and click the **Select Element** button that appears. Below follows a picture of this element highlighted.



Now, the user is going to change the `filename` attribute in the first selector, where there may be `"%ProgramFiles%\\WindowsApps\\Microsoft.SkypeApp_15.61.100.0_x86__XXXXX\\Skype\\Skype.exe"`, the user wants to crop the version of skype and the hash which follows it - specified here as `{XXXXX}`. To do that, simply change it to ``"%ProgramFiles%\WindowsApps\Microsoft.SkypeApp_*"``[1].

Now the user must also deselect the fourth selector. All the changes made so far are shown in the image below.

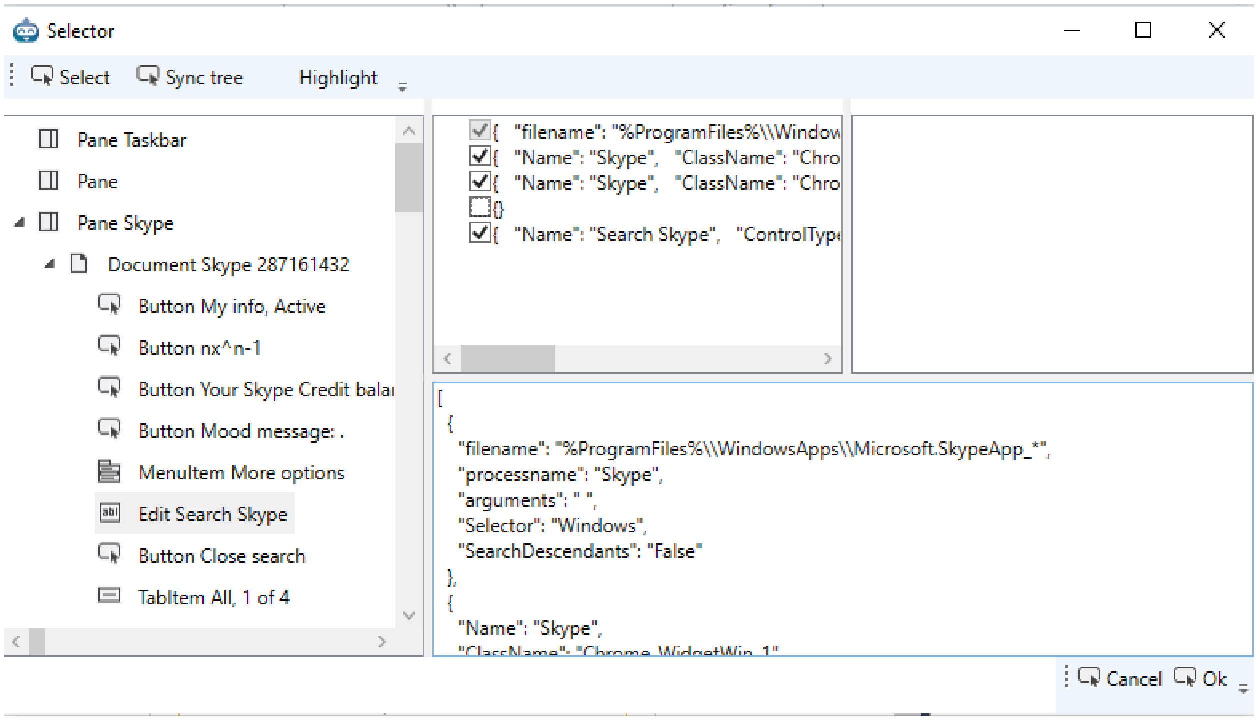


```
✓{ "filename": "%ProgramFiles%\\Window
✓{ "Name": "Skype", "ClassName": "Chro
✓{ "Name": "Skype", "ClassName": "Chro
☐{}
✓{ "Name": "Search for people, groups &
```

```
[
 {
  "filename": "%ProgramFiles%\\WindowsApps\\Microsoft.SkypeApp_*",
  "processname": "Skype",
  "arguments": " ",
  "Selector": "Windows",
  "SearchDescendants": "False"
 },
 {
  "Name": "Skype",
  "ClassName": "Chrome_WidgetWin_1"
```

After that, simply the **Ok** button in the lower-left side of the screen. It is recommended that the user check once more whether the proper `Selector` was configured by clicking the **Highlight** button inside the activity.

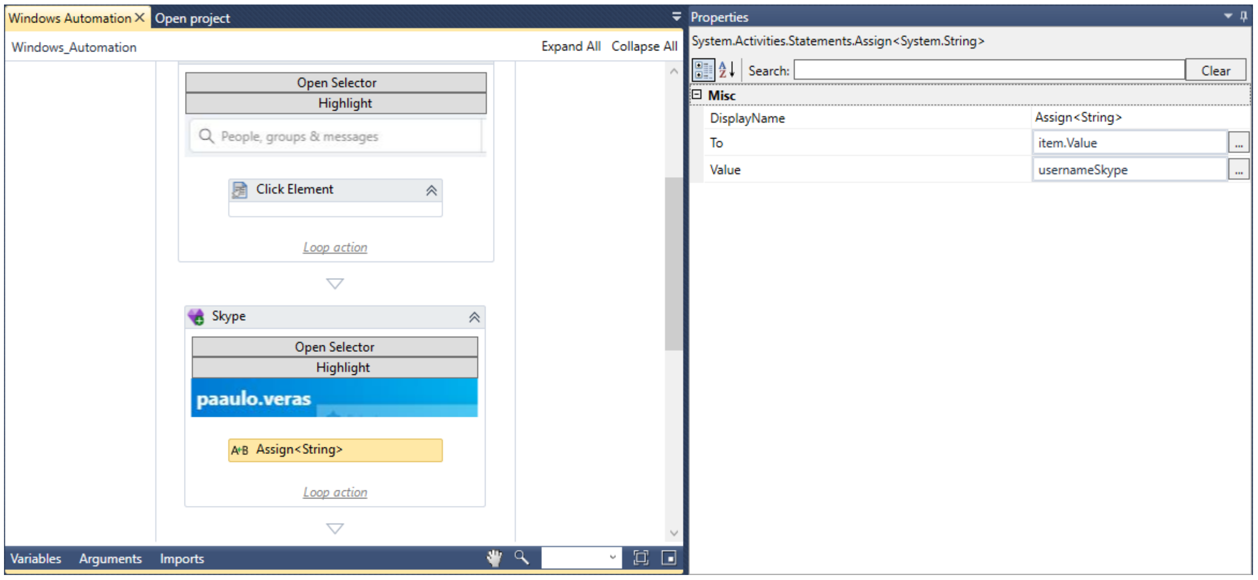
Now the user must do the same steps specified above inside the **GetElement** activity responsible for capturing the `Search` tab input. Select the element inside `Pane Skype` -> `Document Skype XXXXXXXXX` -> `Edit Search Skype`. It is important to remark that the `Search` tab must be visible inside the **Skype** application before opening the selector so OpenRPA can recognize it as an element inside the window.



```
{ "filename": "%ProgramFiles%\\Window
{ "Name": "Skype",  "ClassName": "Chro
{ "Name": "Skype",  "ClassName": "Chro
{}
{ "Name": "Search Skype",  "ControlType
```

```
[
 {
  "filename": "%ProgramFiles%\\WindowsApps\\Microsoft.SkypeApp_*",
  "processname": "Skype",
  "arguments": " ",
  "Selector": "Windows",
  "SearchDescendants": "False"
 },
 {
  "Name": "Skype",
  "ClassName": "Chrome_WidgetWin_1"
```

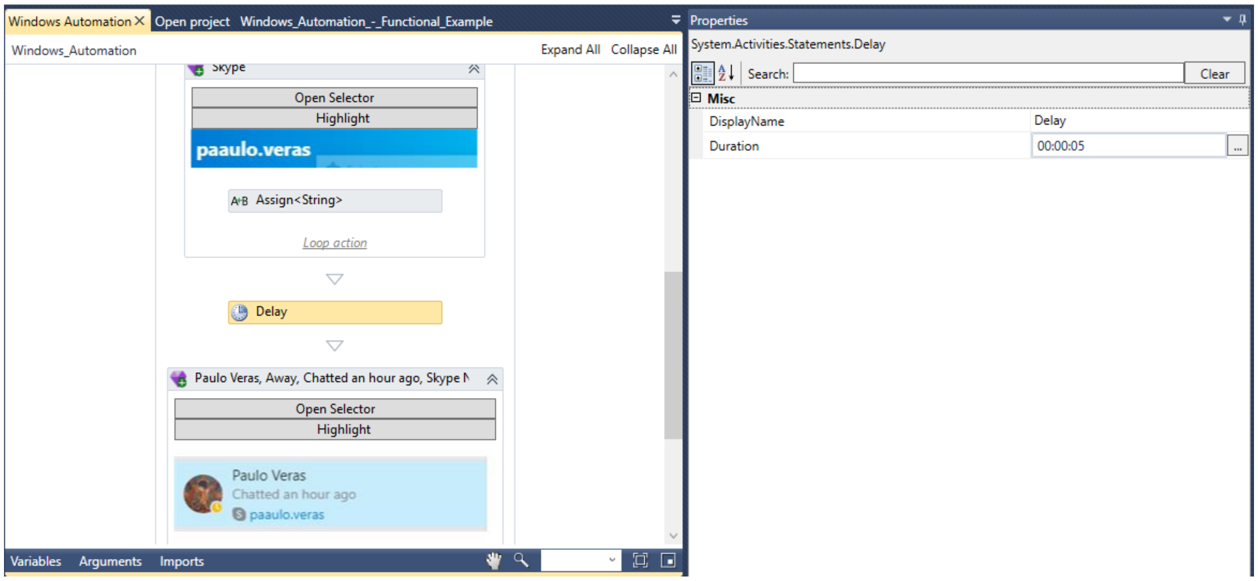
The user must also change the right-hand value of the **Assign** activity, inside the current **GetElement** activity, to the variable which is going to hold the `username` of the desired contact. In our case, that variable is `usernameSkype`, as previously defined in the **Forms** section.



System.Activities.Statements.Assign<System.String>

DisplayName	Assign<String>
To	item.Value
Value	usernameSkype

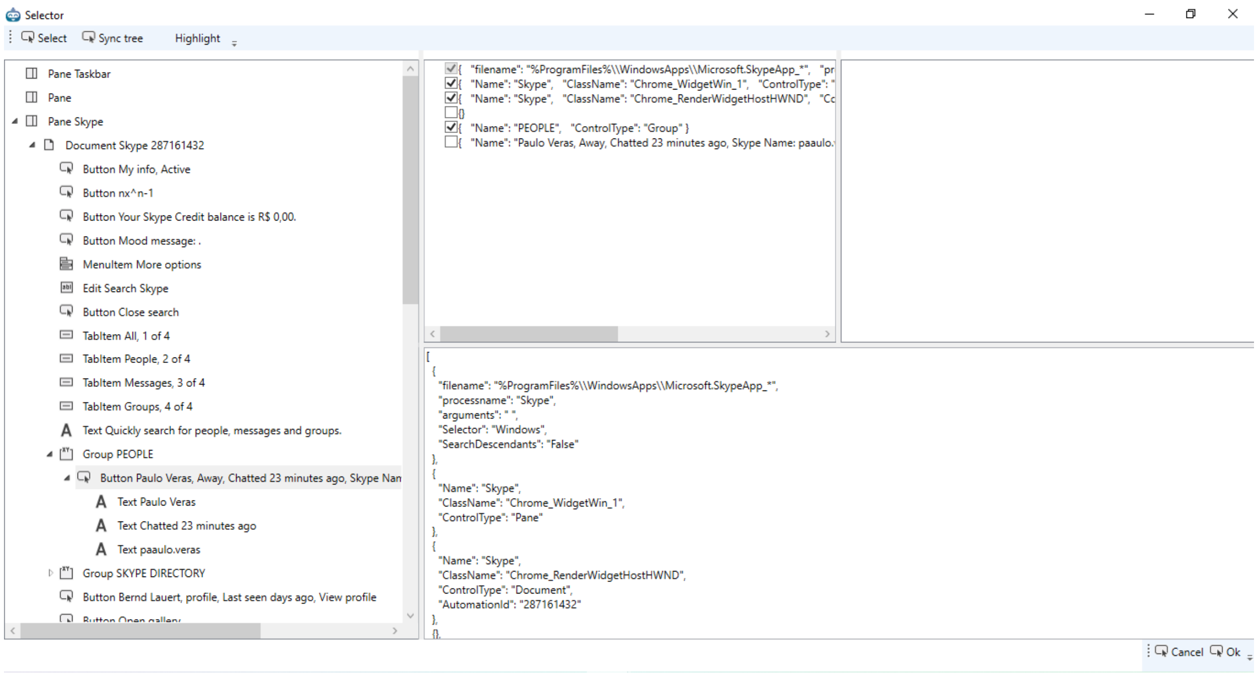
Now the user will add a **Delay** activity to the workflow. Set the **Delay**'s `Duration` property to `00:00:05`, which means this delay will last for 5 seconds. This is needed specifically because of the autocomplete function built-in in the **Skype** application. It is a hard-coded constraint in which the robot must wait for the autocomplete function to complete before proceeding to the next steps.



System.Activities.Statements.Delay

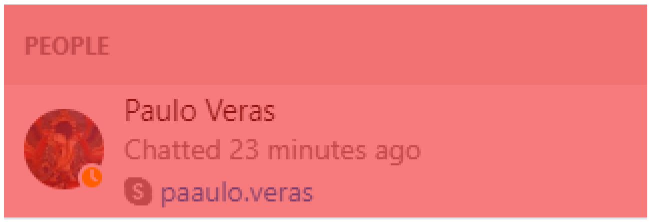
DisplayName	Delay
Duration	00:00:05

Then, the user must repeat the steps specified above inside the **GetElement** activity responsible for capturing the `Contact` button. Select the element inside `Pane Skype` -> `Document Skype XXXXXXXXX` -> `Group PEOPLE` -> `Button {Contact Name}, {Status}, {Chatted Last}, {Skype Name}` - namely, in our case, `Paulo Veras`, `Away`, `Chatted 23 minutes ago`, `Skype Name`. It is important to remark that the `Contact` button must be visible inside the **Skype** application before opening the selector so OpenRPA can recognize it as an element inside the window. Below is the `Selector` properly set up and an image of the button highlighted.



```
✓{   "filename": "%ProgramFiles%\\WindowsApps\\Microsoft.SkypeApp_*",   "pr
✓{   "Name": "Skype",   "ClassName": "Chrome_WidgetWin_1",   "ControlType": "
✓{   "Name": "Skype",   "ClassName": "Chrome_RenderWidgetHostHWND",   "Cc
☐{}
✓{   "Name": "PEOPLE",   "ControlType": "Group" }
☐{   "Name": "Paulo Veras, Away, Chatted 23 minutes ago, Skype Name: paaulo.
```

```
[
 {
  "filename": "%ProgramFiles%\\WindowsApps\\Microsoft.SkypeApp_*",
  "processname": "Skype",
  "arguments": " ",
  "Selector": "Windows",
  "SearchDescendants": "False"
 },
 {
  "Name": "Skype",
  "ClassName": "Chrome_WidgetWin_1",
  "ControlType": "Pane"
 },
 {
  "Name": "Skype",
  "ClassName": "Chrome_RenderWidgetHostHWND",
  "ControlType": "Document",
  "AutomationId": "287161432"
 },
 {},
```

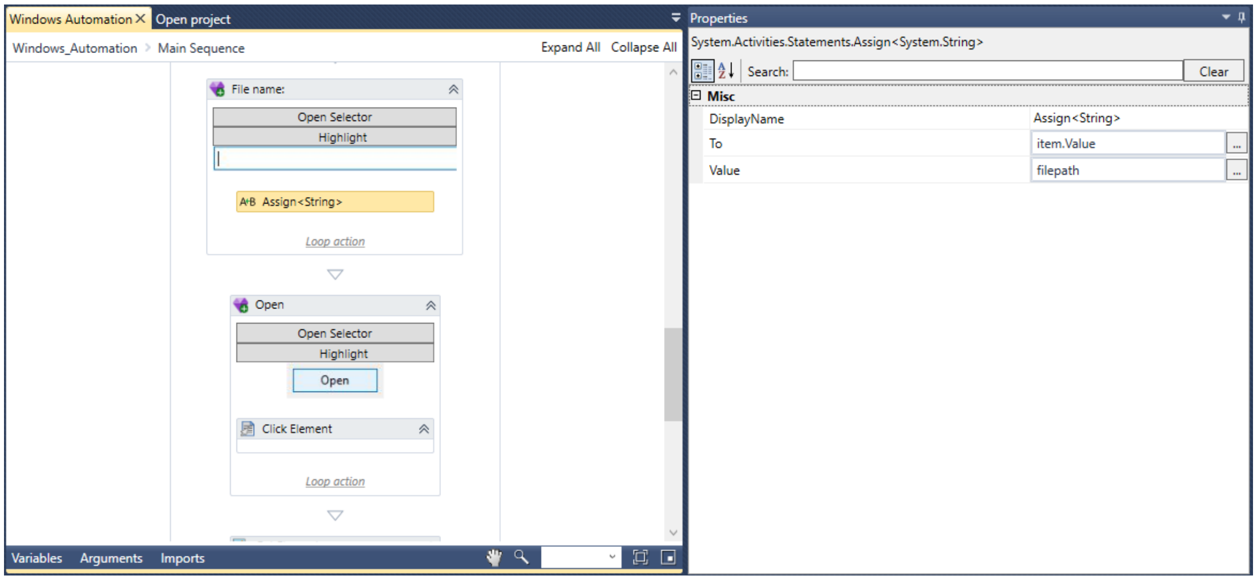

PEOPLE

Paulo Veras
Chatted 23 minutes ago
paaulo.veras

Now add a **post wait** by using the `Post Wait` properties inside the **Click Element** activity which is contained inside the **GetElement** activity responsible for capturing the `Contact` button. This value should be set to `00:00:02`. The **post wait** is used so, after clicking the desired contact button which appears in the left pane of the **Skype** window, the application itself will wait for the `Messages` tab to load. Since we have added the **post wait**, the **Delay** activity is not needed.

The user must now repeat the steps specified for selecting the proper `Selector` for the remaining activities and assure that the elements inside the screen are highlighted when properly configured. Remember, you may have to open a window... in the case of the **Add file** button to properly select the elements.

Now set the proper value for the `filepath` for the file which will be sent through the **Skype** app. Assign the proper value inside the respective **Assign** activity which corresponds to the input field where the `filepath` is going to be set. If you haven't created the `argument` yet, now is the time to do it - remember it must have its `Direction` set as `In`. Below it is shown how to set this argument properly.



System.Activities.Statements.Assign<System.String>

DisplayName	Assign<String>
To	item.Value
Value	filepath

Finally, check whether the configs are correct for the element inside the `Selector` corresponding to `Pane Skype` -> `Document Skype XXXXXXXXX` -> `Edit Type a message` and alter item.value to item.SendKeys inside the **Assign** activity included in the **GetElement** activity.

If the user refrains from doing so, OpenRPA will attempt to directly set the value of the Skype's edit field to the contents of the variable templateMessage. The problem is: Skype does not like that! The Skype app assumes that the user would type a letter at a time instead of changing the edit field value directly, otherwise the UI gets messed up with the placeholder "Type a message " ending on top of the actual text message.

The correct way to mimic an user typing a letter at a time is to use item.SendKeys method.

If the `Detector` is set properly, the user can now test the workflow example by inserting a `.PDF` file inside the directory previously defined. Else, he can follow the note below.

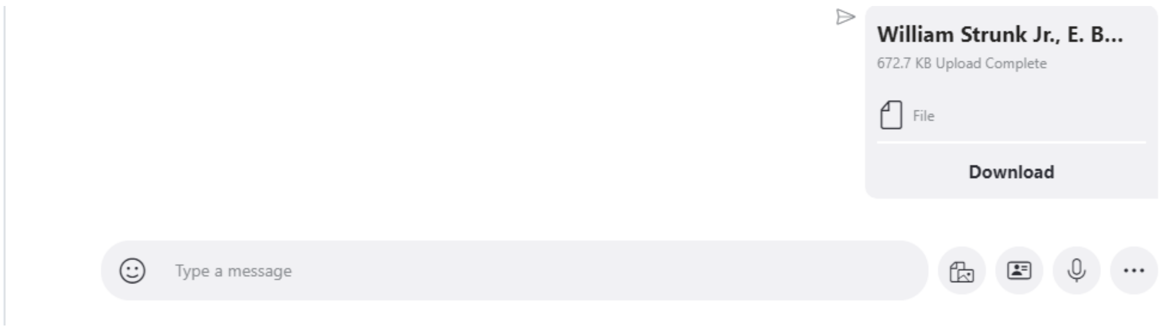
Note: If the user hasn't set the `filepath` argument to be acquired from the `FileWatcherDetectorPlugin`, as seen in the 1st step in this workflow example, he must set its value manually by using an **Assign** activity.

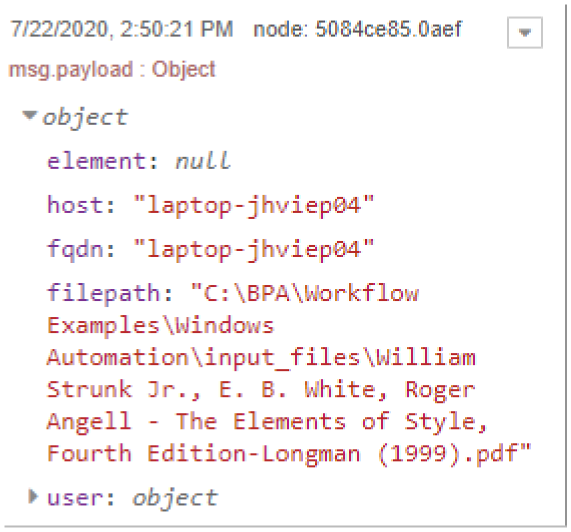
Note: To change the name, or `DisplayName` - as set inside the activity's parameters, the user can either click twice in the name of the activity, type the desired name and press `RETURN` or set the parameter `DisplayName` inside the **Properties box**.

Note: [1] - Wildcards (`https://ryanstutorials.net/linuxtutorial/wildcards.php`) are a set of building blocks that allow you to create a pattern defining a set of files or directories. As you would remember, whenever we refer to a file or directory on the command line we are actually referring to a path. Whenever we refer to a path we may also use wildcards in that path to turn it into a set of files or directories..

Workflow Finished

Here lie the images showing the output of the workflow both in the **Skype** application and in Node-RED.





```
7/22/2020, 2:50:21 PM   node: 5084ce85.0aef
msg.payload : Object
▾object
  element: null
  host: "laptop-jhviep04"
  fqdn: "laptop-jhviep04"
  filepath: "C:\BPA\Workflow
  Examples\Windows
  Automation\input_files\William
  Strunk Jr., E. B. White, Roger
  Angell - The Elements of Style,
  Fourth Edition-Longman (1999).pdf"
 ▸user: object
```

3.11.3 Image Search / OCR

In this example, the user can practice OCR on a image loaded from a file and then check the results against a validation standard.

To reduce the scope of the activity, this workflow example's only purpose is to check whether the detected document is a driver's license or not.

The validation procedure is as follows: if the text "DRIVER" is located within tzhe image, OpenRPA will assume this document is indeed a driver's license

---

Note: As well known, OCR is **not** guaranteed to work 100% of the time, as text readability and image quality both play a crucial role in its success rate.

Load Image from File

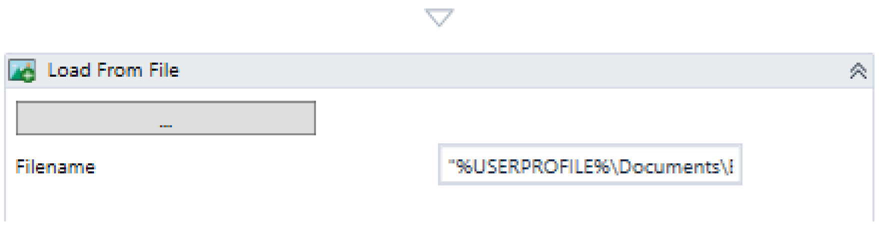
Drag the **Load From File** activity to the main sequence.

Click the `...` button to select the image to be loaded.

Open the **Properties tab** and assign a variable to the `Result` parameter which will carry the `ImageElement` created by the activity.

It is useful to remind the user that everything captured from an activity that returns any kind of `Element` is automatically mapped to the `item` Element.

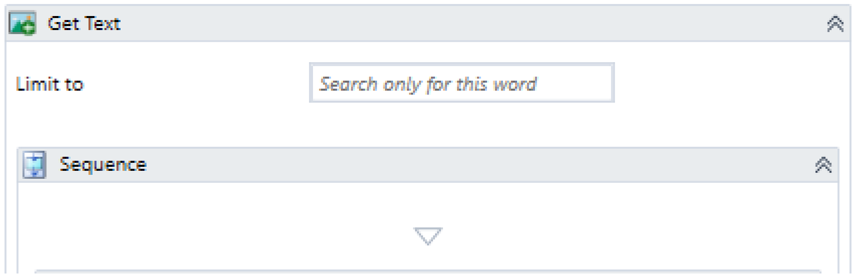
Note: While creating a new variable in the **Properties box**, after typing the variable name and with the caret still inside the input field, by pressing `Ctrl+K` OpenRPA will automatically create this new variable with the correct type required for the activity.



Filename "%USERPROFILE%\Documents\

Get Text

Drag **Get Text** activity to the sequence inside the **Load From File** activity.



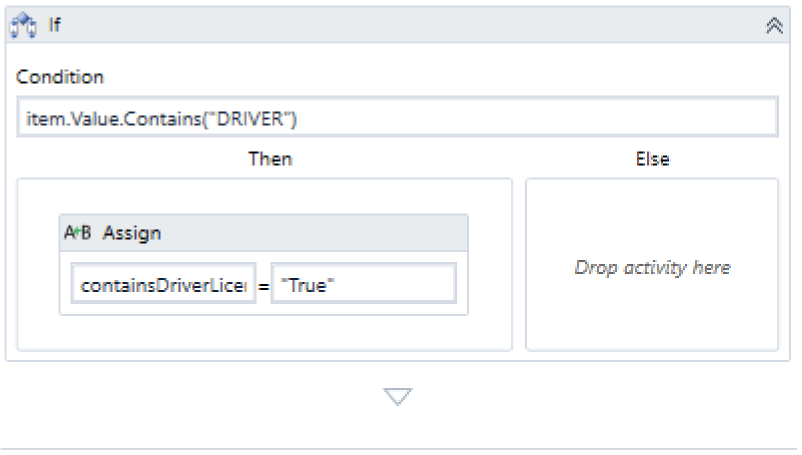
Evaluation

Drag an **If** activity to the sequence inside the **Get Text** activity. Here the user will implement the whole validation logic.

Inside the `Condition` input field, insert `item.Value.Contains("DRIVER")`. Here it is checked whether any of the text acquired using the **Get Text** activity - ie. inside `item.Value` - contains the word `DRIVER`.

Drag an **Assign** activity to the `Then` sequence inside the **If** activity. Assign the variable `containsDriver-License` to `True`. This means that if the workflow finds any instances of the values that contains `DRIVER` it will automatically change the value of the variable `containsDriverLicense` to `True`. Else, it will keep executing, since there is nothing contained inside the `Else` sequence.

Note: The **Assign** activity is used to reference a value to a variable. To do that, enter in the left-hand side of the activity the name, or alias, of the variable which you want to map a value to. If desired, you can press `Ctrl+K` to automatically create the variable or create it manually. Use the right-hand side to enter the value which will be assigned to the variable.

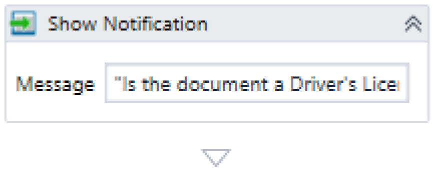


Show Notification

Now drag a **Show Notification** activity to the end of the main sequence. This will be the way of showing to the user whether the validation has been successful or not.

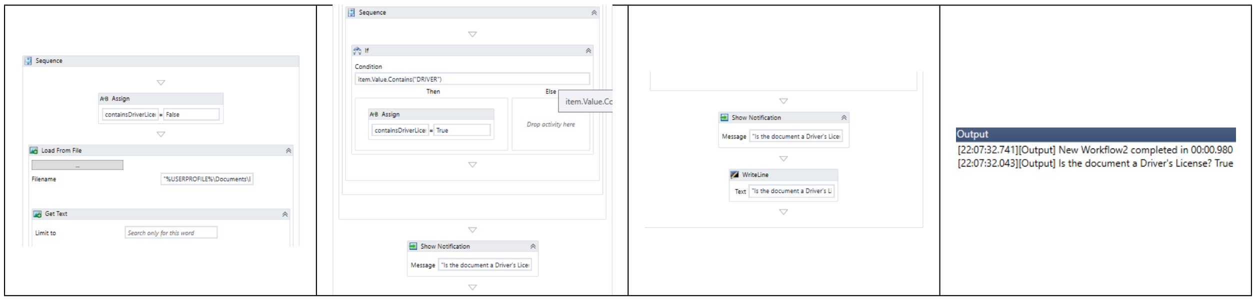
Inside the `Message` input field, in the activity´s dialogue or in the **Properties box**, insert the text `"Is the document a Driver's License? " + containsDriverLicense.ToString`.

Here we cast the value of `containsDriverLicense`, which is a `bool`, to a `string` so it can be printed in the **Show Notification** activity.



Workflow Finished

Here's a preview of how the workflow will look like after running it - the Output window is obviously detached for exhibiting the data.

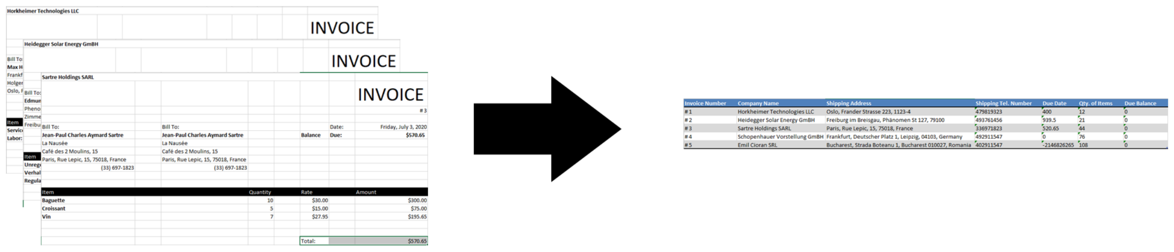

Sequence
Assign
containsDriverLice = False
Load From File
Filename
"%USERPROFILE%\Documents\
Get Text
Limit to
Search only for this word
If
Condition
item.Value.Contains("DRIVER")
Then
Else
Assign
containsDriverLice = True
Drop activity here
Show Notification
Message
"Is the document a Driver's Lice
WriteLine
Text
Output

```
[22:07:32.741][Output] New Workflow2 completed in 00:00.980
[22:07:32.043][Output] Is the document a Driver's License? True
```

3.11.4 Excel Read

In this example, the user will read a couple of invoices contained in `Excel files (.xlsx)`, capture the data inside them and generate a new `Excel file` with the merged data.

The invoices which will be used to demonstrate this example can be found here Invoices (`https://drive.google.com/file/d/1WkBjsWhA5Dm_fp5zg_kdlBh9oe2vKnTO/view?usp=sharing`).



Invoice Number	Company Name	Shipping Address	Shipping Tel. Number	Due Date	Qty. of Items	Due Balance
# 1	Horkheimer Technologies LLC	Oslo, Frander Strasse 223, 1123-4	479819323	400	12	0
# 2	Heidegger Solar Energy GmBH	Freiburg im Breisgau, Phänomen St 127, 79100	493761456	939.5	21	0
# 3	Sartre Holdings SARL	Paris, Rue Lepic, 15, 75018, France	336971823	520.65	44	0
# 4	Schopenhauer Vorstellung GmBH	Frankfurt, Deutscher Platz 1, Leipzig, 04103, Germany	492911547	0	76	0
# 5	Emil Cioran SRL	Bucharest, Strada Boteanu 1, Bucharest 010027, Romania	402911547	-2146826265	108	0

Unpack the `.rar` file at the desired directory and save the folder path somewhere, it shall be used later on.

The following steps are followed to accomplish this workflow example.

1. Create an Empty DataTable.
2. **For each Invoice.**
   - Read Individual Cells
   - Read Table Cells ("items")
   - Add this iteration's invoice to DataTable
3. Write Excel from DataTable

Create an Empty DataTable.

All the Excel data that will be read from the invoices will be saved into a `DataTable`.

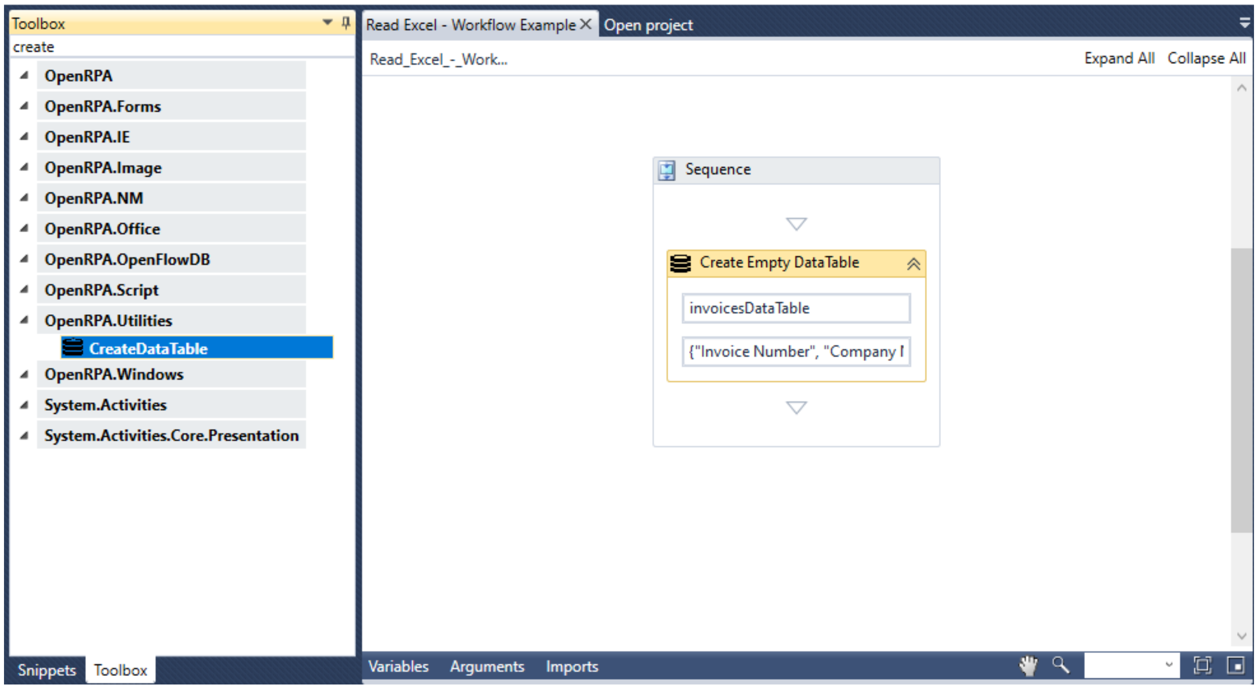
Drag the **CreateDataTable** activity to the end of the main sequence.

In the first input field, assign the `DataTable` name as `invoicesDataTable`.

Note: While creating a new variable in the **Properties box**, after typing the variable name and with the caret still inside the input field, by pressing `Ctrl+K` OpenRPA will automatically create this new variable with the correct type required for the activity.

In our case, the fields which are going to be assigned to the second input field are headers. Therefore our second input field should contain `{"Invoice Number", "Company Name", "Shipping Address", "Shipping Tel. Number", "Due Date", "Qty. of Items", "Due Balance"}` as its content.

It is advisable to rename the `DisplayName` for each activity to something meaningful and that helps you keep track of what each part of the workflow does! Considering that, rename this activity to "Create Empty DataTable".



invoicesDataTable

{"Invoice Number", "Company I

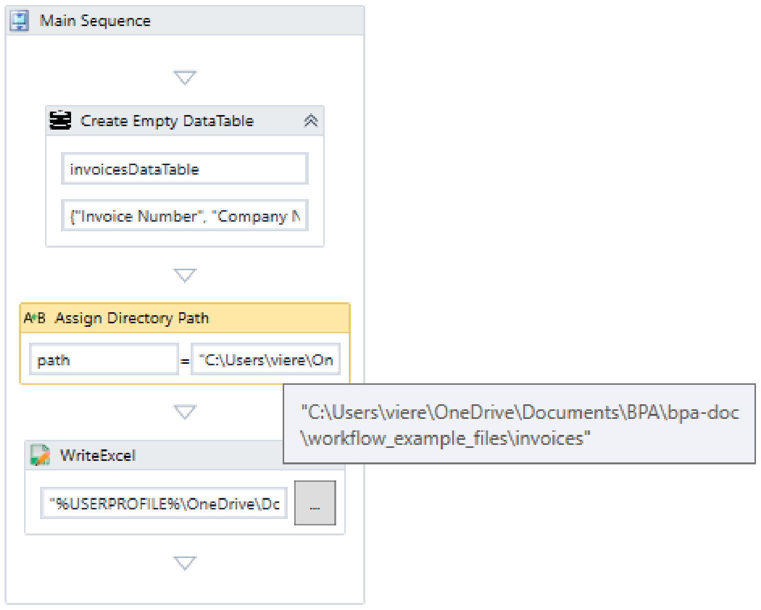
Note: In OpenRPA, a `DataTable` corresponds to a 2-dimensional non-serializable object used to hold values. Think of it as a matrix with `i` rows and `j` columns. The `i` rows are accessible by using the `.Rows(n)` attribute, where `n` is the number of the row - starting from 0 - which you want to access. The `j` columns are accessible by using the `.Item(n)` attribute, where `n` is the number of the column - starting from 0 - which you want to access. It is useful to remind here that the user must have accessed the `Rows` attribute, therefore, if the user wanted to access the 2nd column on the 3rd row of the datatable he would use `.Rows(2).Item(1)` attribute.

Note: In the .NET API, a `DataTable object` corresponds to one table of in-memory data contained in a `DataSet object` from the class equally named. When you read an `Excel file`, it is automatically assigned to a `DataSet` class and all the `Sheets` are saved into `DataTable` classes - numbered from 0 to the quantity of sheets inside the `Excel file`. Here it is reminded that `DataTable` objects are case sensitive.

For each Invoice

The first activity will loop a directory and will return all filenames contained in it.

First drag the **Assign** activity to assign the absolute path of the directory you want to loop through to a variable - ie. `path`.



Main Sequence

Create Empty DataTable

invoicesDataTable

{"Invoice Number", "Company N

Assign Directory Path

path = "C:\Users\viere\On

"C:\Users\viere\OneDrive\Documents\BPA\bpa-doc
\workflow_example_files\invoices"

WriteExcel

"%USERPROFILE%\OneDrive\Dc

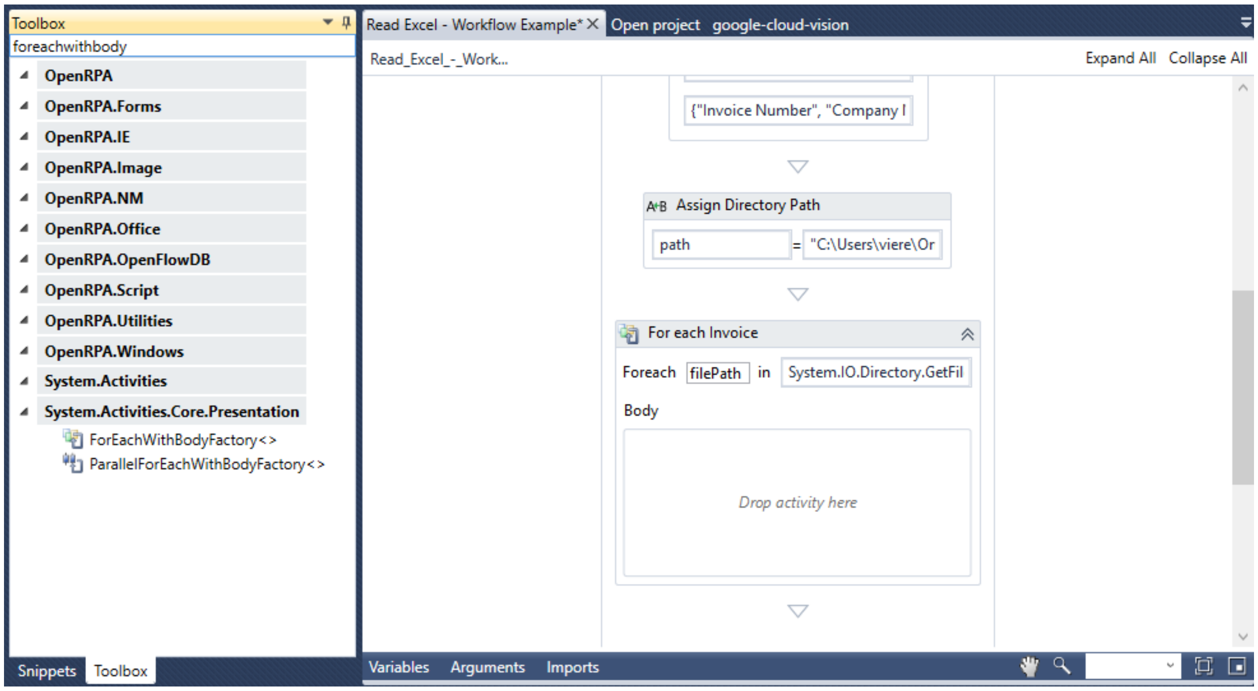
Drag the **ForEachWithBodyFactory<>** activity to the main sequence.

Set the proper `TypeArgument` parameter inside the **Properties box** to `String`. This is done because the directory which will be iterated through contain filepaths, which are returned as values of `String` type.

In the right-hand side of the activity change the `item` variable assigned to `filePath`.

In the left-hand side of the activity insert `System.IO.Path.GetFiles(path)`.

The variable `filePath` now contains the `String` to each respective invoice full filepath.



Read Individual Cells

Here the robot will read individual cells pertaining to fields that contain only a single value - these fields are static (the number of rows is fixed - one).

As these steps are quite repetitive, we will be reproducing only the steps pertaining to the `invoiceNumber` variable. If you want to practice further, we strongly recommend doing the same to the other variables as well, namely `companyName`, `shippingAddress`, `shippingTelephoneNumber` and `dueDate`.

Use the Recorder to select the cell or drag the **ReadCell** activity, found inside the `OpenRPA.Utilities` toolbox, to the sequence inside the **ForEachWithBodyFactory<>** activity.

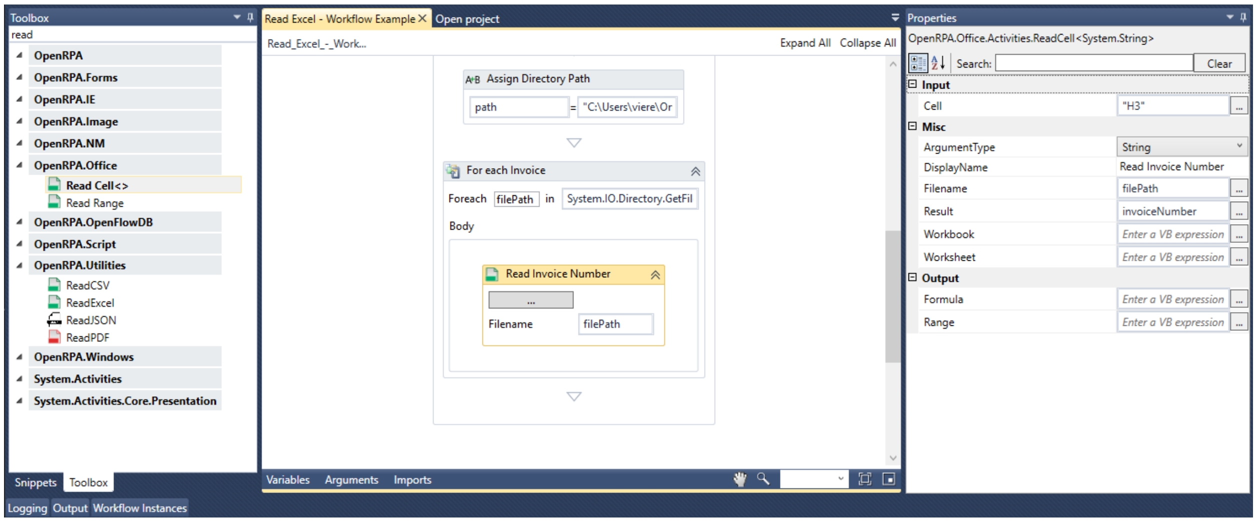
If the Recorder wasn't used, the user must follow a few more steps in order to use this activity. First, assign the `Filename` parameter inside the **Properties box** to the `filePath` variable to select the `Excel file` which will be read. Enter the cell which will be captured into the `Cell` parameter. In our case, it is `"H3"`, which corresponds to the invoice number.

The **ReadCell activity** will read from an `Excel file` and save the cell into a `String` variable by default. The user can change the type of the data captured by changing the `ArgumentType` parameter inside the **Properties tab**.

Since this variable does not exist yet, we must first create it. A reminder that variable must be obligatorily set with the type **System.Data.String**.

Simply click in the `Result` input field inside the **Properties tab** and type the name of the variable you want to use. In this case, it is `invoiceNumber`.

Now repeat the procedure above for the remaining variables.



Assign Directory Path: path = "C:\Users\viere\Or

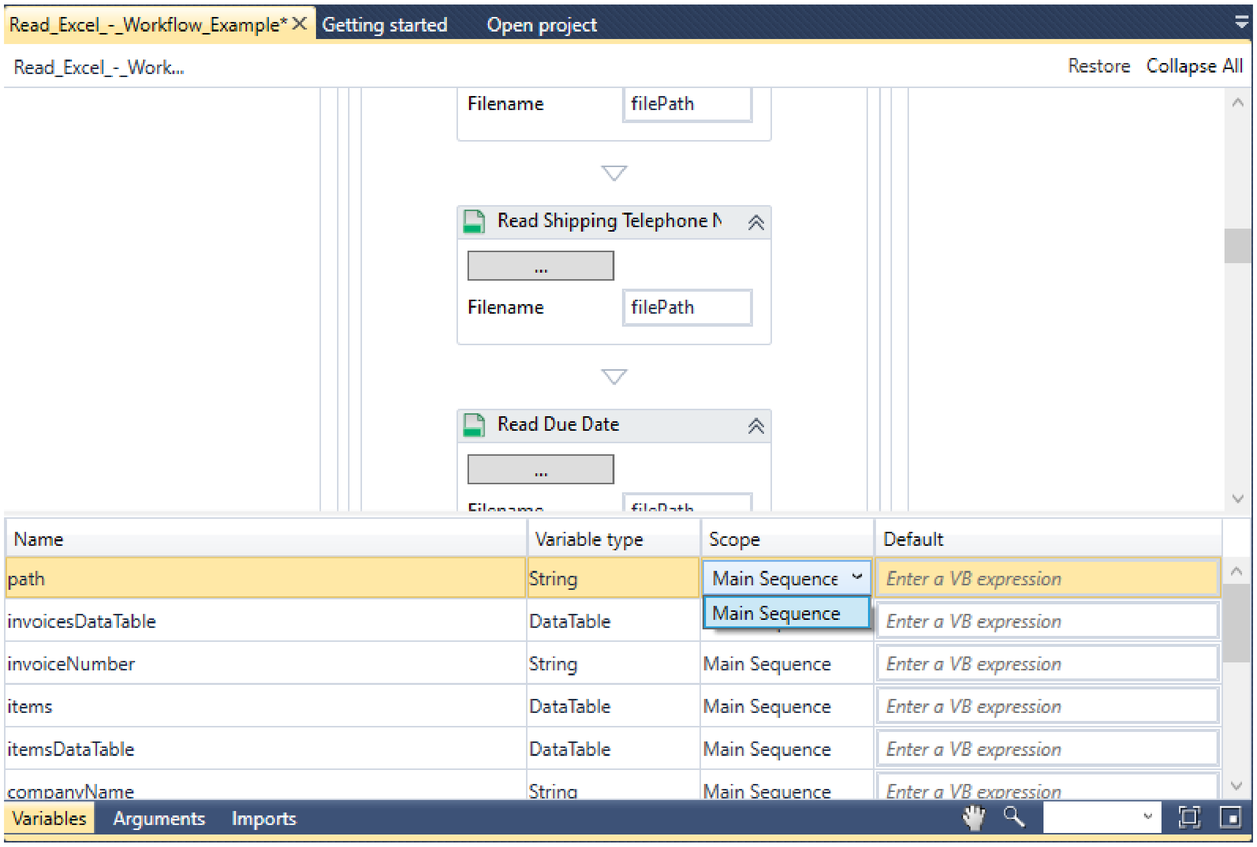
Foreach filePath in System.IO.Directory.GetFil

Read Invoice Number — Filename: filePath

OpenRPA.Office.Activities.ReadCell<System.String>

Input	
Cell	"H3"
Misc	
ArgumentType	String
DisplayName	Read Invoice Number
Filename	filePath
Result	invoiceNumber
Workbook	Enter a VB expression
Worksheet	Enter a VB expression
Output	
Formula	Enter a VB expression
Range	Enter a VB expression

Remember that it is needed to change the variables `Scope` to the outermost sequence to be able to capture all the values gathered in here and save them into a `DataTable` later on.



Filename: filePath

Name	Variable type	Scope	Default
path	String	Main Sequence	Enter a VB expression
invoicesDataTable	DataTable	Main Sequence	Enter a VB expression
invoiceNumber	String	Main Sequence	Enter a VB expression
items	DataTable	Main Sequence	Enter a VB expression
itemsDataTable	DataTable	Main Sequence	Enter a VB expression
companyName	String	Main Sequence	Enter a VB expression

Note: While creating a new variable in the **Properties box**, after typing the variable name and with the caret still inside the input field, by pressing `'K` OpenRPA will automatically create this new variable with the correct type required for the activity.

Read Table Cells ("items")

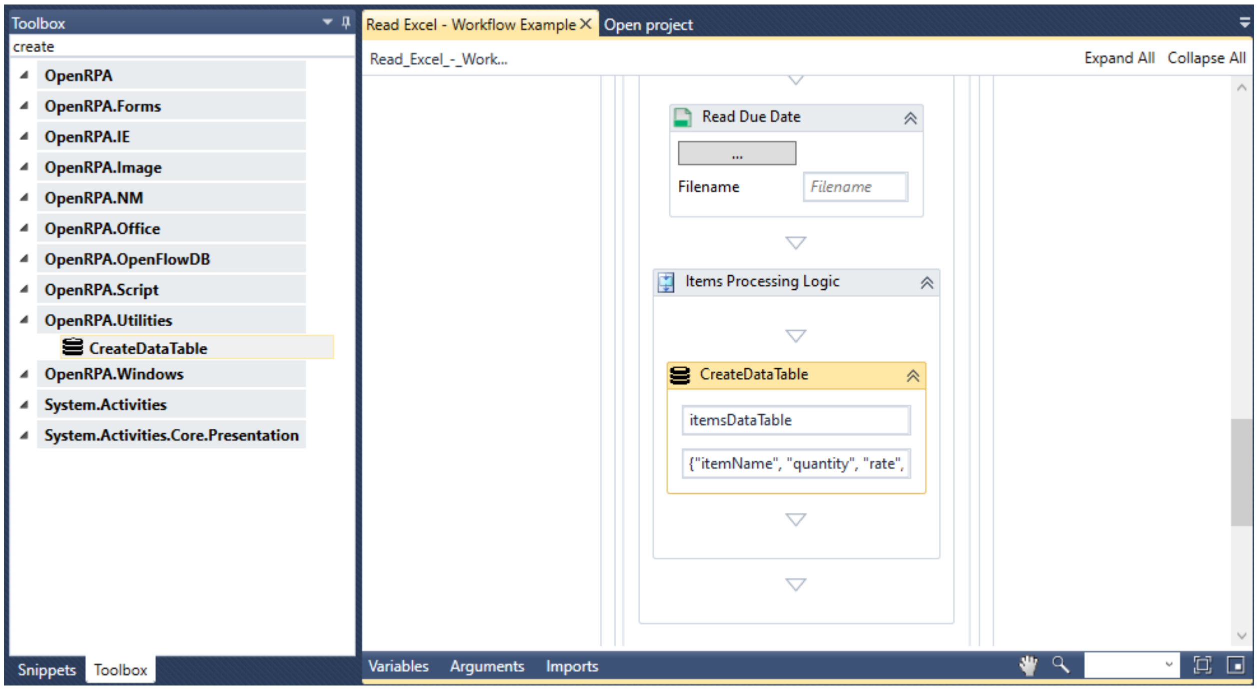
Here the robot will read a 'table' inside the invoice and differently to reading individual cells, tables are usually dynamic (the number of rows may vary) or. This is done in a programmatic manner that allows the developer the option to read a specific range of cells which would be simply too huge as to permit him to manually add the **ReadCell** activity for each variable.

This 'table' will be stored in a new `DataTable` called 'itemsDataTable' and some logic will be applied as well, such as removing empty columns, summing and counting values.

Drag a **Sequence** activity and name it `Items Processing Logic`.

Inside this sequence, drag a **CreateDataTable** activity and create a table named `itemsDataTable`.

The headers for this `DataTable` shall be `{"itemName", "quantity", "rate", "subTotal"}`.



itemsDataTable

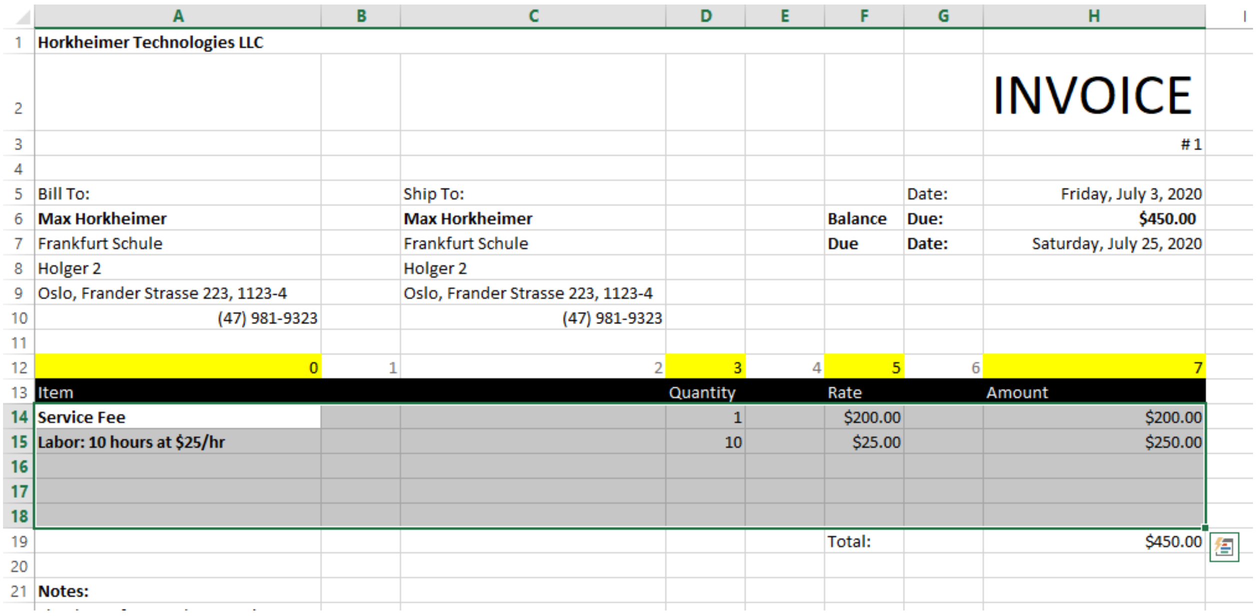
{"itemName", "quantity", "rate",

For the assignments, we will be using columns 0, 3, 5, 7, if you take a look at the invoice template, these are the indexes for the columns that contain data.

Now, by using the `Read Range` activity, we will get the table contents into a `DataTable`.

First enter the filepath of the file from which the activity will read from inside the `Filename` input field.

Enter the cells you want to capture in the `Cells` field inside the **Properties box**. In our case, those are `"A14:H18"`.

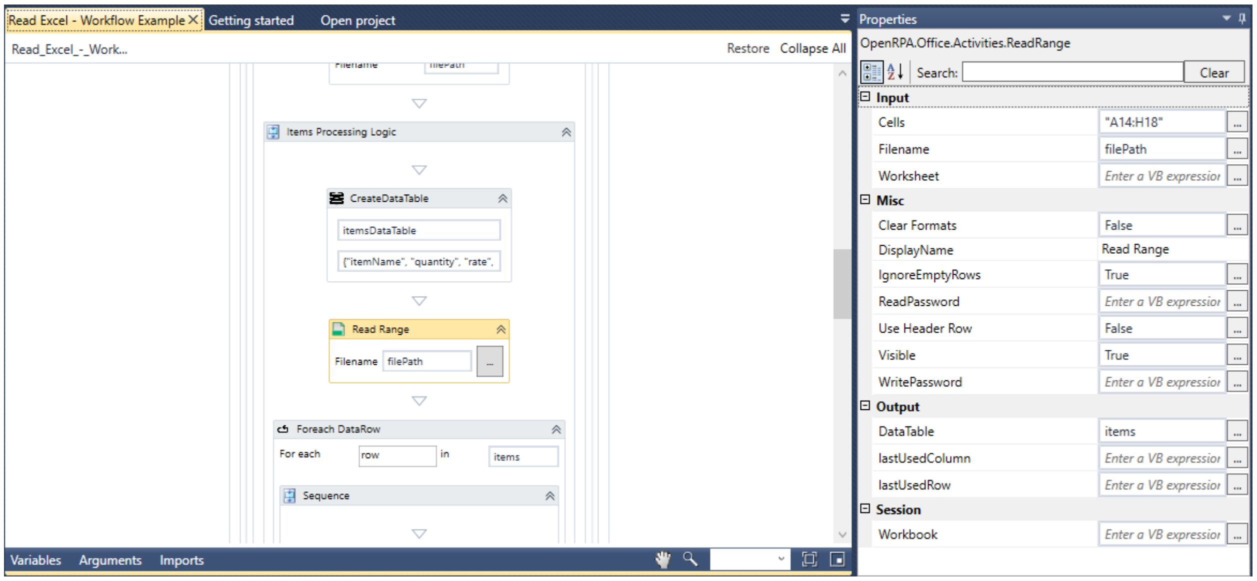


	A	B	C	D	E	F	G	H
1	Horkheimer Technologies LLC							
2								INVOICE
3								# 1
4								
5	Bill To:		Ship To:				Date:	Friday, July 3, 2020
6	**Max Horkheimer**		**Max Horkheimer**			**Balance**	**Due:**	**$450.00**
7	Frankfurt Schule		Frankfurt Schule			**Due**	**Date:**	Saturday, July 25, 2020
8	Holger 2		Holger 2					
9	Oslo, Frander Strasse 223, 1123-4		Oslo, Frander Strasse 223, 1123-4					
10	(47) 981-9323		(47) 981-9323					
11								
12	0	1	2	3	4	5	6	7
13	Item			Quantity		Rate		Amount
14	**Service Fee**			1		$200.00		$200.00
15	**Labor: 10 hours at $25/hr**			10		$25.00		$250.00
16								
17								
18								
19						Total:		$450.00
20								
21	**Notes:**							

The indexes 0, 3, 5 and 7 refer to the indexed objects inside the current `row`, ie. the columns.

Enter the name for the `DataTable` you want to save it to inside the `DataTable` field inside the **Properties box**. In our case this `DataTable` is `items`.

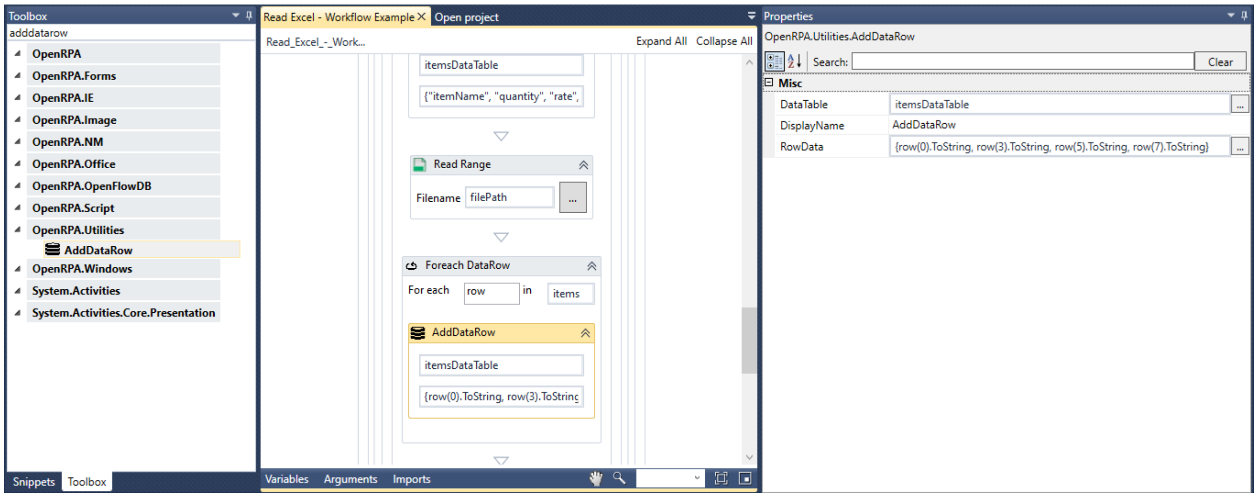
In the **Properties box**, you may find the property `IgnoreEmptyRows`, set this value to `True`. Since our table does not contain a header, set the `Use Header Row` property to `False` - this will ensure the activity reads the first row of the data (row 14).



OpenRPA.Office.Activities.ReadRange

Input	
Cells	"A14:H18"
Filename	filePath
Worksheet	Enter a VB expression
Misc	
Clear Formats	False
DisplayName	Read Range
IgnoreEmptyRows	True
ReadPassword	Enter a VB expression
Use Header Row	False
Visible	True
WritePassword	Enter a VB expression
Output	
DataTable	items
lastUsedColumn	Enter a VB expression
lastUsedRow	Enter a VB expression
Session	
Workbook	Enter a VB expression

And by looping the `DataTable` with a **ForEach DataRow** activity, the **Add DataRow** activity is used to loop through each item information to our ‘itemsDataTable’



itemsDataTable

{"itemName", "quantity", "rate",

Filename filePath

For each row in items

itemsDataTable

{row(0).ToString, row(3).ToString

OpenRPA.Utilities.AddDataRow

DataTable	itemsDataTable
DisplayName	AddDataRow
RowData	{row(0).ToString, row(3).ToString, row(5).ToString, row(7).ToString}

Now comes the logic for calculating the subtotal value and adding this value to the total value which will be described in the final `Excel sheet`.

Declare a new variable `quantity` with type `System.Decimal`. Using the **Assign** activity, assign its value to `quantity + Convert.ToDecimal(row(3).ToString)` - this will cause the variable to increment its value for each item iteration.

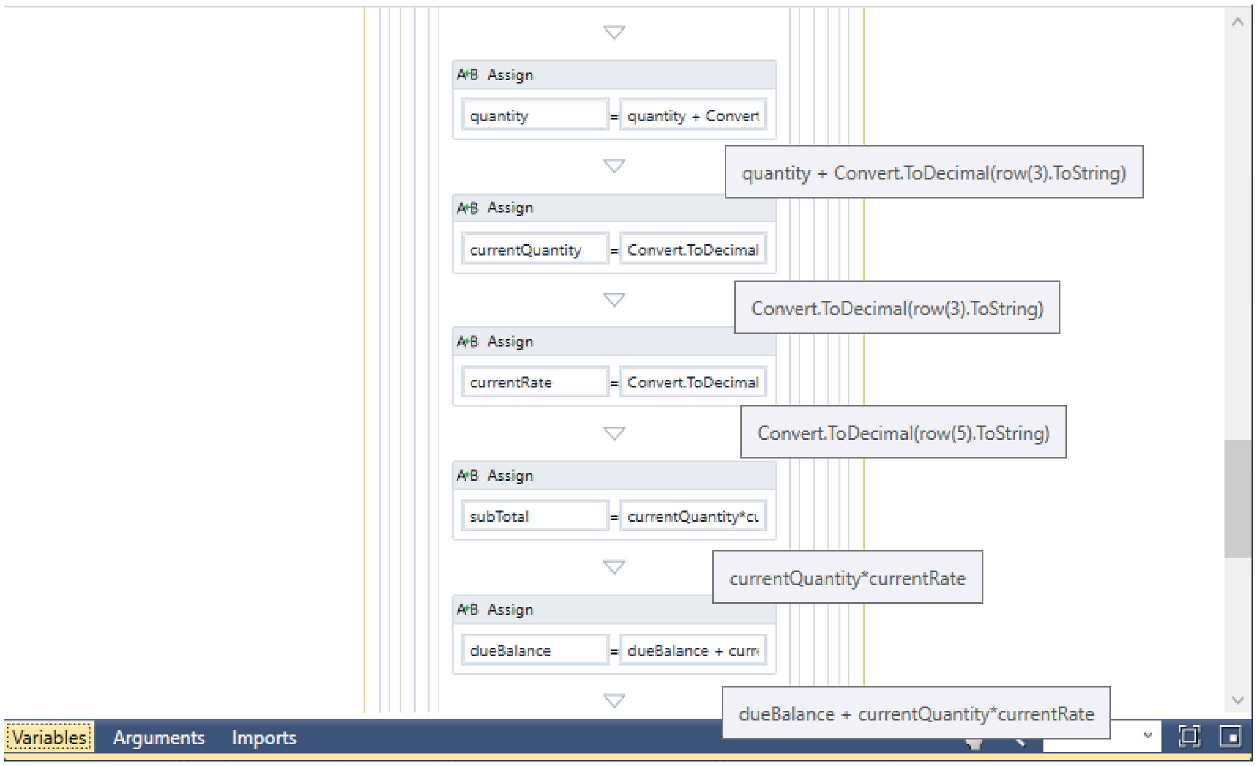
Declare a new variable `currentQuantity` with type `System.Decimal`. Using the **Assign** activity, assign its value to `Convert.ToDecimal(row(3).ToString)`

Do the step above for the variable `currentRate` as well, using `row(5)` as its base-value.

Declare a new variable `subTotal` with type `System.Decimal`. Using the **Assign** activity, assign its value to `currentQuantity*currentRate`.

Finally, declare a new variable `dueBalance` with type `System.Decimal`. Using the **Assign** activity, assign its value to `dueBalance + currentQuantity*currentRate`.

It is useful to remark that the user must change the variables `Scope` to the outermost sequence to be able to capture all the values gathered in here and save them into a `DataTable` later on.


A+B Assign
quantity = quantity + Convert
quantity + Convert.ToDecimal(row(3).ToString)
A+B Assign
currentQuantity = Convert.ToDecimal
Convert.ToDecimal(row(3).ToString)
A+B Assign
currentRate = Convert.ToDecimal
Convert.ToDecimal(row(5).ToString)
A+B Assign
subTotal = currentQuantity*cu
currentQuantity*currentRate
A+B Assign
dueBalance = dueBalance + curr
dueBalance + currentQuantity*currentRate
Variables
Arguments
Imports

Note: In the **Read Range** activity, the : colon symbol is used to delimit the area you want to capture, the cell to the left-hand side of it is the starting cell and the cell to the right-hand side of it is the ending cell.

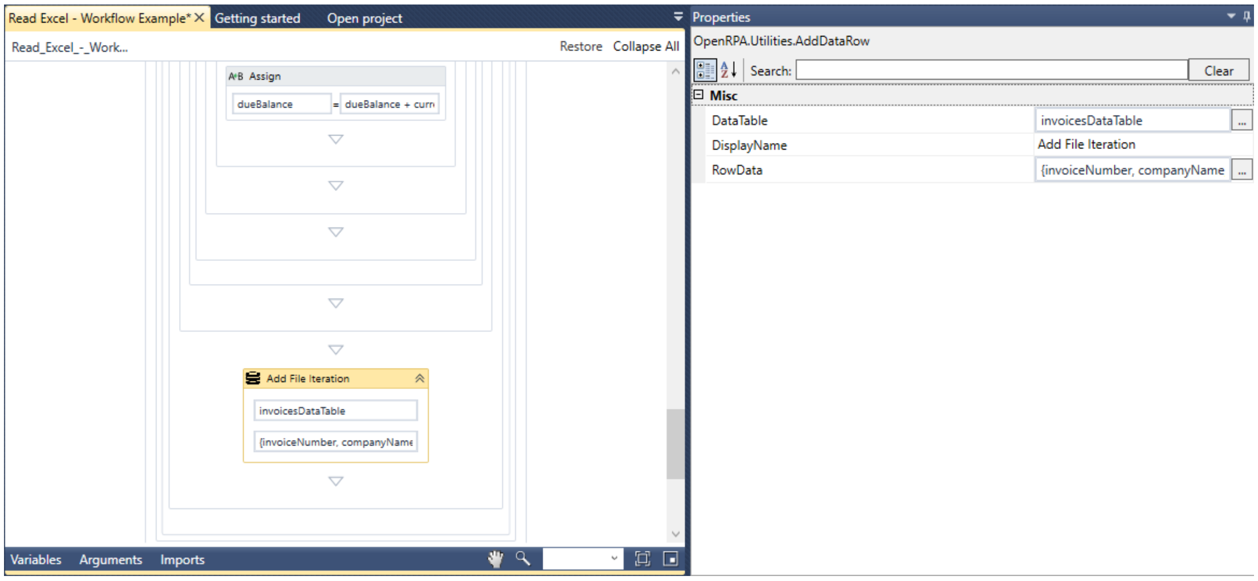
Add this iteration's invoice to DataTable

Now that we've gathered all the important data from this iteration's invoice, we want to add these info to the `DataTable` we created on step 1.

Drag the **AddDataRow** activity to the end of the sequence inside the **ForEachWithBodyFactory<>** activity.

In the first input field, insert `invoicesDataTable`.

In the second input field, insert `{invoiceNumber, companyName, shippingAddress, shippingTelephoneNumber, dueDate, quantity, total}`.



Properties
OpenRPA.Utilities.AddDataRow
Search: Clear
Misc

DataTable	invoicesDataTable
DisplayName	Add File Iteration
RowData	{invoiceNumber, companyName

Write Excel from DataTable

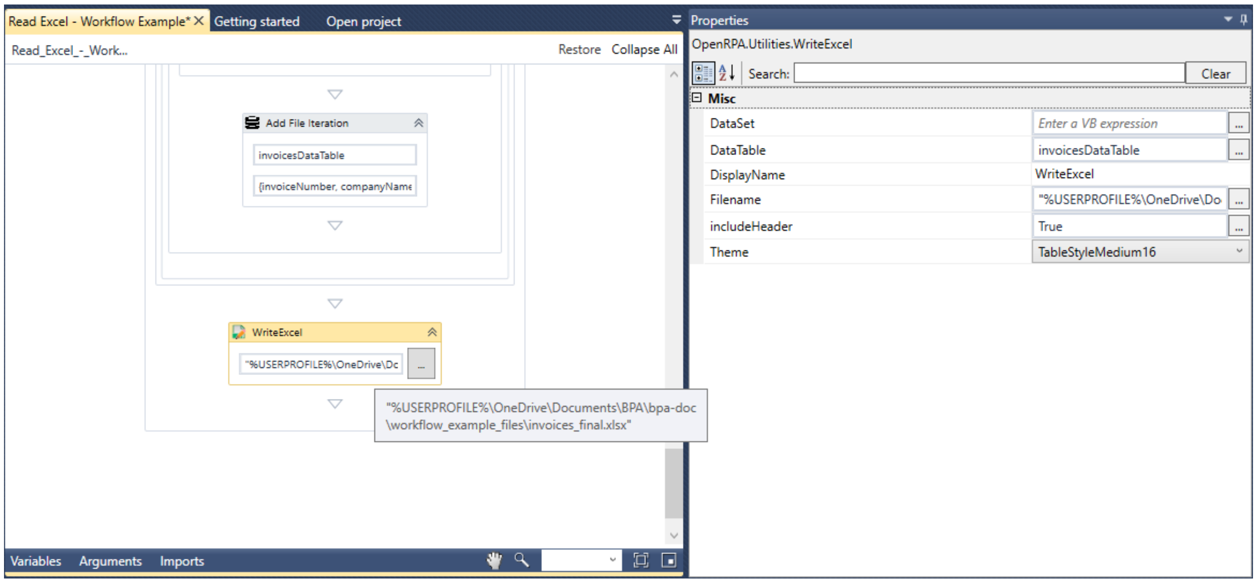
This is the last part of the workflow. Here the user will finally write the `Excel file` containing all the data gathered.

Drag a **WriteExcel** activity into the end of the main sequence.

Click the `...` button to select the filename to which the `Excel file` will be saved to. Enter the filename inside the `File name` input field on the explorer window which will appear. Then press `Save` to finally assign it.

Assign the `DataTable` from which the data gathered will be saved into the `Excel file` by entering its name inside the `DataTable` property inside the **Properties box**.

Enter the theme which will be used to stylize the `Excel file` in the `Theme` property inside the **Properties box**.



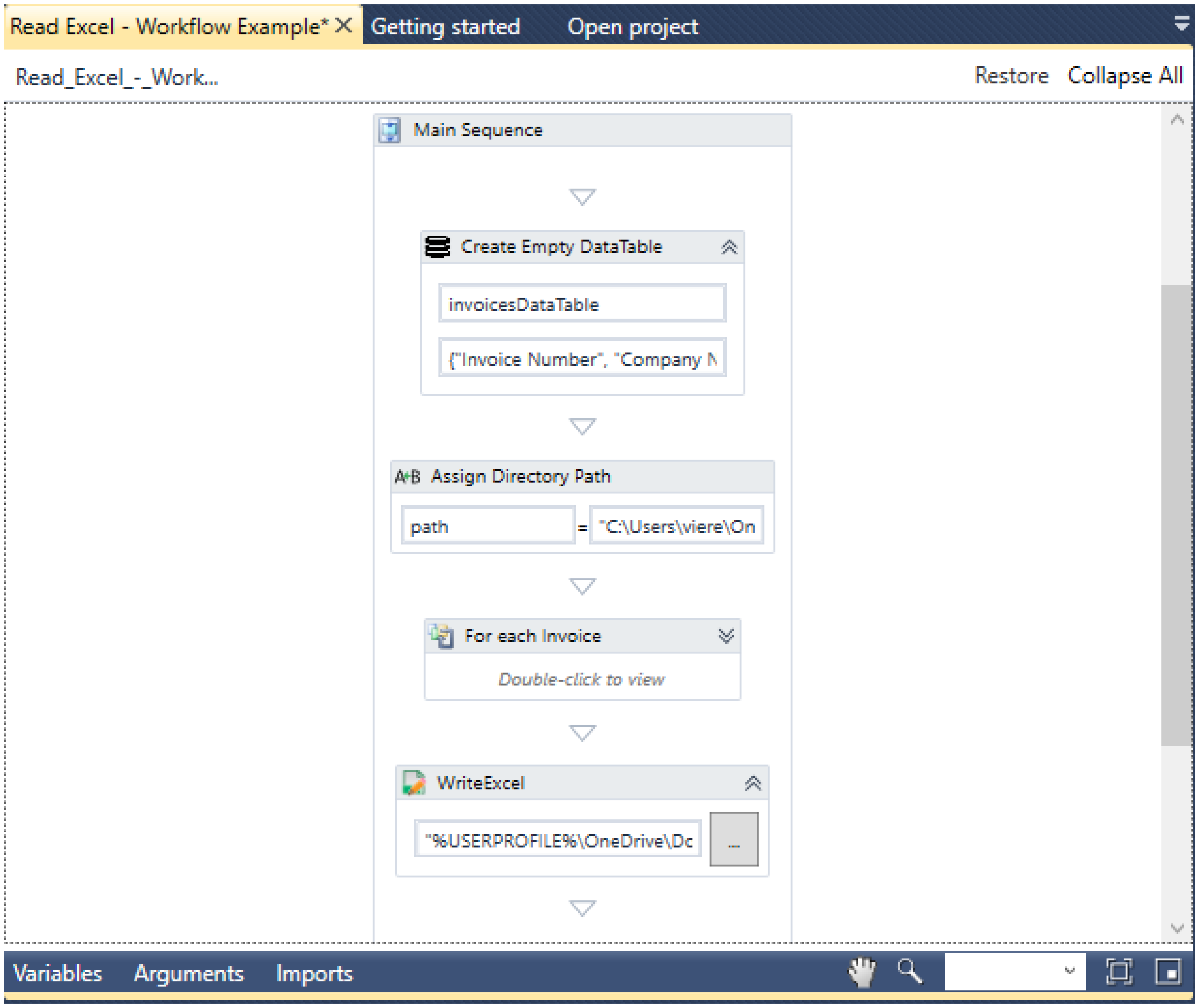
"%USERPROFILE%\OneDrive\Documents\BPA\bpa-doc
\workflow_example_files\invoices_final.xlsx"

Properties
OpenRPA.Utilities.WriteExcel
Search: Clear
Misc

DataSet	Enter a VB expression
DataTable	invoicesDataTable
DisplayName	WriteExcel
Filename	"%USERPROFILE%\OneDrive\Do
includeHeader	True
Theme	TableStyleMedium16

Workflow Finished

Here's a preview of how the workflow will look like after completing it.

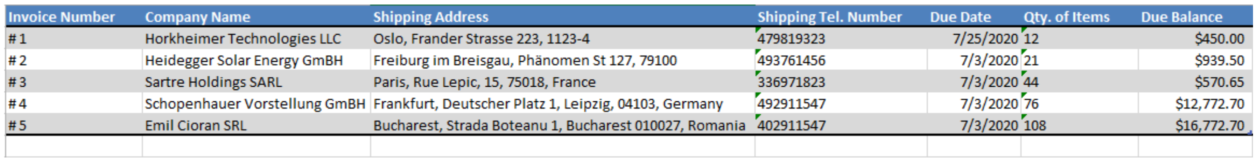


invoicesDataTable

{"Invoice Number", "Company N

path = "C:\Users\viere\On

"%USERPROFILE%\OneDrive\Dc



Invoice Number	Company Name	Shipping Address	Shipping Tel. Number	Due Date	Qty. of Items	Due Balance
# 1	Horkheimer Technologies LLC	Oslo, Frander Strasse 223, 1123-4	479819323	7/25/2020	12	$450.00
# 2	Heidegger Solar Energy GmBH	Freiburg im Breisgau, Phänomen St 127, 79100	493761456	7/3/2020	21	$939.50
# 3	Sartre Holdings SARL	Paris, Rue Lepic, 15, 75018, France	336971823	7/3/2020	44	$570.65
# 4	Schopenhauer Vorstellung GmBH	Frankfurt, Deutscher Platz 1, Leipzig, 04103, Germany	492911547	7/3/2020	76	$12,772.70
# 5	Emil Cioran SRL	Bucharest, Strada Boteanu 1, Bucharest 010027, Romania	402911547	7/3/2020	108	$16,772.70

3.11.5 MongoDB Entities Manipulation

Coming soon - Work in progress

3.11.6 File Manipulation

Coming soon - Work in progress

Remove

Coming soon - Work in progress

Save

Coming soon - Work in progress

Move

Coming soon - Work in progress

Delete

Coming soon - Work in progress

3.11.7 SOAP Service

Coming soon - Work in progress

3.11.8 REST Service (integration with Node-RED)

In this example, the user will create a workflow that connects to the Chuck Norris Jokes API (`https://api.chucknorris.io/`) and captures a joke.

The user will also create a dialog box using Forge.Forms dynamic WPF forms (`https://github.com/WPF-Forge/Forge.Forms`) where a category for the joke can be selected.

A simple validation method is also used to check whether the message is a Chuck Norris joke or not.

The steps are listed below.

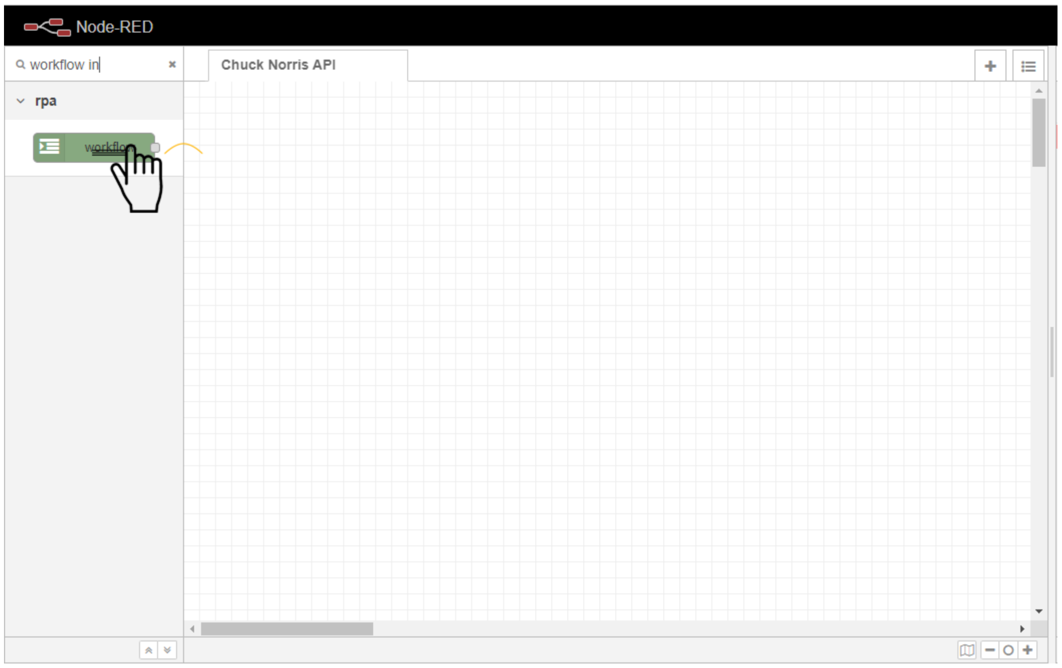
1. Configure Variable Passing To/From in Node-RED
2. Message Manipulation in Node-RED
3. OpenRPA Forms
4. Invoke Workflow through OpenRPA
5. Create Joke Failed Workflow
6. Evaluate Chuck Norris Joke
7. Invoke Joke Failed Workflow
8. Setting Evaluation Failure case in Node-RED
9. Output logic
10. Workflow Finished

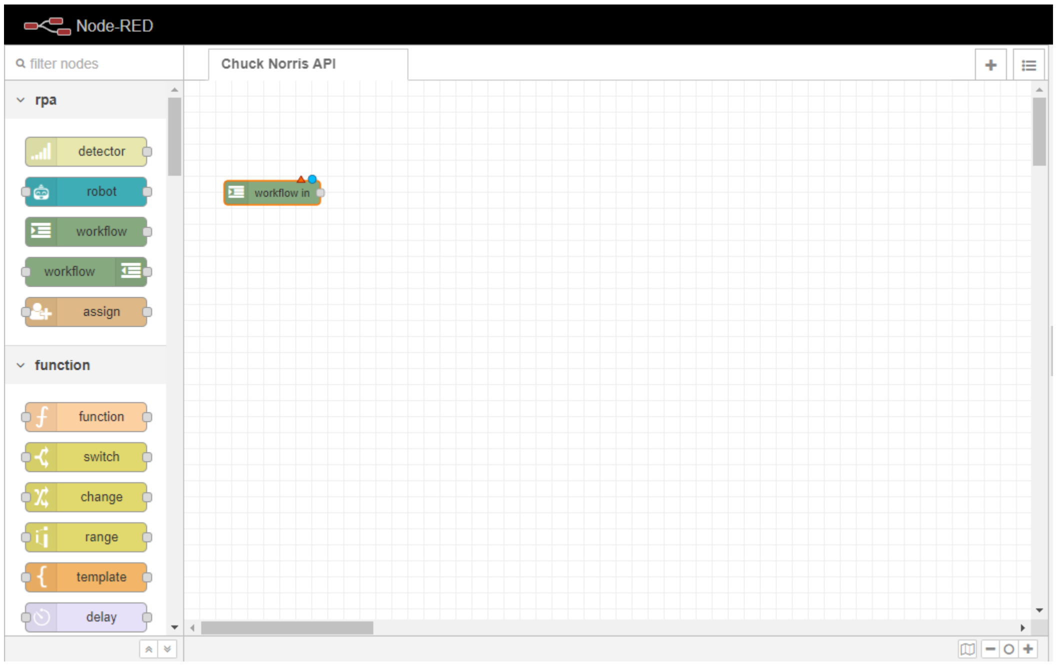
Configure Variable Passing To/From in Node-RED

Here lies the logic behind receiving data from the robot, querying the Chuck Norris API and then returning the values back to the robot.

Workflow Logic

Drag a `Workflow In` node to the workspace.



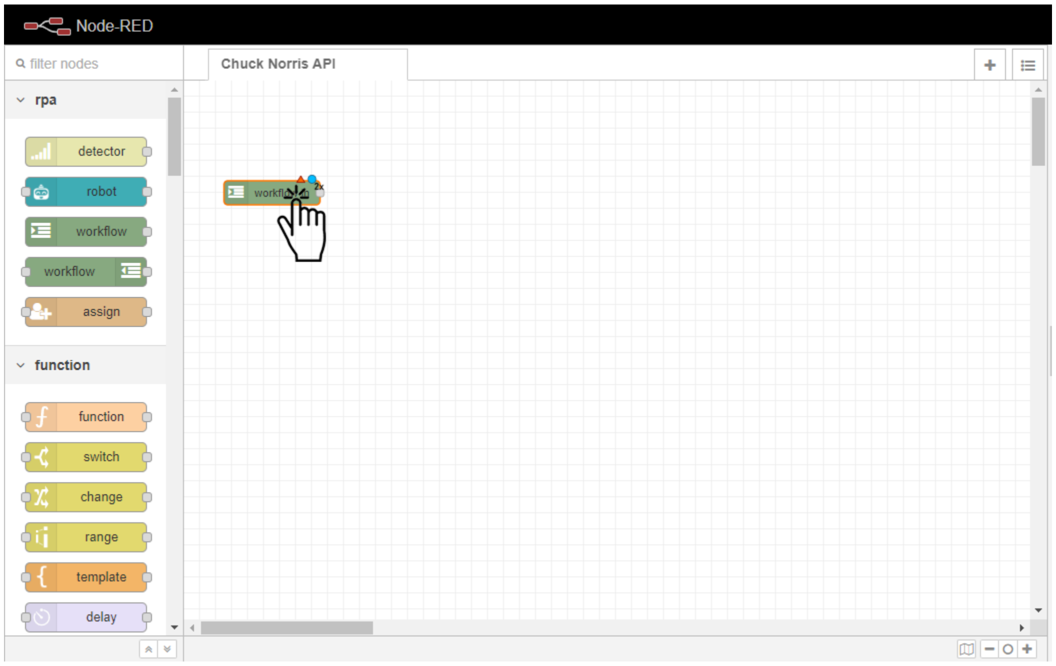


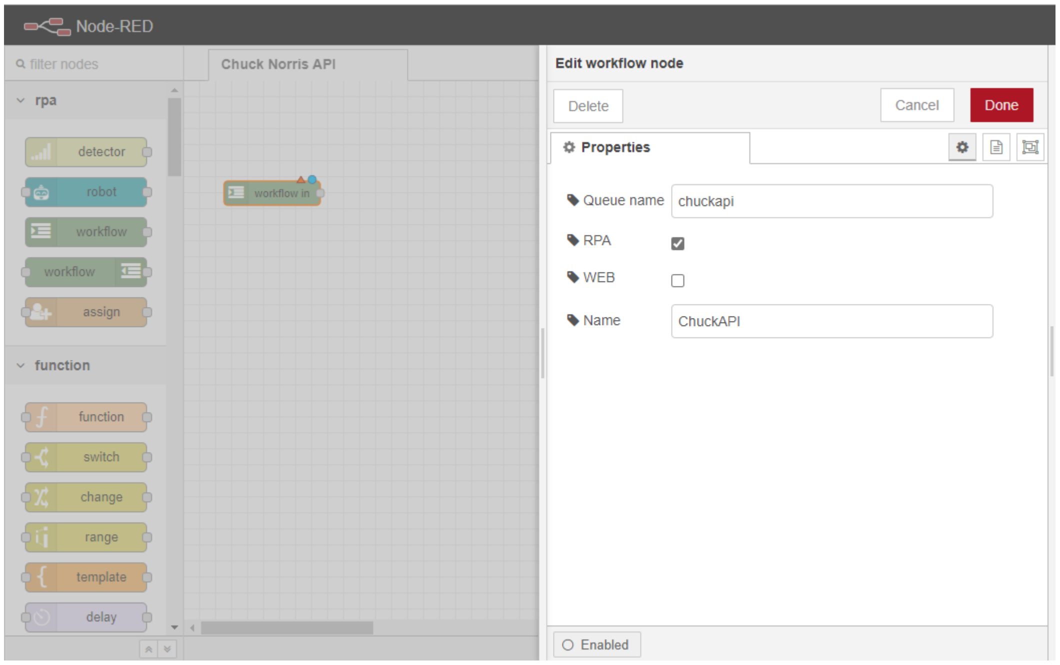
Set the `Queue name` to define the queue which the OpenFlow workflow will be assigned to in `Workflow In`'s properties. The queue name is completely arbitrary, the user can choose for any name he wishes to. In our case, it is set as `chuckapi`.

Note: To access any node's properties inside the Node-RED workspace, simply double click the given node. The **Properties box** for the node, containing each input fields required by itself will then appear.

Check the `RPA` checkbox to allow this flow's workflow to be invoked from OpenRPA agents.

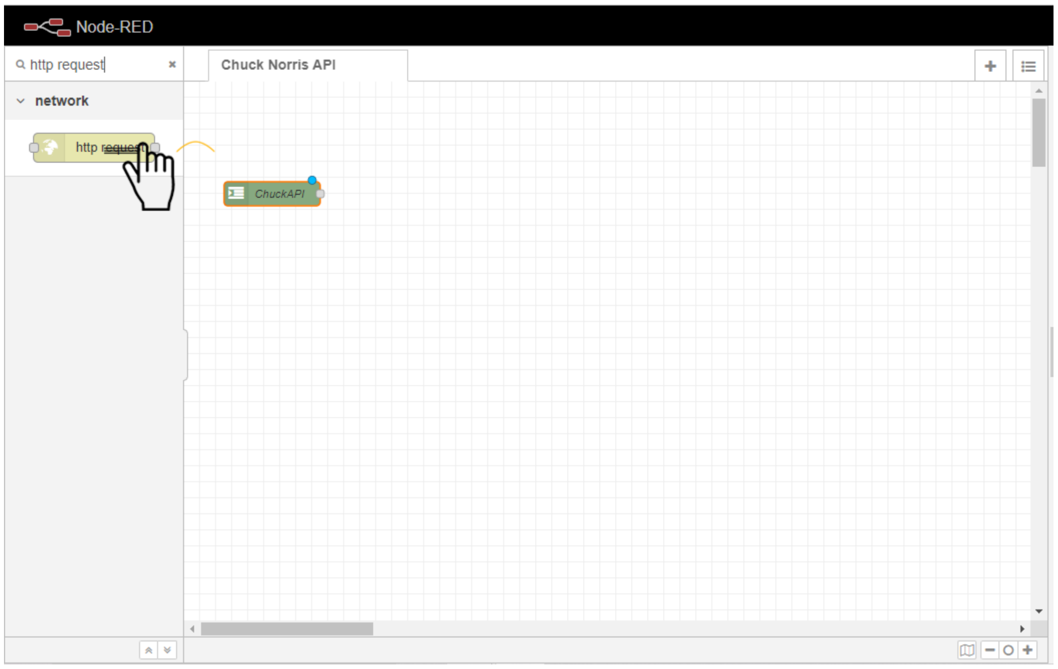
Set the name for the workflow in the `Name` input field. Again, it is named arbitrarily. In our case, it is set as `ChuckAPI`.



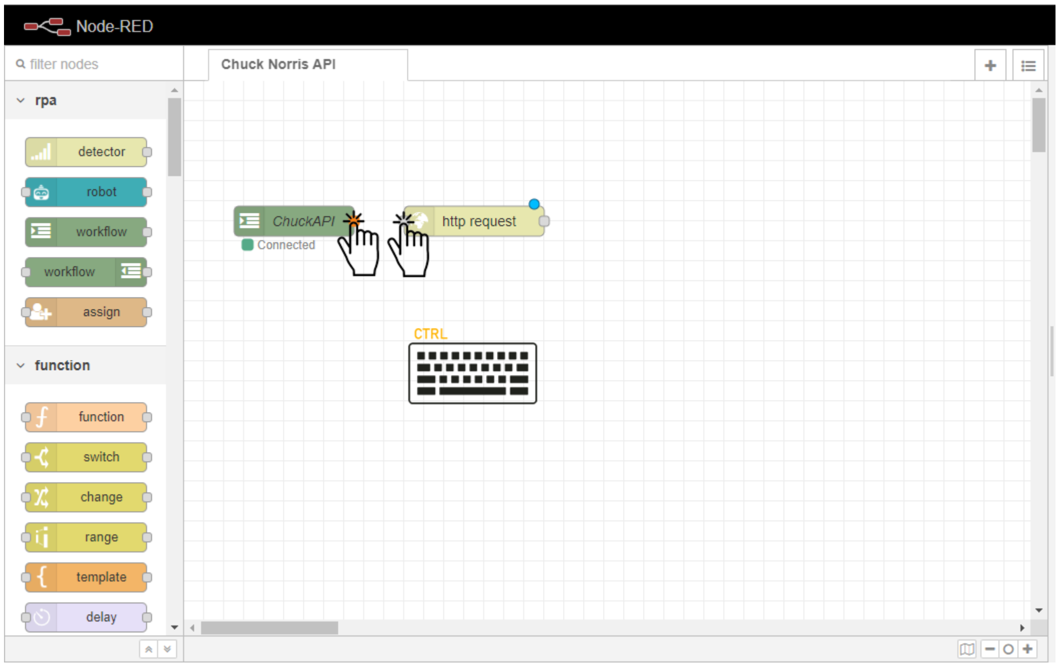


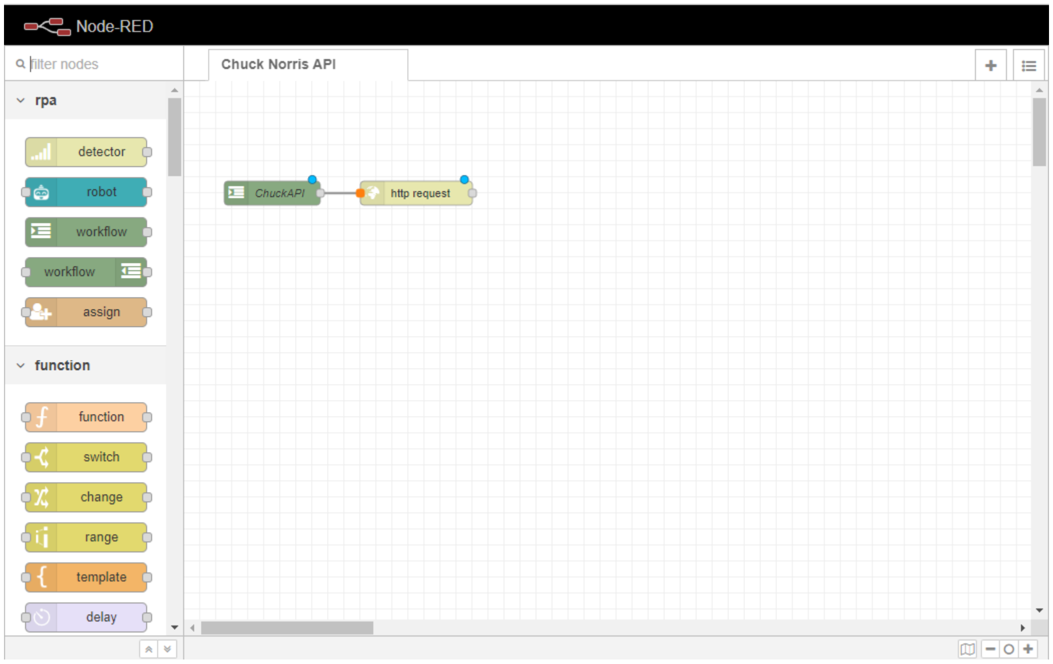
Queue name: chuckapi
RPA: ✓
WEB: ☐
Name: ChuckAPI

Drag a `HTTP Request` node to the workspace.



Connect the `Workflow In` node to the `HTTP Request` node.





Note: To easily connect a node simply press the `Ctrl` key while clicking in the grey rectangles = called ports - in the outer part of each node. The user can also click and hold to connect the nodes but the author finds it more feasible simply using the `Ctrl` key.

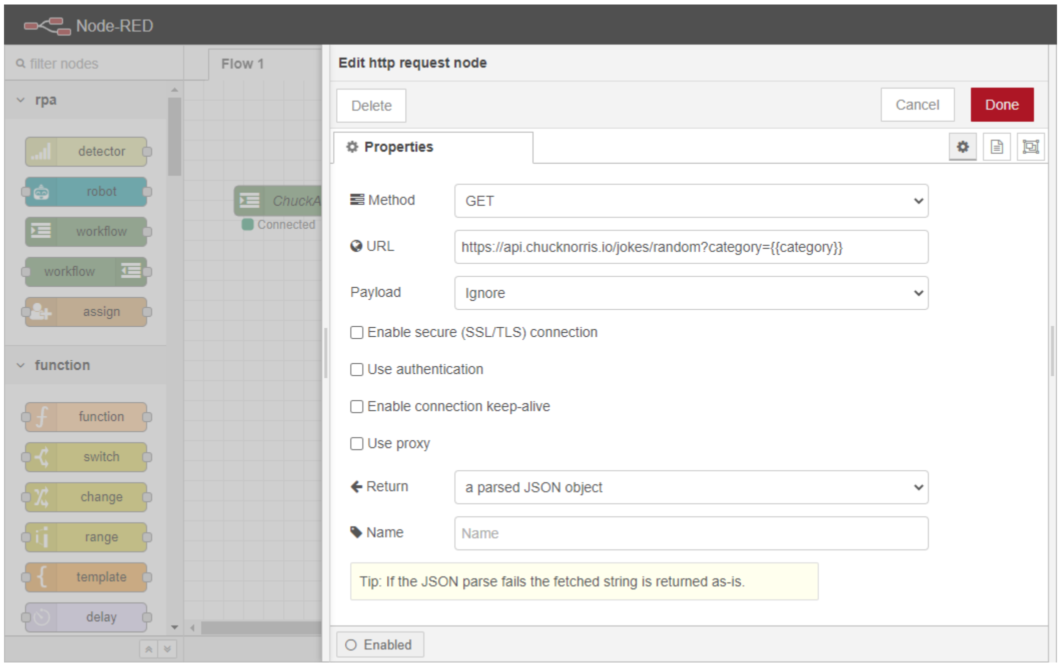
Set the `Method` for the request as `GET`.

Set the desired `URL` for the API which the request will be sent to. In our case, `https://api.chucknorris.io/jokes/random?category={{category}}`.

The variable inside the Mustache (https://mustache.github.io/mustache.5.html), ie. `category` is arbitrarily named as well.

This variable will be used by OpenRPA to pass the value of the choice captured from the user. The user may name it as per his own option.

Set the desired `Return` type which will be returned. In our case, it's `parsed JSON object`.



Edit http request node

Method: GET

URL: https://api.chucknorris.io/jokes/random?category={{category}}

Payload: Ignore

☐ Enable secure (SSL/TLS) connection

☐ Use authentication

☐ Enable connection keep-alive

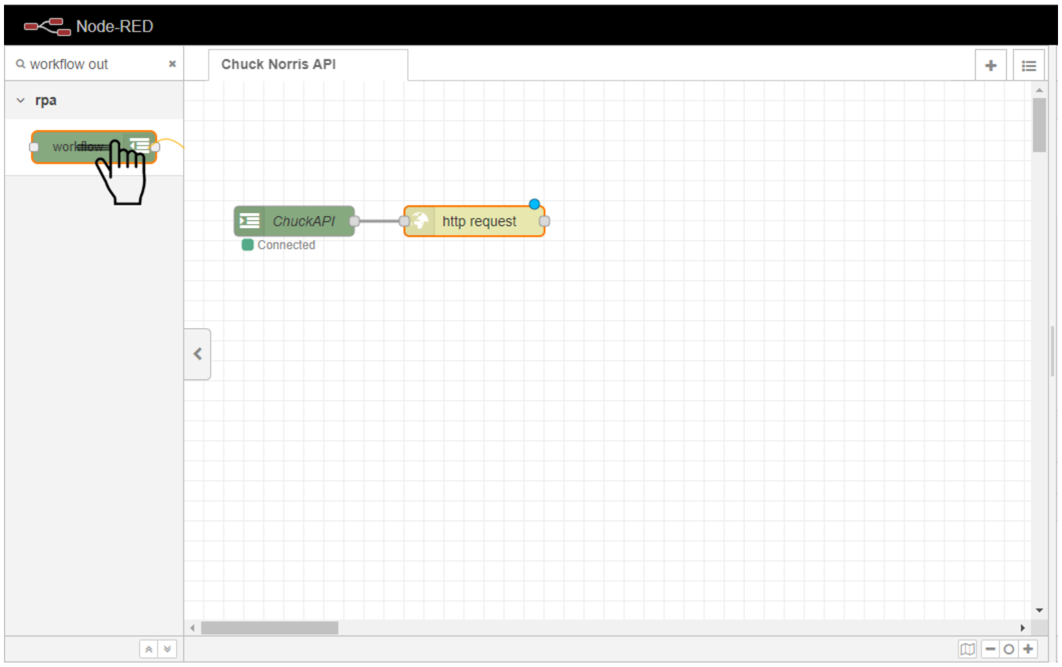
☐ Use proxy

Return: a parsed JSON object

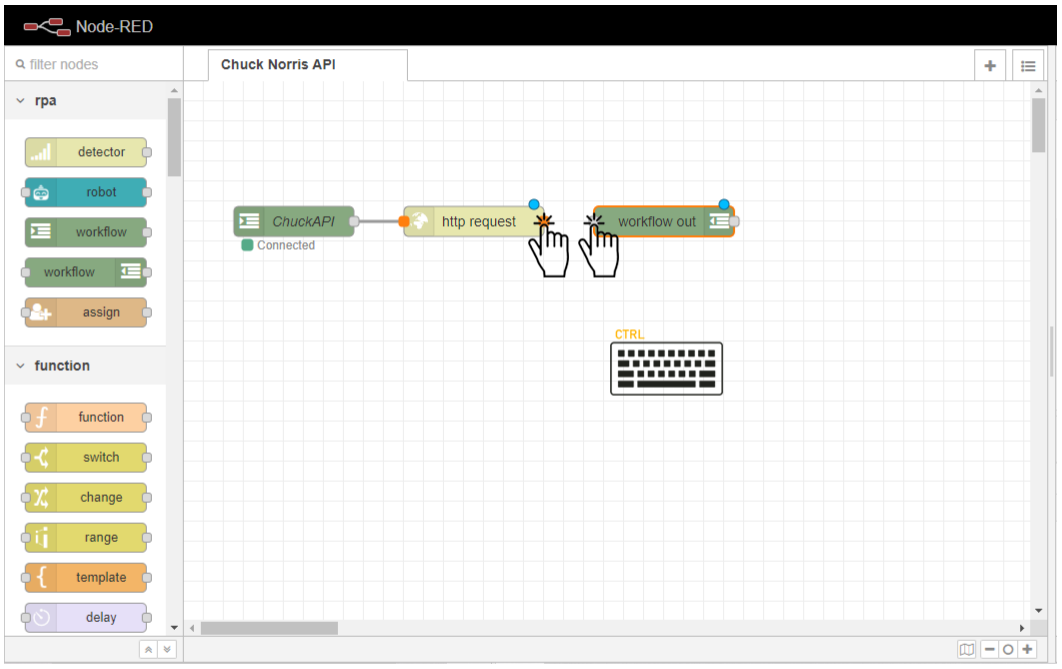
Name: Name

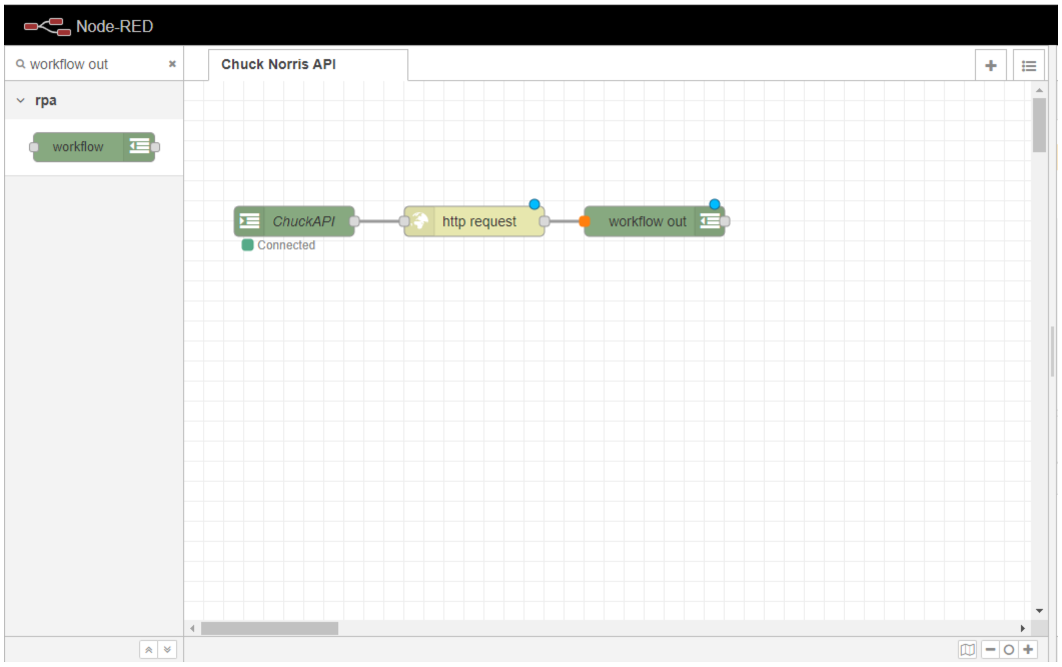
Tip: If the JSON parse fails the fetched string is returned as-is.

Drag a `Workflow Out` node to the workspace.



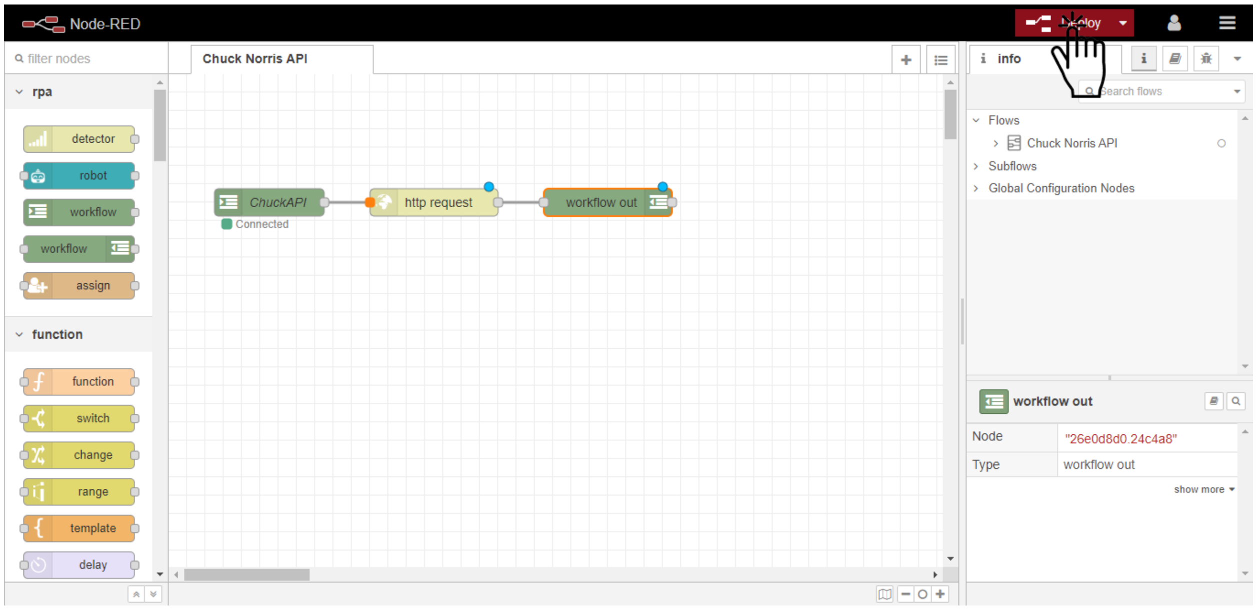
Connect the `HTTP Request` node to the `Workflow Out` node.





Note: To easily connect a node simply press the `Ctrl` key while clicking in the grey rectangles - called ports - in the outer part of each node. The user can also click and hold to connect the nodes but the author finds it more feasible simply using the `Ctrl` key.

Deploy your flow by clicking the red Deploy button at the top-left side of the screen.



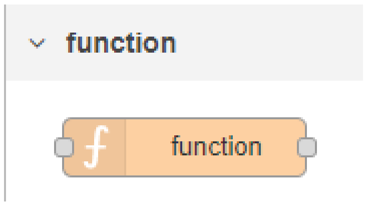
Message Manipulation in Node-RED

Now we want to send a parameter from OpenRPA to Node-RED so that our workflow is not so static.

This parameter contains a `category` to which the `Chuck Norris API`'s joke will relate.

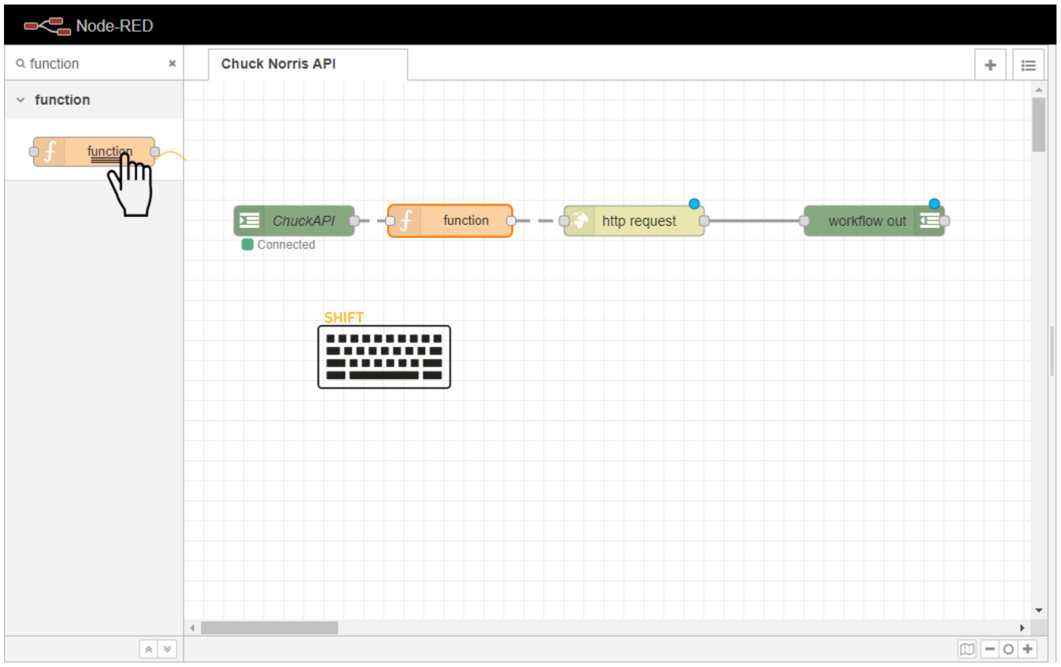
Changing parameters from a variable gathered through a workflow inside Node-RED is easily done through the `Function` node and its properties.

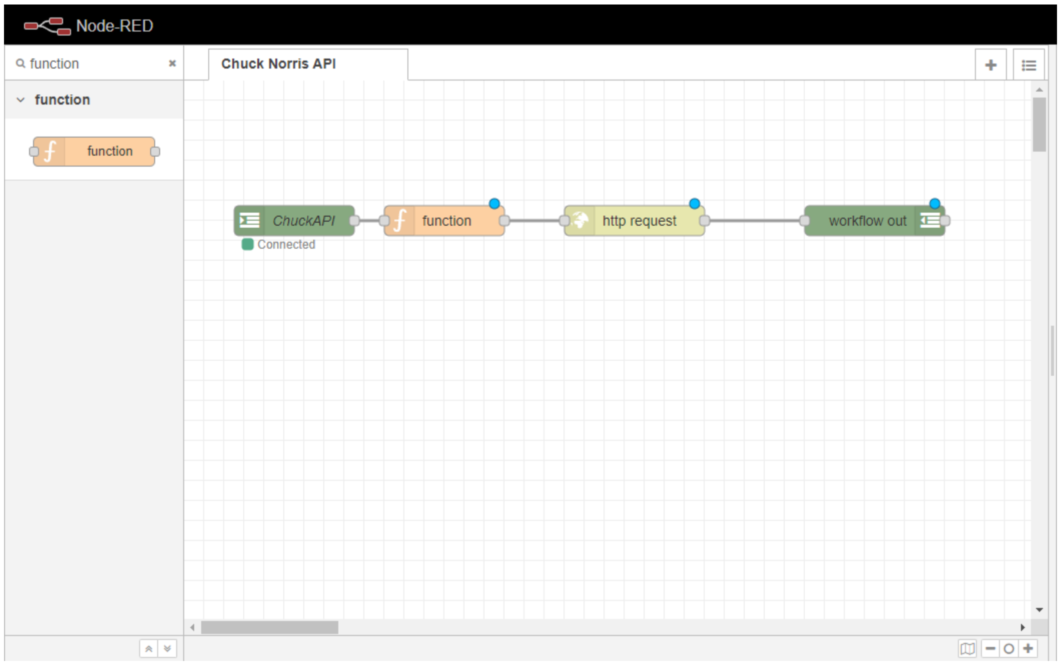
Robots `In/Out` arguments always are passed to the `msg.payload` attribute (in Node-RED) and it is easily to manipulate these by using the `Function` node.



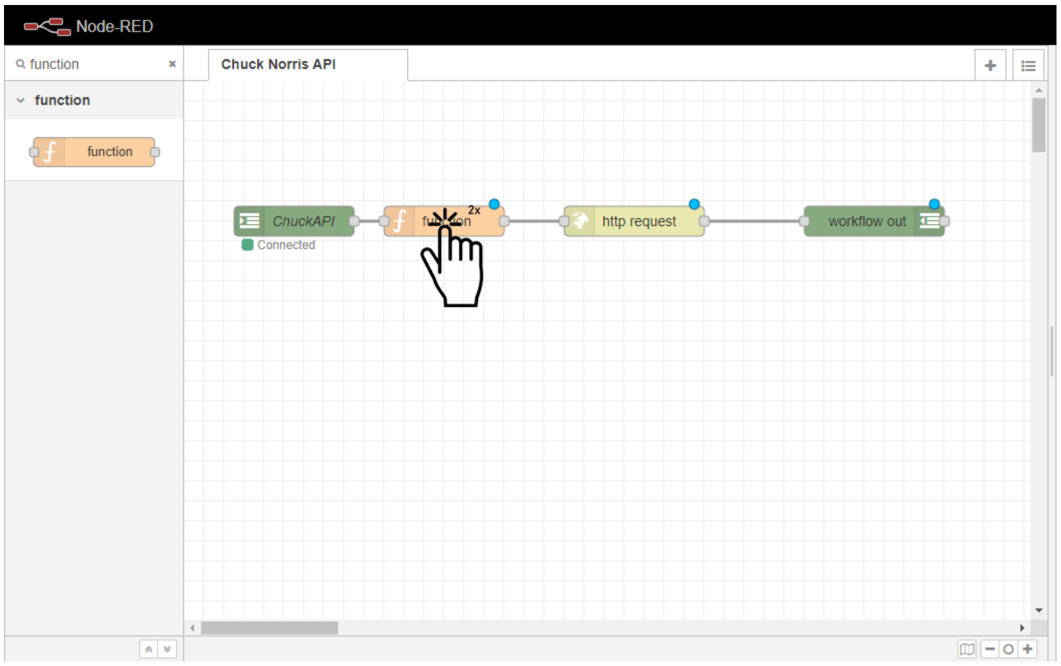
Simply drag a `Function` node to the workflow, between the **ChuckAPI** `Workflow In` node and `HTTP Request` node.

Note: To automatically add a node and bind it to the sequence, drag it while pressing the `Shift` key. This way, the node is automatically entered in-between the sequence already established before it was dragged.



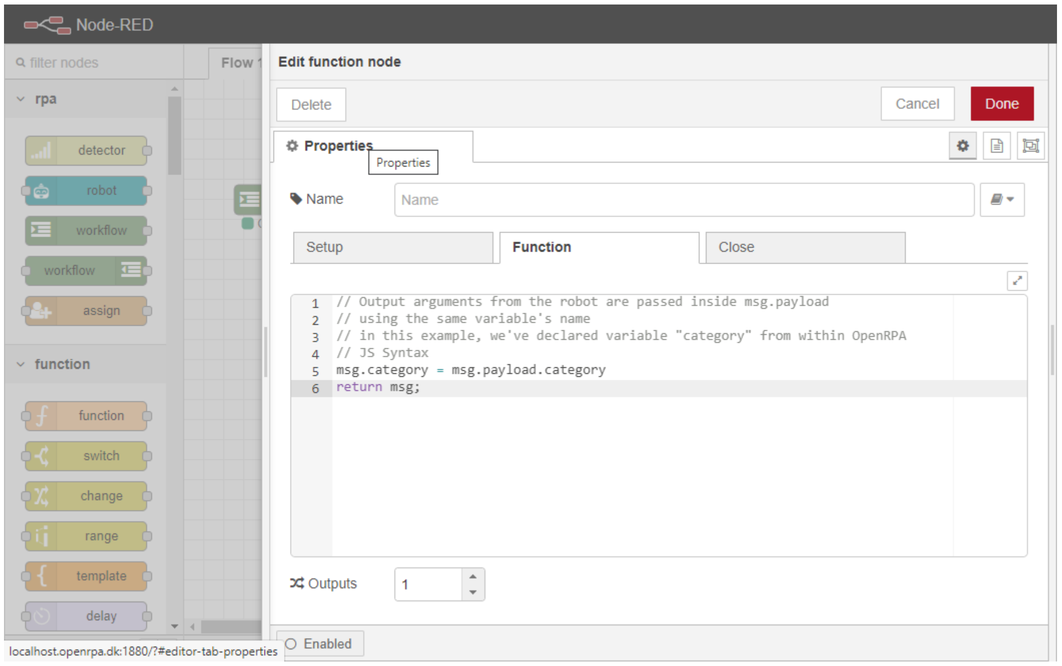


In our example, the parameters needed are `msg.category` - which will be gathered as `msg.payload.category` from the `category` argument passed by the OpenRPA workflow.



In the `Function` node's properties, set `msg.category = msg.payload.category`.

This is used so we can set the `category` parameter to the `HTTP Request` node, captured from the OpenRPA workflow which the user will build next.



```
// Output arguments from the robot are passed inside msg.payload
// using the same variable's name
// in this example, we've declared variable "category" from within OpenRPA
// JS Syntax
msg.category = msg.payload.category
return msg;
```

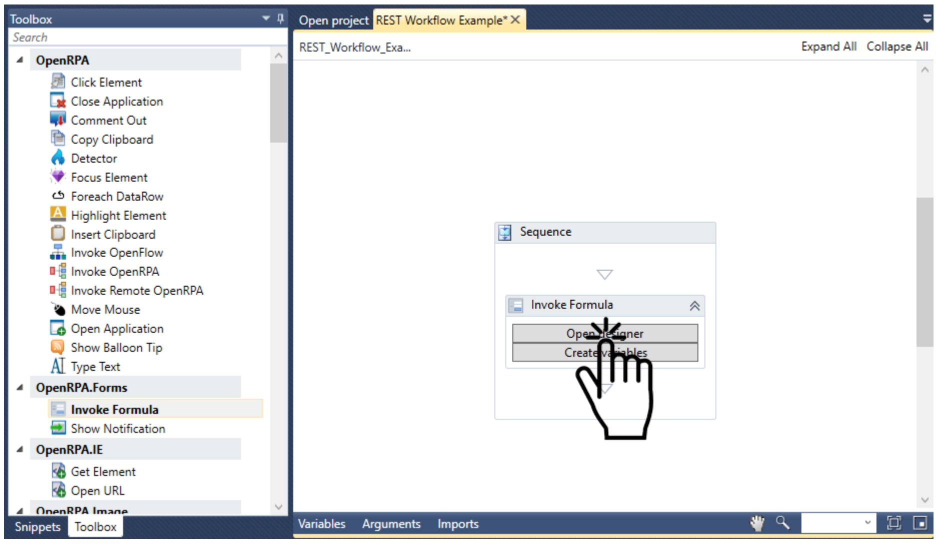
Note: In this example, the user could use the `Change` node to set the `msg.category` attribute to `msg.payload.category` as well, or simply pass `msg.payload.category` directly to the HTTP Request, which will be discussed further on.

OpenRPA Forms

Now the user creates a dynamic WPF form to gather the `category` value from the client.

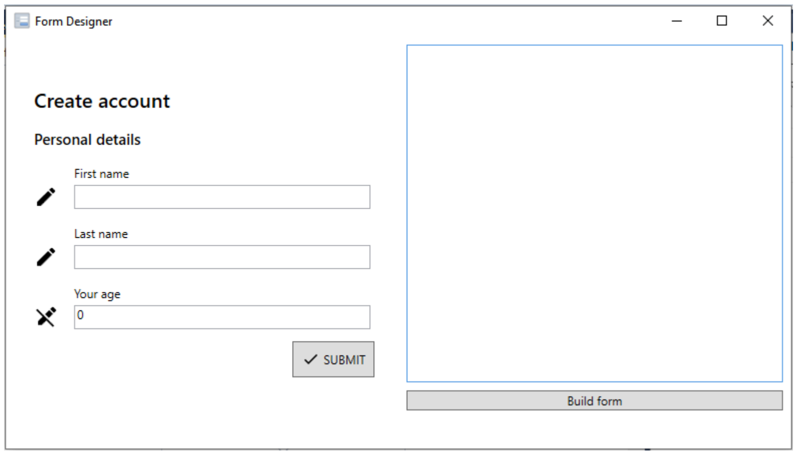
First drag a `Invoke Formula` snippet to the main sequence.

Now click `Open designer` to open the `Form Designer`.



Note: The `Form Designer` is the place where the user can create his desired WPF forms to gather input from the client. At the right-side of it, a preview of the input form is shown. At the left-side of it, the user enters the code used to design the input form. After adding code to the `Form Designer`, always press the **Build form** button to save changes and visualize the preview.

Clear all example code and add the code snippet below to the designer code editor. Then press the **Build form** button to update the form.

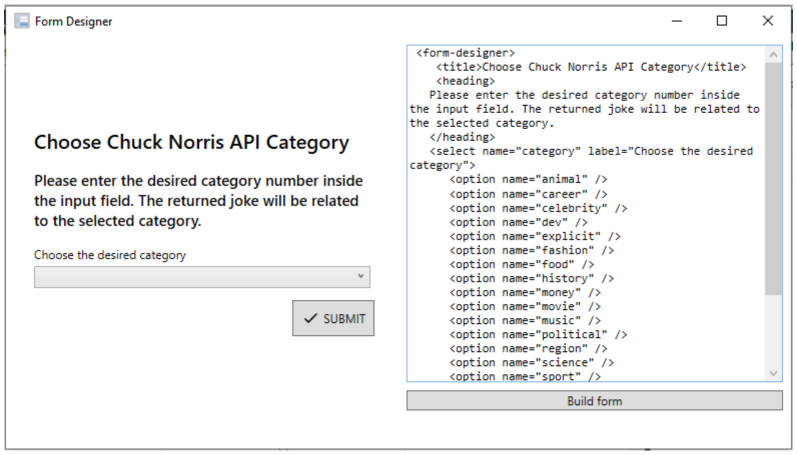


```
<form-designer>
 <title>Choose Chuck Norris API Category</title>
```

(continues on next page)

(continued from previous page)

```
 <heading>
Please enter the desired category number inside the input field. The returned joke
↪will be related to the selected category.
</heading>
<select name="category" label="Choose the desired category">
   <option name="animal" />
   <option name="career" />
   <option name="celebrity" />
   <option name="dev" />
   <option name="explicit" />
   <option name="fashion" />
   <option name="food" />
   <option name="history" />
   <option name="money" />
   <option name="movie" />
   <option name="music" />
   <option name="political" />
   <option name="region" />
   <option name="science" />
   <option name="sport" />
   <option name="travel" />
</select>
<action name="submit" content='SUBMIT' icon='check' validates='true' ClosesDialog=
↪'true' ></action>
</form-designer>
```



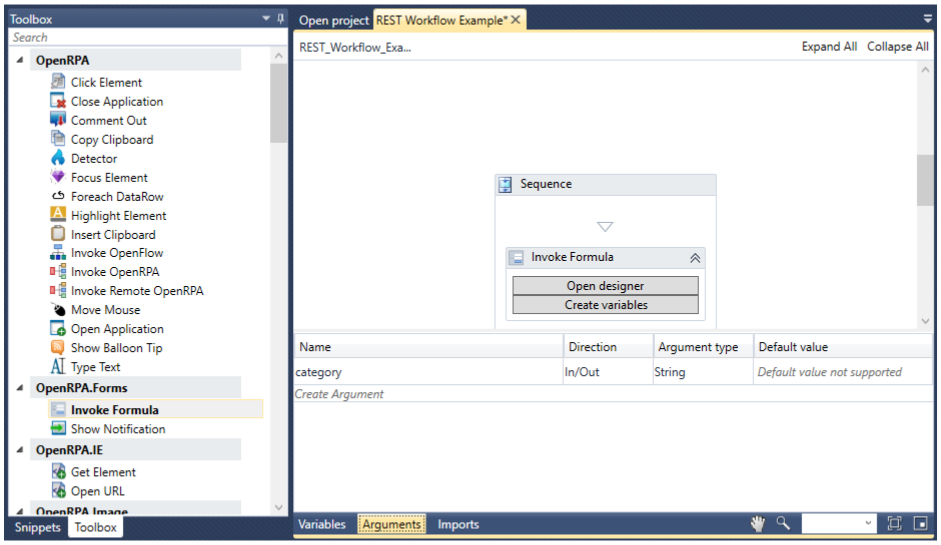
Choose Chuck Norris API Category

Please enter the desired category number inside the input field. The returned joke will be related to the selected category.

```
<form-designer>
   <title>Choose Chuck Norris API Category</title>
   <heading>
  Please enter the desired category number inside
the input field. The returned joke will be related to
the selected category.
  </heading>
  <select name="category" label="Choose the desired
category">
     <option name="animal" />
     <option name="career" />
     <option name="celebrity" />
     <option name="dev" />
     <option name="explicit" />
     <option name="fashion" />
     <option name="food" />
     <option name="history" />
     <option name="money" />
     <option name="movie" />
     <option name="music" />
     <option name="political" />
     <option name="region" />
     <option name="science" />
     <option name="sport" />
```

Note: After adding the code snippet, all the variables used in the WPF form can be automatically created by pressing the **Create variables** button.

In our case, the workflow is unable to use a `Variable`. Since the value will be consumed by Node-RED, it must obligatorily be set as an `Argument` with **Direction** set as `In/Out` or `Out` and type `String`.



Name	Direction	Argument type	Default value
category	In/Out	String	Default value not supported

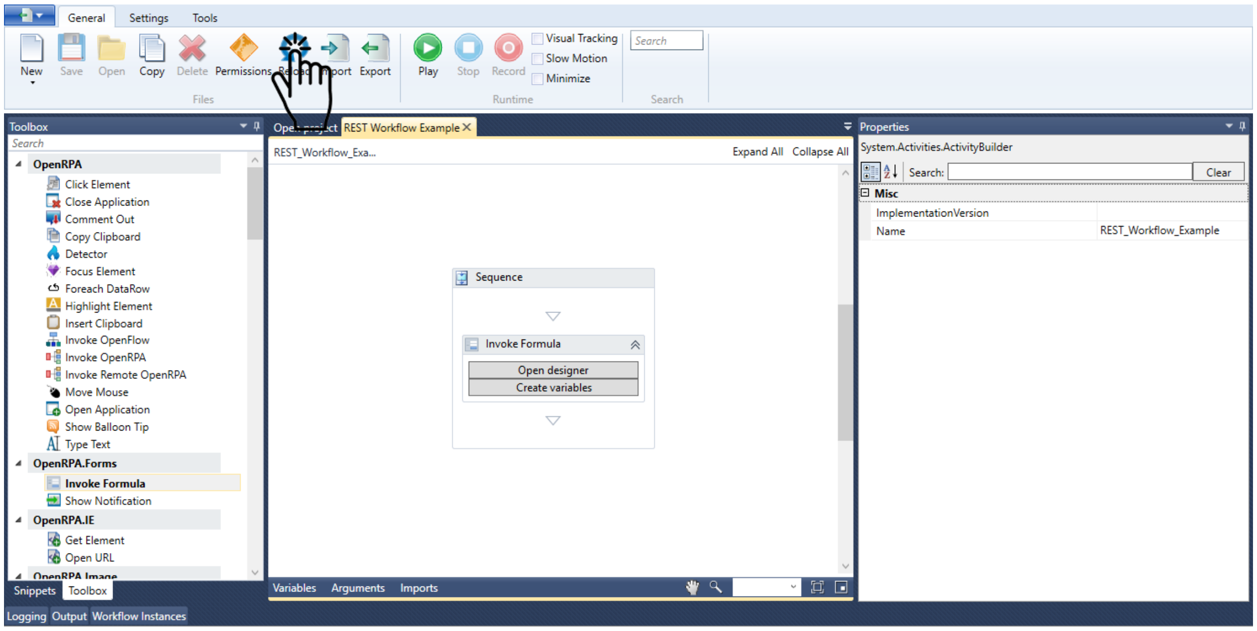
Now, to guarantee that the value gathered from the WPF form has the correct type, the user will cast it to a `String` using the **Assign** activity.

Invoke Workflow through OpenRPA

Now its time to finally invoke the workflow through OpenRPA and devise the processing logic.

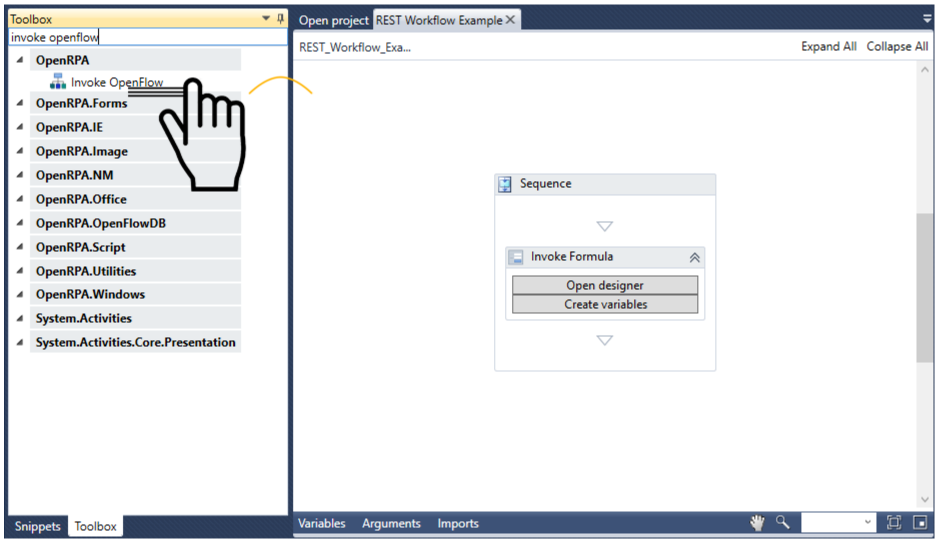
Since the flow is already deployed, it is needed that the user synchronize the workflows with the OpenFlow repository.

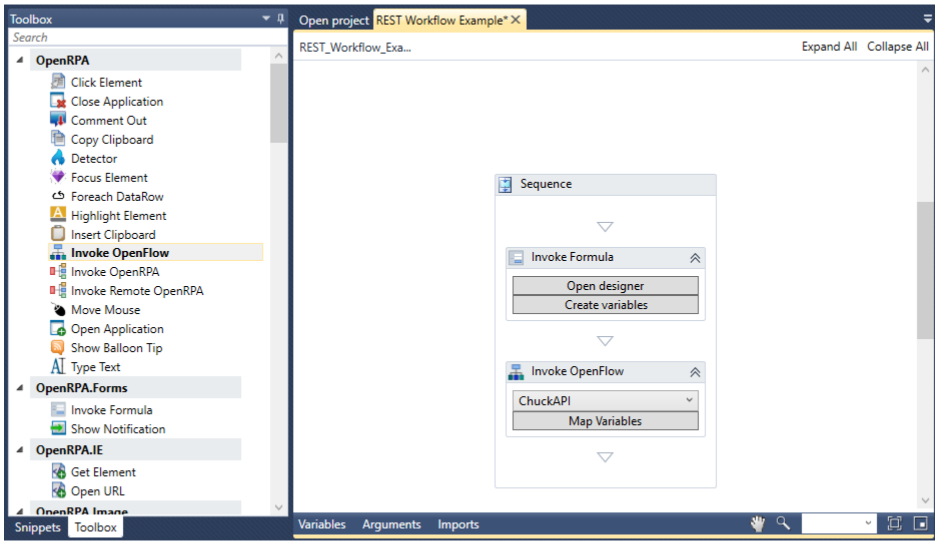
To do the aforementioned task, click the **Reload** button inside OpenRPA's ribbon. This will refresh all workflows contained inside the OpenFlow repository to OpenRPA.



Drag an **Invoke OpenFlow** activity to the main sequence inside OpenRPA's Designer.

Select the **RPA Workflow** which you want to invoke. In our case `ChuckAPI`.



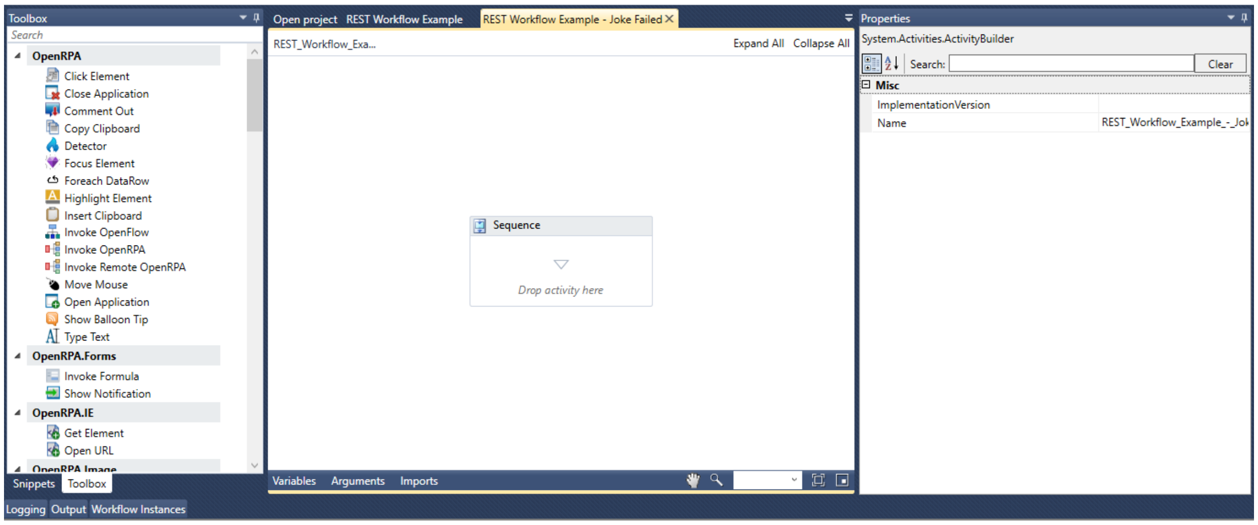


Note: Note: When Node-RED finishes executing the flow, whatever is set in the msg.payload will return back to the robot. Variables are automatically mapped between the two applications: if msg.payload.person exists in NodeRED, whatever returns in there will automatically go in the variable named "person" inside OpenRPA.

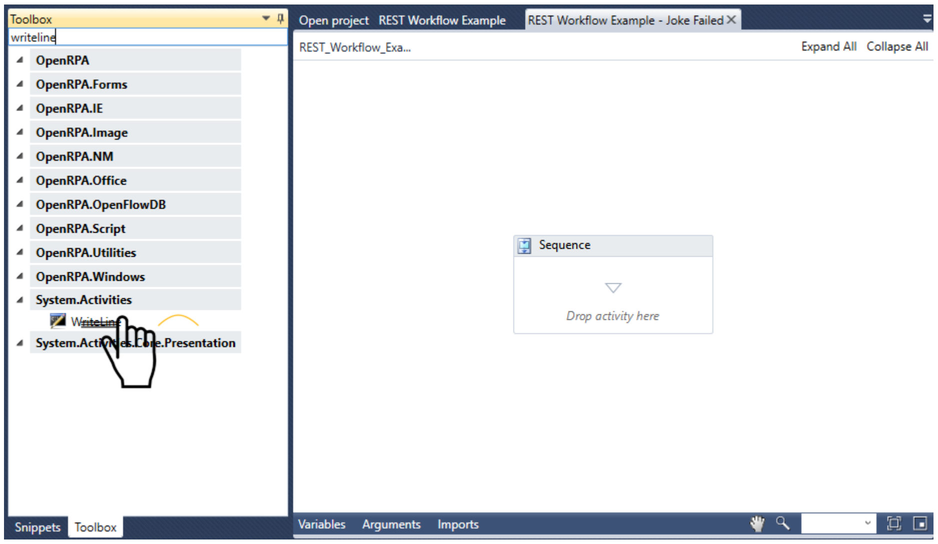
Create Joke Failed Workflow

Now that the ChuckAPI process is set, the user can proceed to set a workflow which will be executed in case any failures occur upon execution of the usual process.

Create a new workflow, in our case the name `REST Workflow Example - Joke Failed` is used to specify the workflow.

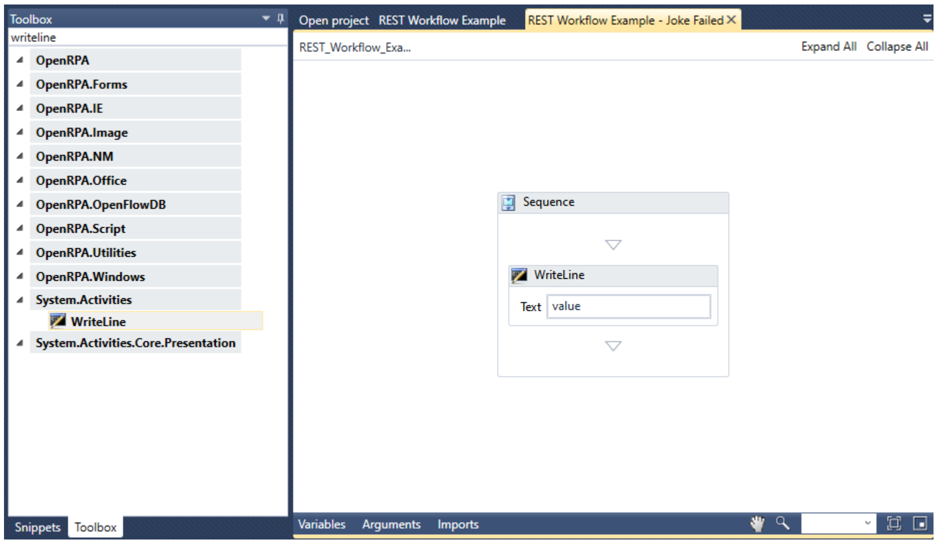


Drag a **WriteLine** activity to the main sequence.



Now it is needed to create the default variable - inside the **Variables & Arguments box** - from which the entry containing the `Chuck Norris API` joke content is returned. This variable alias is `value`, as it can be seen in Chuck Norris API Random Joke (`https://api.chucknorris.io/jokes/random`).

Set the output value, in the `Text` input field or through the **Properties box** to a new variable of alias `value`.



Note: Press `Ctrl+K` to automatically create the variable when entering it upon an input field or create it manually by clicking the **Variables box** under the bottom part of the screen. Use the right-hand side to enter the value which will be assigned to the variable.

This workflow is merely used as an example of the almost endless possibilities within OpenRPA. The user could read the `ChuckAPI` joke into a `.csv` or `.xslx` file, invoke one or many more workflows to deal with the data, pass it into `MongoDB Entities` - as stated before, there are almost no limits to what the user can accomplish in terms of automation.

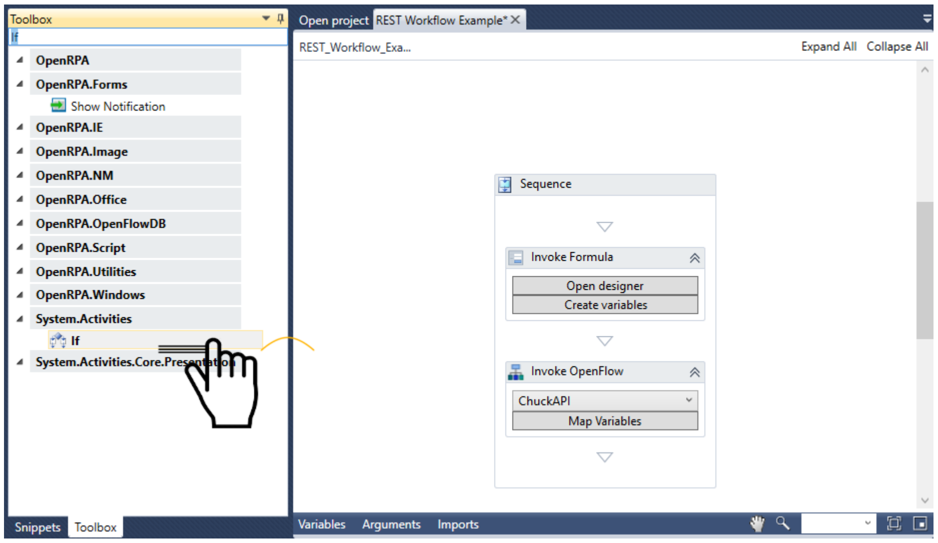
Evaluate Chuck Norris Joke

Here we set the evaluation process for checkign whether the joke captured from the `Chuck Norris API` is a joke or not.

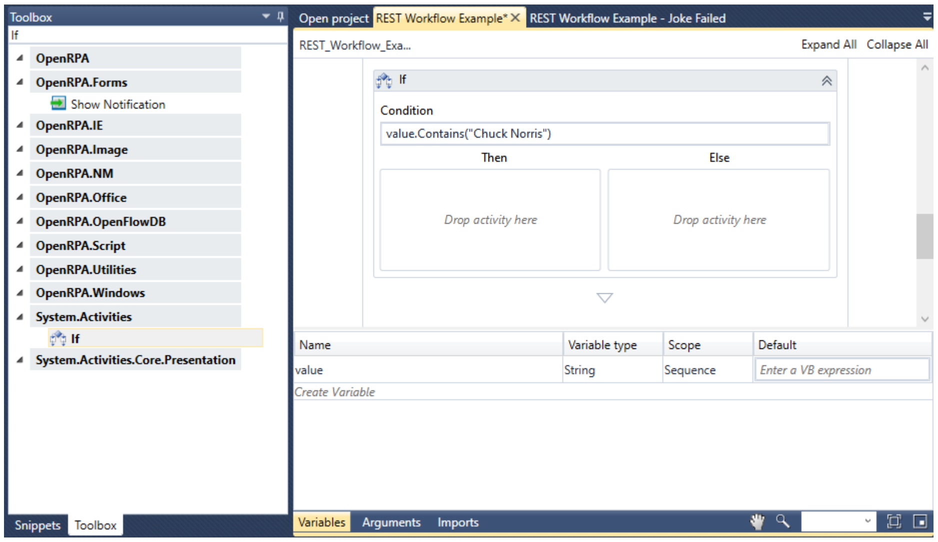
The validation standard used here is to check whether the content of the `value` entry in the `JSON Dictionary` returned contains the string "Chuck Norris" or not.

First the user needs to create the default variable - inside the **Variables & Arguments box** - from which the entry containing the `Chuck Norris API` joke content is returned. This variable alias is `value`, as it can be seen in Chuck Norris API Random Joke (`https://api.chucknorris.io/jokes/random`).

In `REST Workflow Example` workflow, drag an **If** activity to the Designer.



Enter `value.Contains("Chuck Norris")` inside the `Condition` input field, which is shown in the activity itself or inside the **Properties box.**



value.Contains("Chuck Norris")

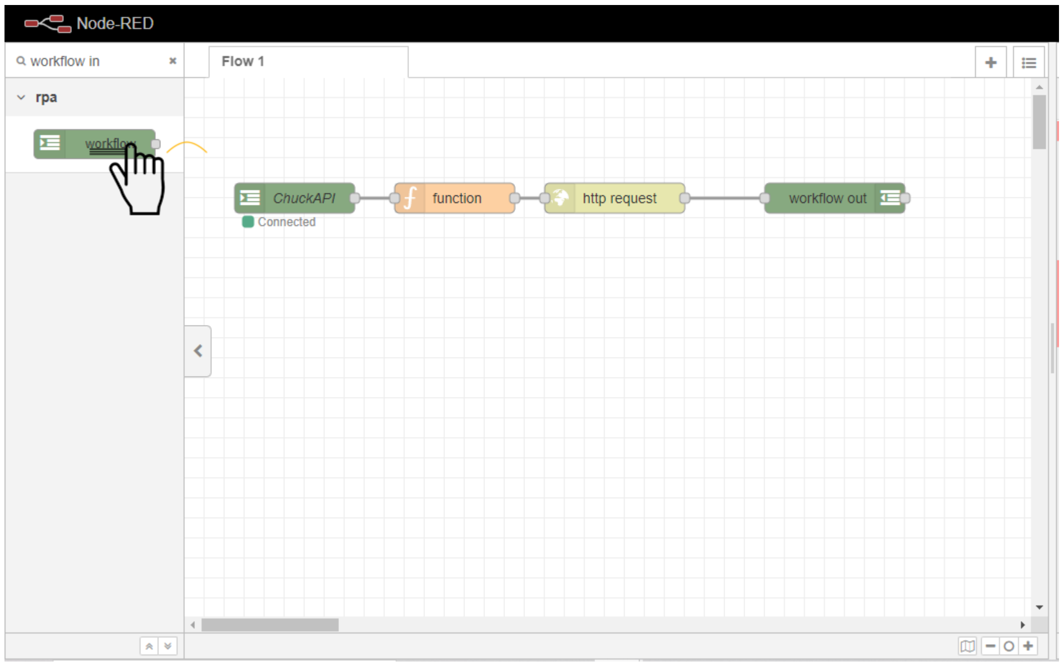
Name	Variable type	Scope	Default
value	String	Sequence	Enter a VB expression

Setting Evaluation Failed process in Node-RED

The user must now set the proper flow which will be further executed in case of failure of evaluation of the `ChuckAPI` process.

Workflow Logic

Drag a `Workflow In` node to the workspace.



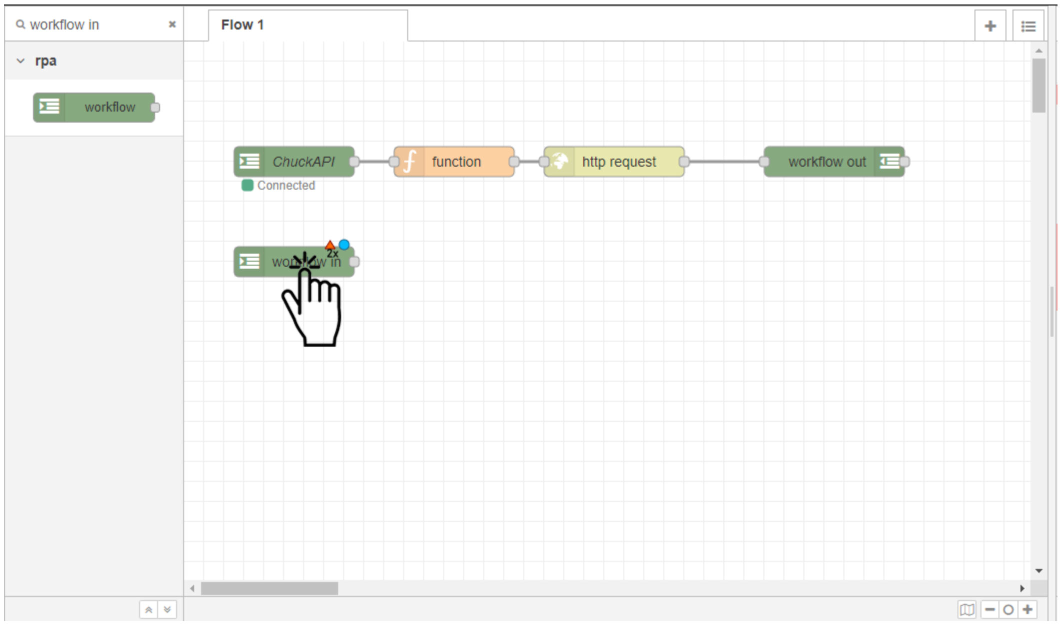
Set the `Queue name` to define the queue which the OpenFlow workflow will be assigned to in `Workflow In`'s properties.

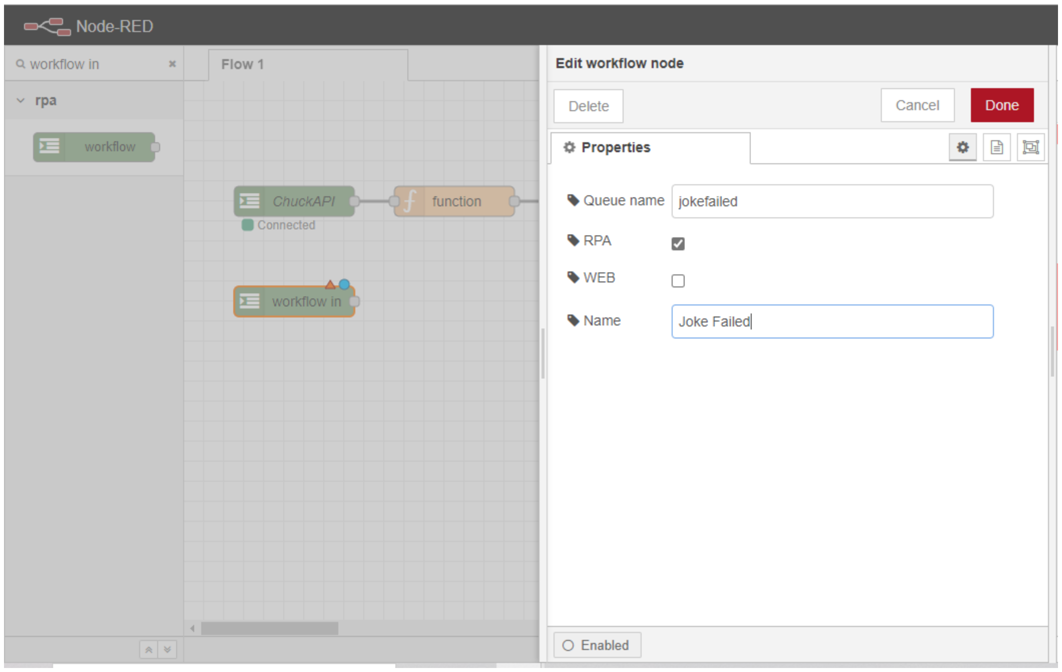
The queue name is completely arbitrary, the user can choose for any name he wishes to. In our case, it is set as `jokefailed`.

Note: To access any node's properties inside the Node-RED workspace, simply double click the given node. The **Properties box** for the node, containing each input fields required by itself will then appear.

Check the `RPA` checkbox to allow this flow's workflow to be invoked from OpenRPA agents.

Set the name for the workflow in the `Name` input field. Again, it is named arbitrarily. In our case, it is set as `Joke Failed`.





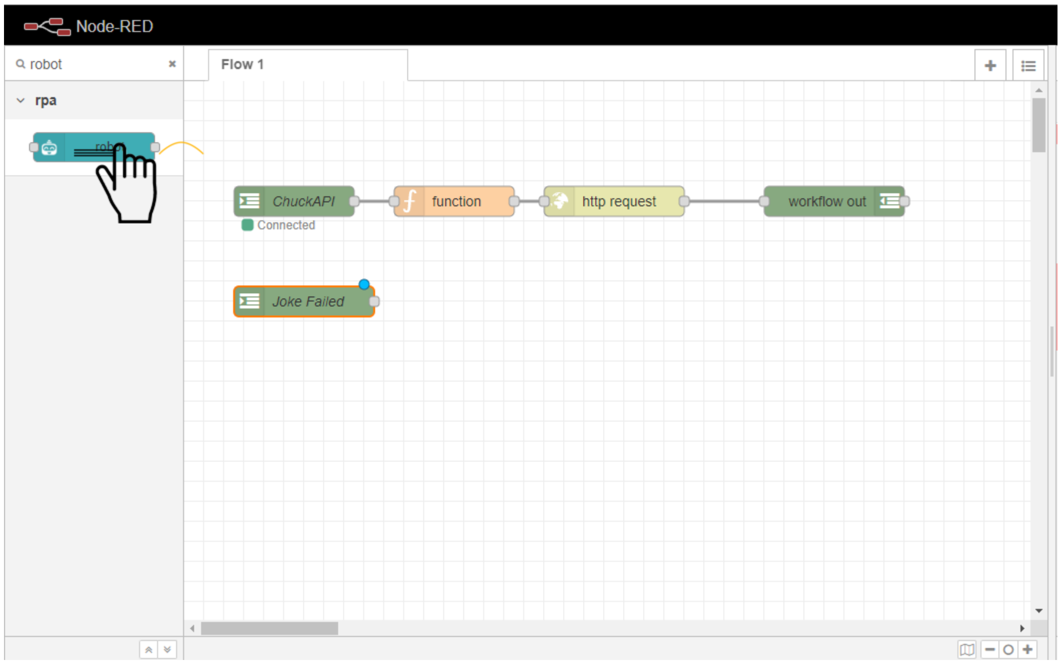
Queue name: jokefailed

RPA: ☑

WEB: ☐

Name: Joke Failed

Drag a `RPA Workflow` or `Robot` node to the workspace. This node enables us to invoke the `REST Workflow - Joke Failed` workflow upon failure on evaluation.

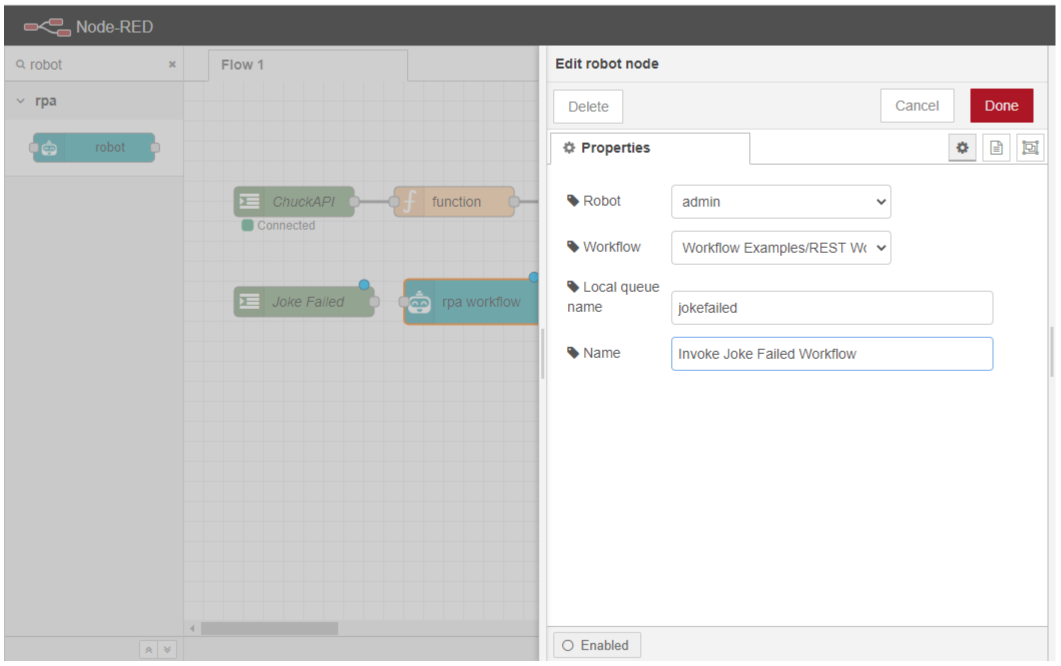


Set the `Robot` which will invoke the workflow and the `Workflow` which will be invoked, inside `robot` node's properties. As well as `Local queue name` and `Name` for the node.

`Local queue name` is arbitrary, it is used to set the queue which will be used for queueing the robot instances.

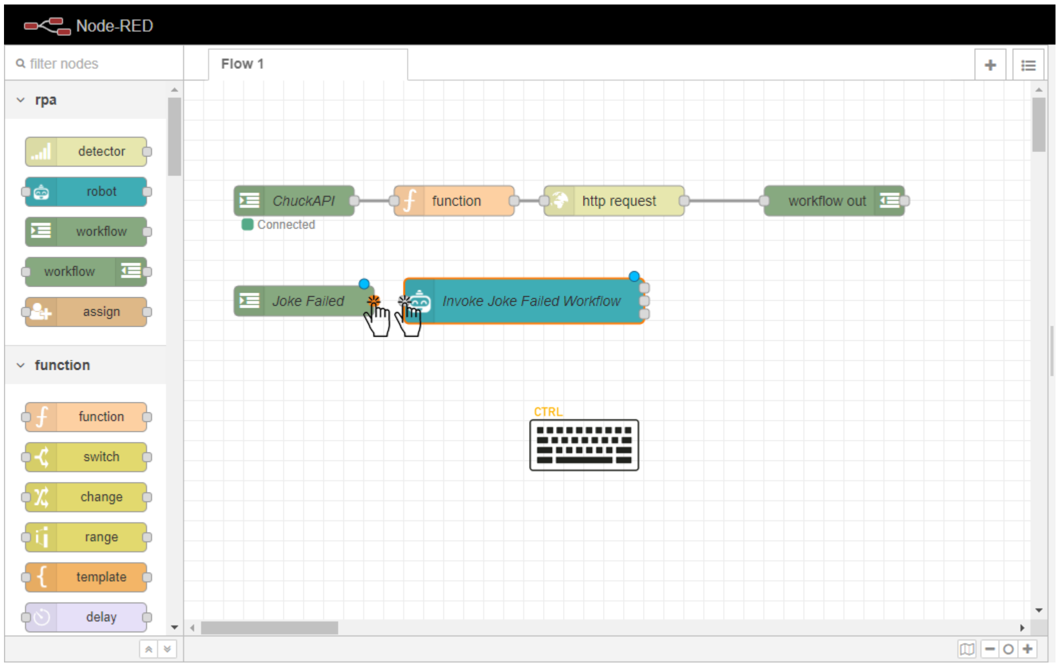
`Name` parameter is used only for displaying purposes inside Node-RED.

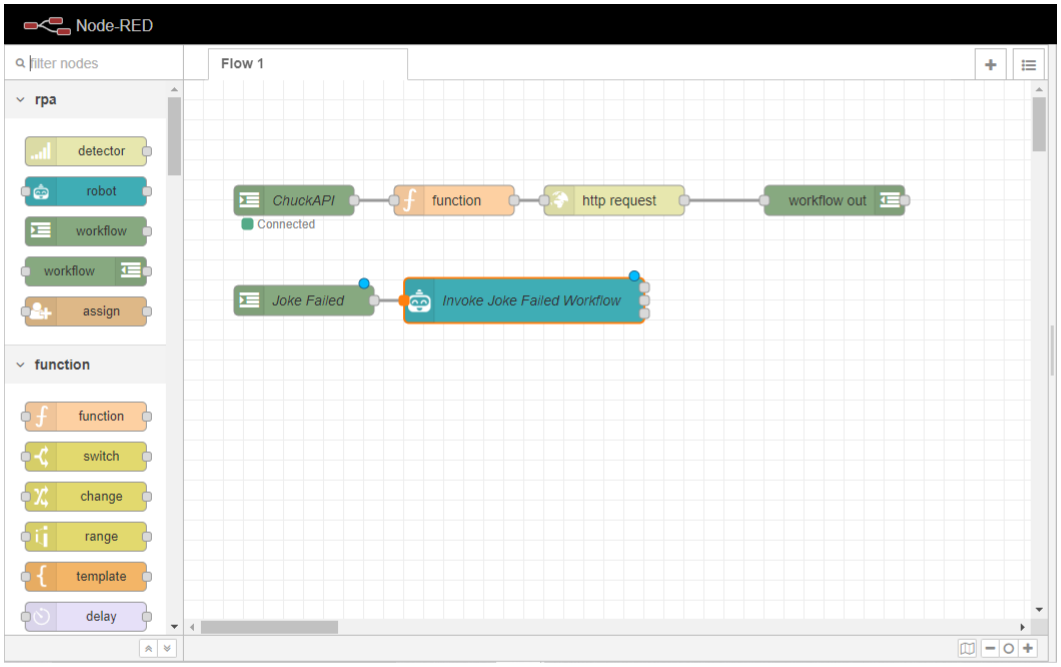
In our case, `Local queue name` is set as `jokefailed` and `Name` is set as `Invoke Joke Failed Workflow`.



Robot: admin
Workflow: Workflow Examples/REST W
Local queue name: jokefailed
Name: Invoke Joke Failed Workflow

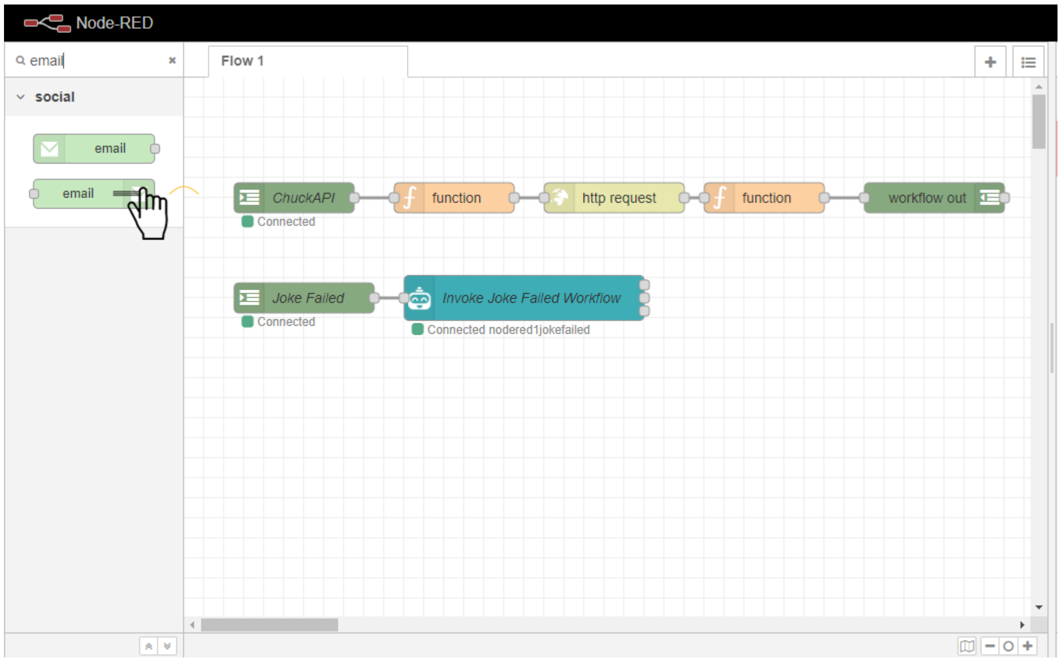
Connect the `Workflow In` node to the `Robot` node.





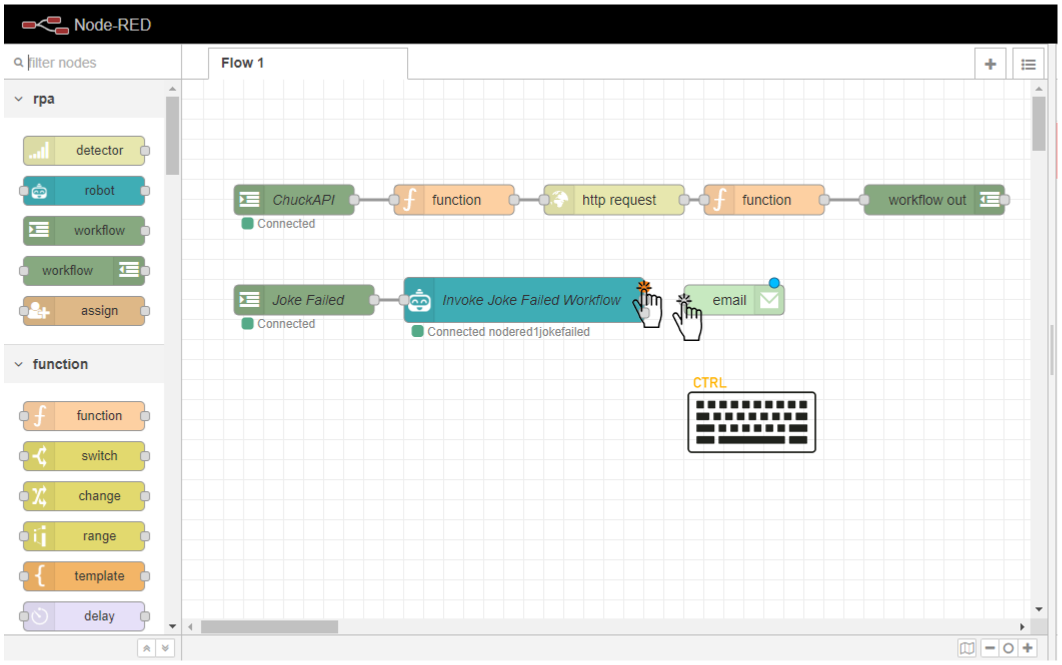
Note: To easily connect a node simply press the `Ctrl` key while clicking in the grey rectangles - called ports - in the outer part of each node. The user can also click and hold to connect the nodes but the author finds it more feasible simply using the `Ctrl` key.

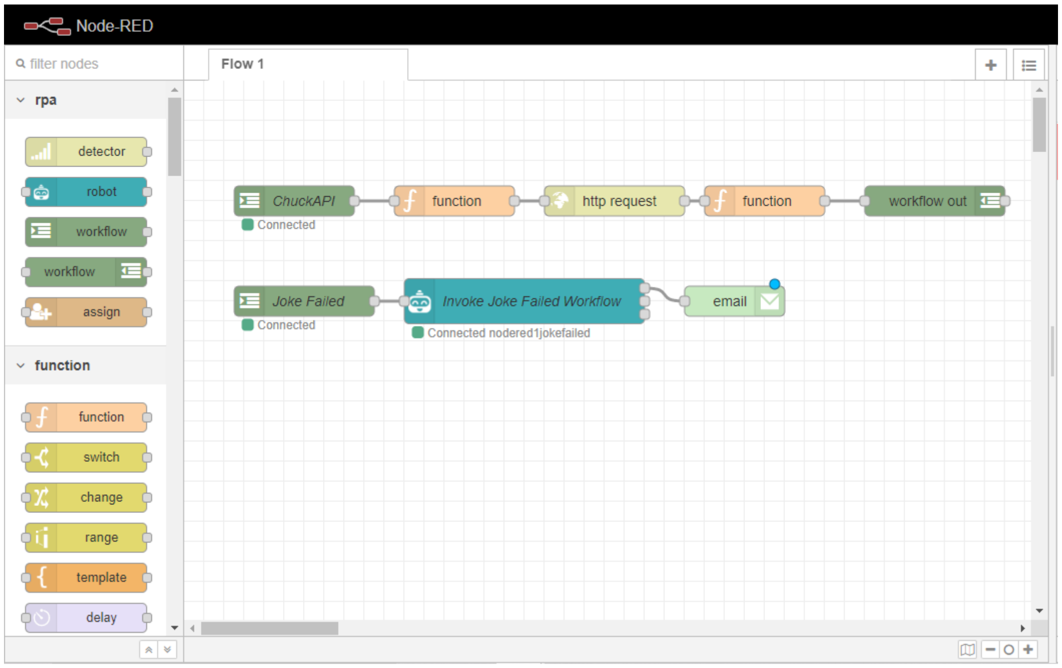
Drag an `Email Out` node - the one with the letter icon to the right-most side of the activity - to the workspace. This node will be responsible to sending the `msg.payload` object JSON stringified to the selected email when an error occurs upon evaluation.



Connect the top-most port in the `Invoke Joke Failed Workflow` node to the port outside the `email` node.

Note: To easily connect a node simply press the `Ctrl` key while clicking in the grey rectangles - called ports - in the outer part of each node. The user can also click and hold to connect the nodes but the author finds it more feasible simply using the `Ctrl` key.





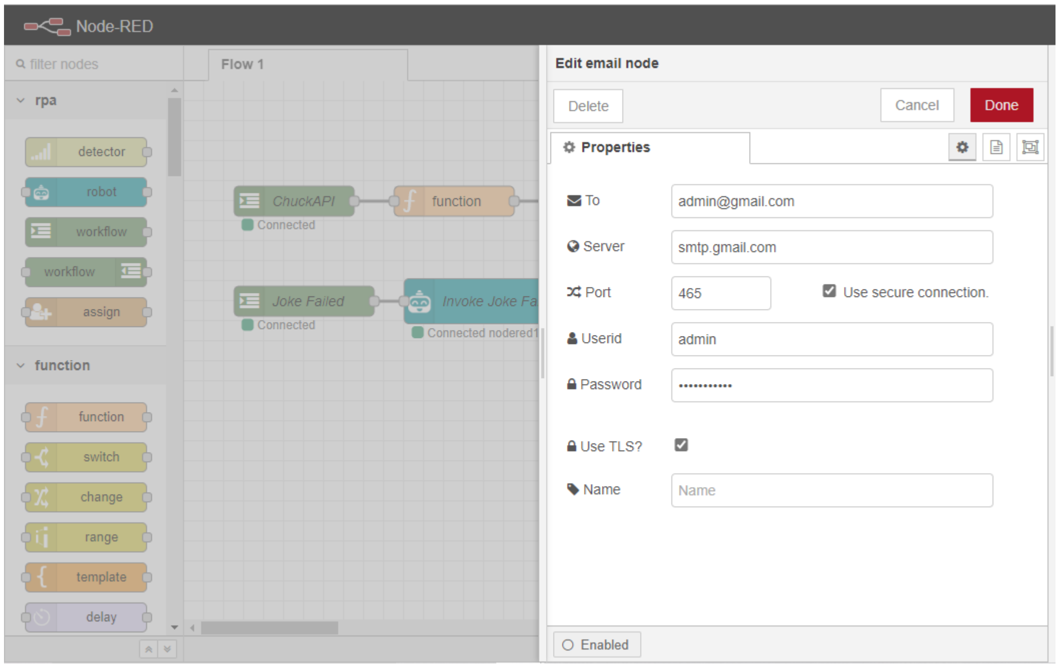
Set the properties for the `email` node.

`To` refers to the the e-mail address from the receiver.

`Userid` is the UserId from the account which will send the e-mail. (ie. e-mail without domain - if the e-mail is `admin@openrpa.dk`, then your UserId is `admin`)

`Password` - the password for the email account from the sender.

In our case, a dummy e-mail is set - as `admin@gmail.com`, which will be used to send and receive the message. It is noted here that the user must change the e-mail or else the workflow obviously won't work.



To: admin@gmail.com

Server: smtp.gmail.com

Port: 465 ☑ Use secure connection.

Userid: admin

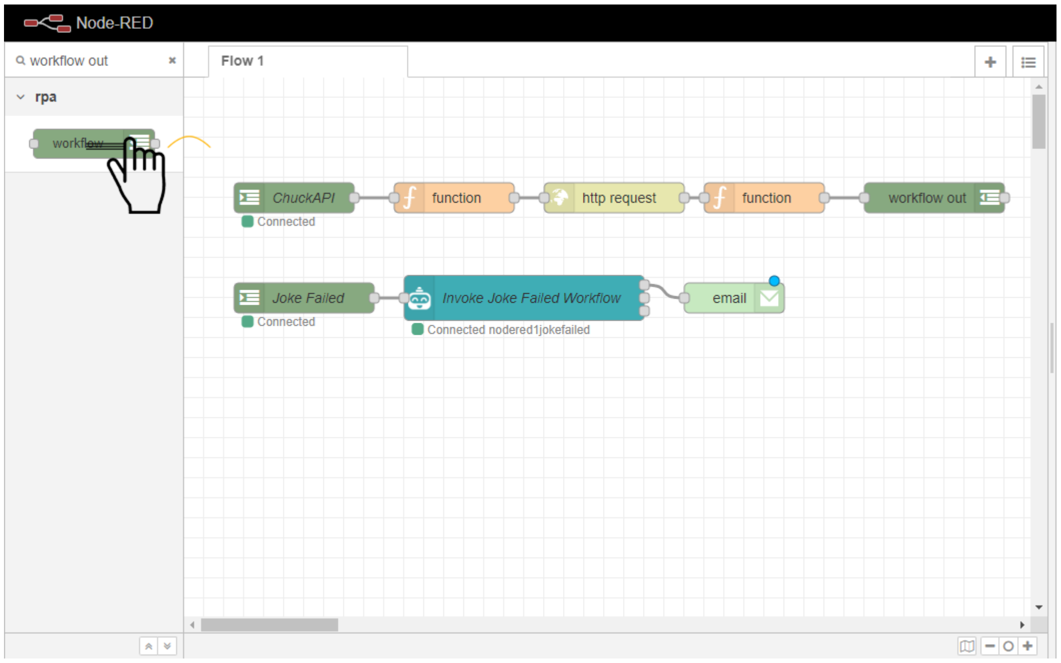
Password: •••••••••••

Use TLS? ☑

Name: Name

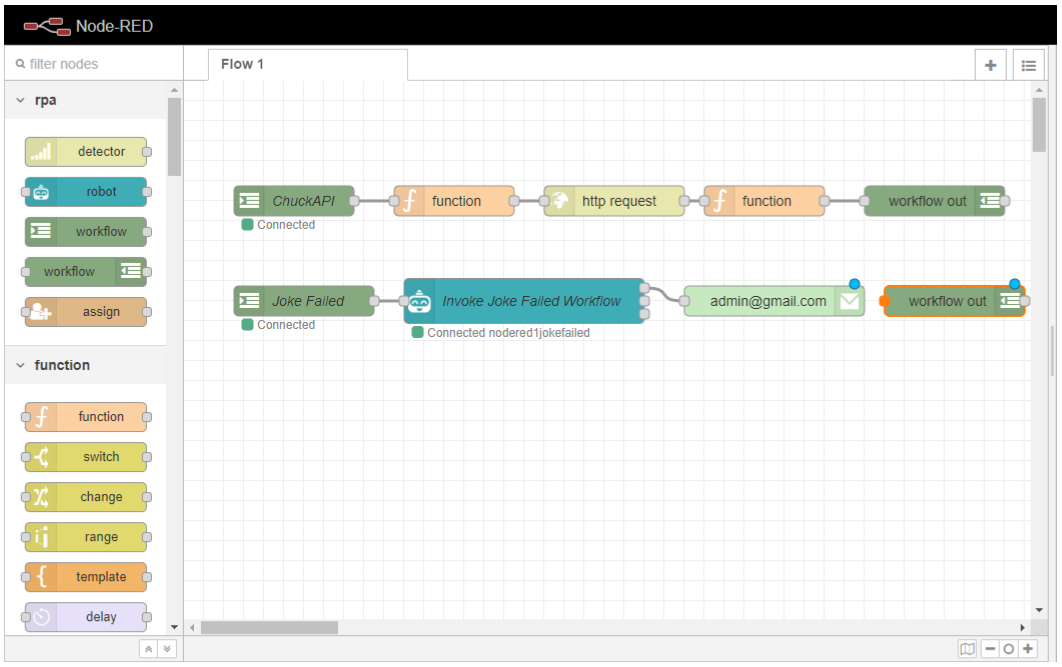
Note: If the user is using Gmail's SMTP Server, it is needed that the settings for allowing **Less secure app access** are correct. Else, the user will receive an error message upon execution of the workflow. To do that, refer to Less secure app access (`https://myaccount.google.com/lesssecureapps`) and make sure that the `Allow less secure apps` settings is set to `ON`.

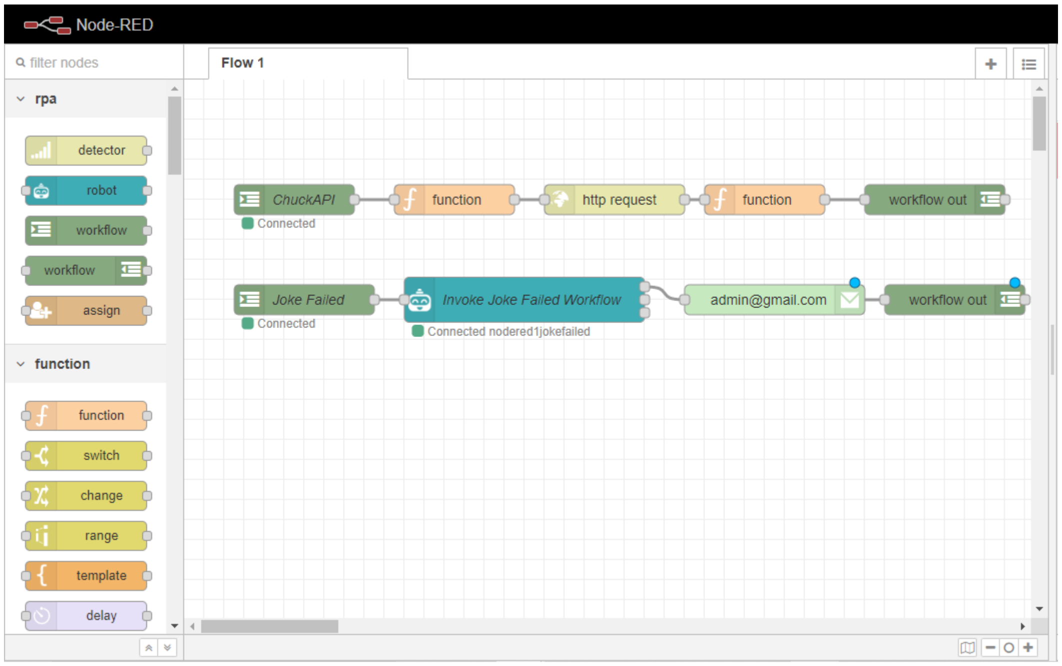
Drag a `Workflow Out` node to the workspace.



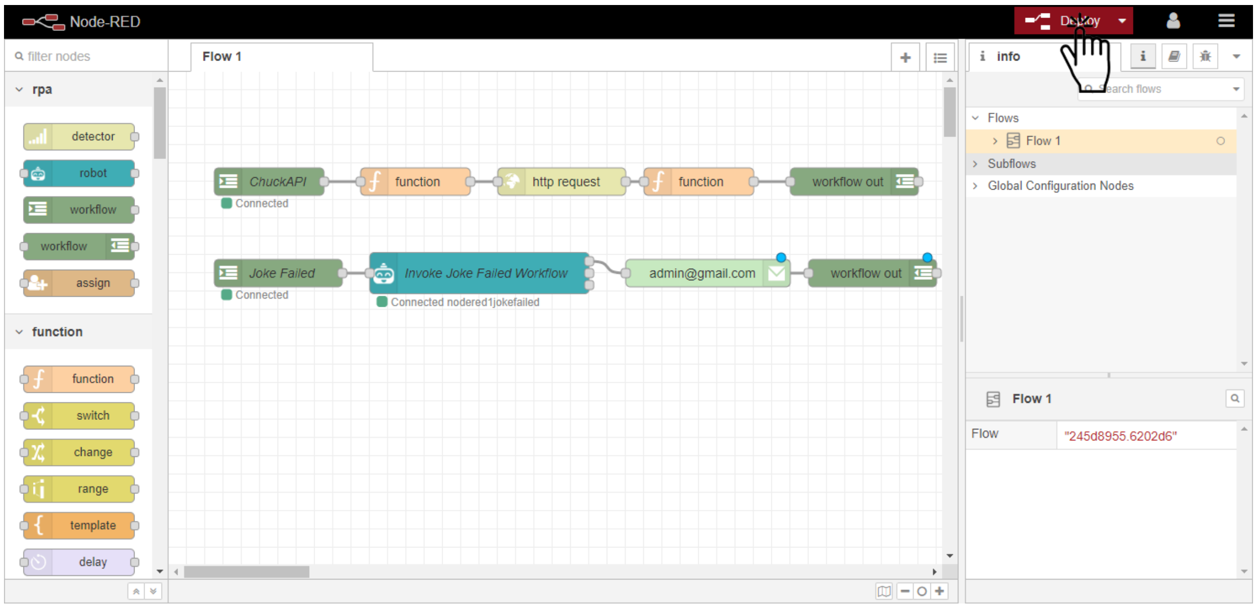
Connect the `Workflow Out` node to the `email` - now `admin@gmail.com` - node.

Note: To easily connect a node simply press the `Ctrl` key while clicking in the grey rectangles - called ports - in the outer part of each node. The user can also click and hold to connect the nodes but the author finds it more feasible simply using the `Ctrl` key.





Deploy your flow by clicking the red Deploy button at the top-left side of the screen.

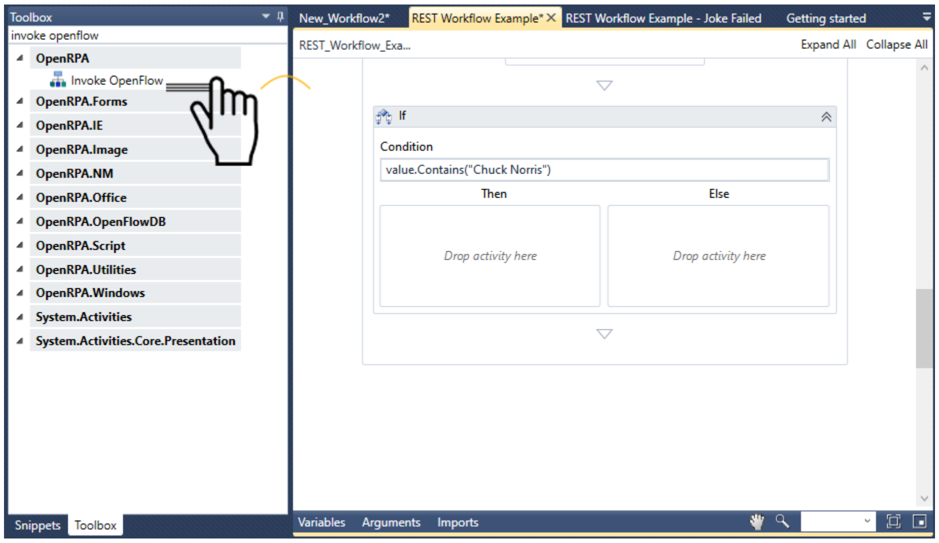


Note: Upon any change inside a Node-RED flow, the user must deploy the given flow so the changes are propagated throughout robots currently using that given flow.

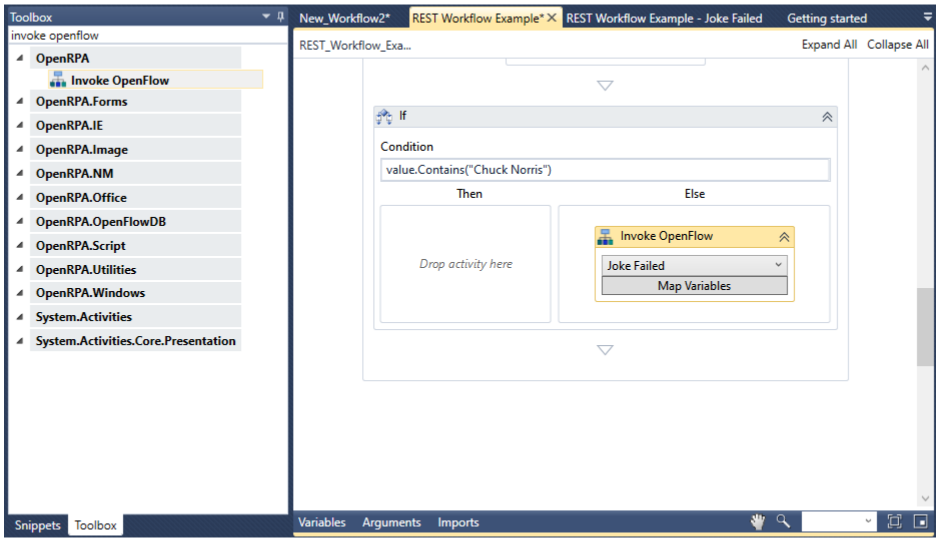
Invoke Joke Failed Workflow

Now it is needed for the workflow to invoke the `REST Workflow Example - Joke Failed` in case the `ChuckAPI` process is not properly evaluated.

To do that, drag an `Invoke OpenFlow` activity to the `Else` sequence, inside the **If** activity.



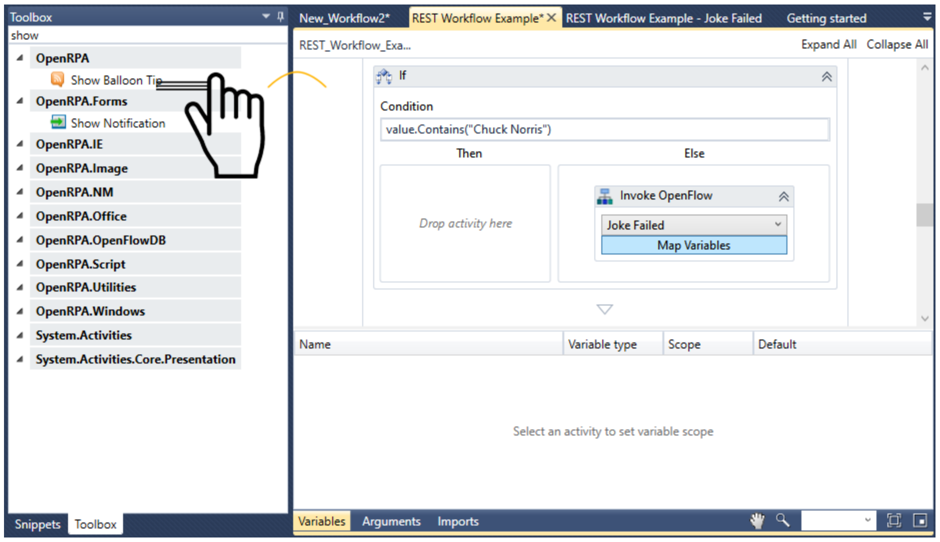
Now select `Joke Failed` as the workflow to be invoked.



This way the `REST Workflow Example - Joke Failed` activity will be automatically invoked upon any failure in the evaluation process.

Output logic

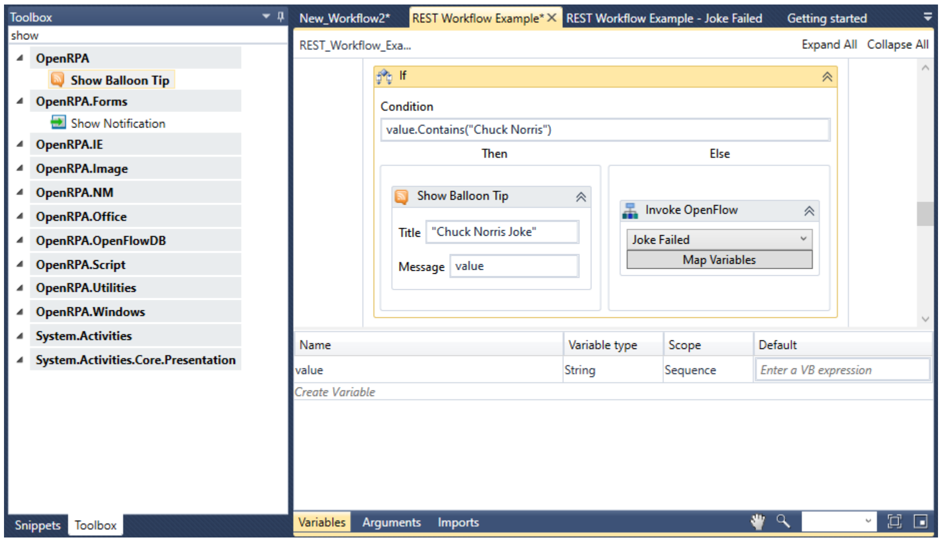
Drag a **Show Balloon Tip** activity to the `Then` sequence, inside the **If** activity.



Insert the `Title` for the **Balloon Tip** which will appear.

Insert the `Message` which will appear to the user. In our case, it is the variable `value`.

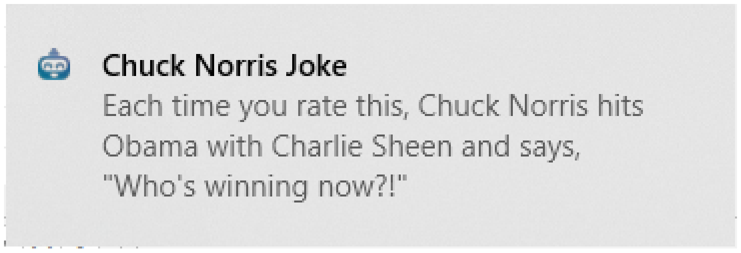
Note: Press `Ctrl+K` to automatically create the variable when entering it upon an input field or create it manually by clicking the **Variables box** under the bottom part of the screen. Use the right-hand side to enter the value which will be assigned to the variable.

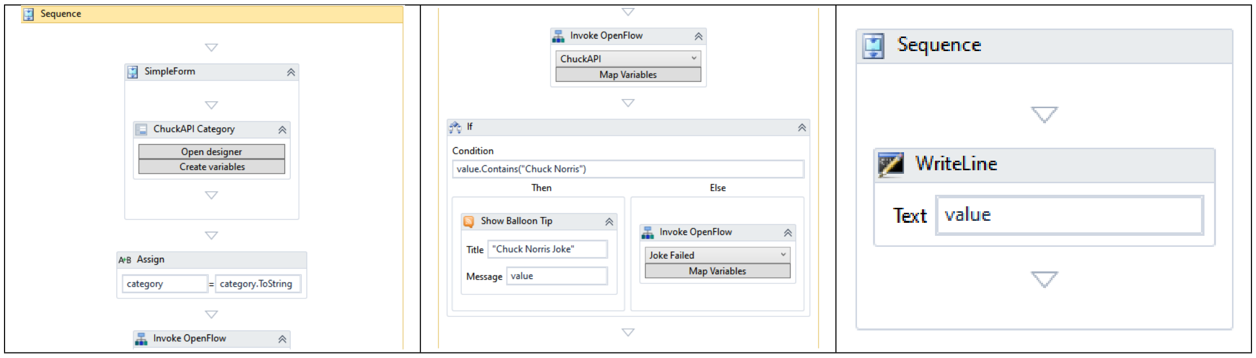


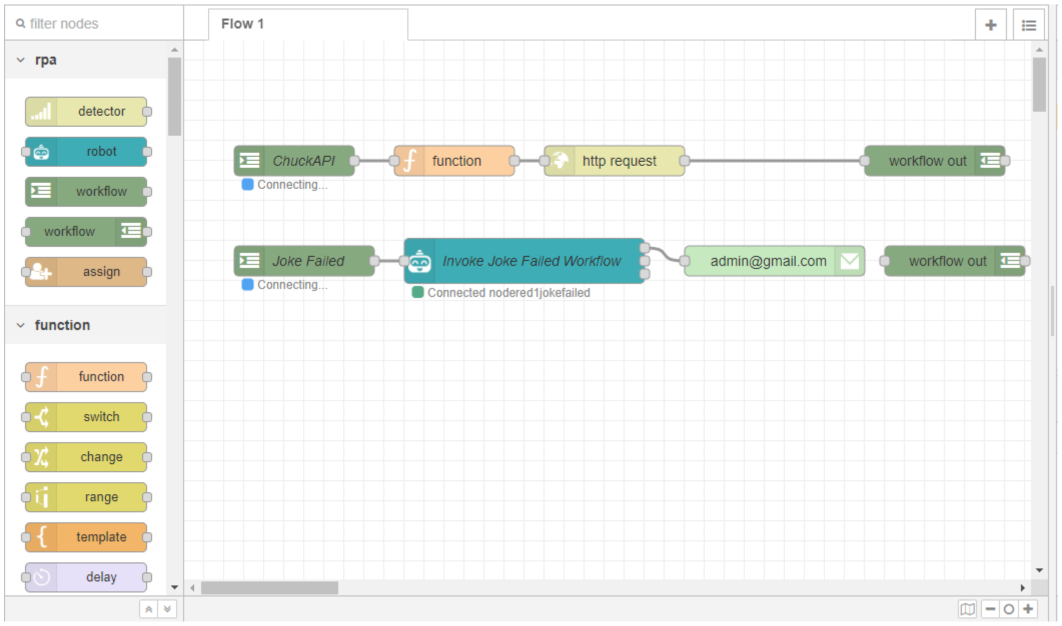
Name	Variable type	Scope	Default
value	String	Sequence	Enter a VB expression
Create Variable			

Workflow Finished

A snippet is shown below containing both the output and the end of the workflow. As well as the whole workflow there after.


Chuck Norris Joke
Each time you rate this, Chuck Norris hits Obama with Charlie Sheen and says, "Who's winning now?!"





3.11.9 Reading and Parsing PDF Files

Coming soon - Work in progress

3.11.10 Using DataTables

Coming soon - Work in progress

Chapter 4

Node-RED

4.1 What is Node-RED?

4.1.1 What is Node-RED?

Node-RED is a programming tool for wiring together our tools (OpenRPA, OpenFlow) as well as hardware devices, APIs and even online services in new and interesting ways. Think of it as a "backend" process flow designer & integrator.

Node-RED provides an in-browser editor where you can connect flows using nodes available at its palette. Each node represents a step that wired together form a meaningful task. It also follows a common pattern: input, processing and output. It is important to note here that Node-RED functions similarly as a middleware to an information processing system. It simply connects the inputs to the workflows and allows them to process it.

4.2 Accessing for the first time

To access NodeRED, access your environment URL (default at http://localhost.openrpa.dk:1880).

Once you access the URL, a button is shown with the text `Sign in with SAML`. When you click the button, Node-RED automatically gathers your OpenFlow's authentication data and logs in to Node-RED with the user currently logged in on OpenFlow.

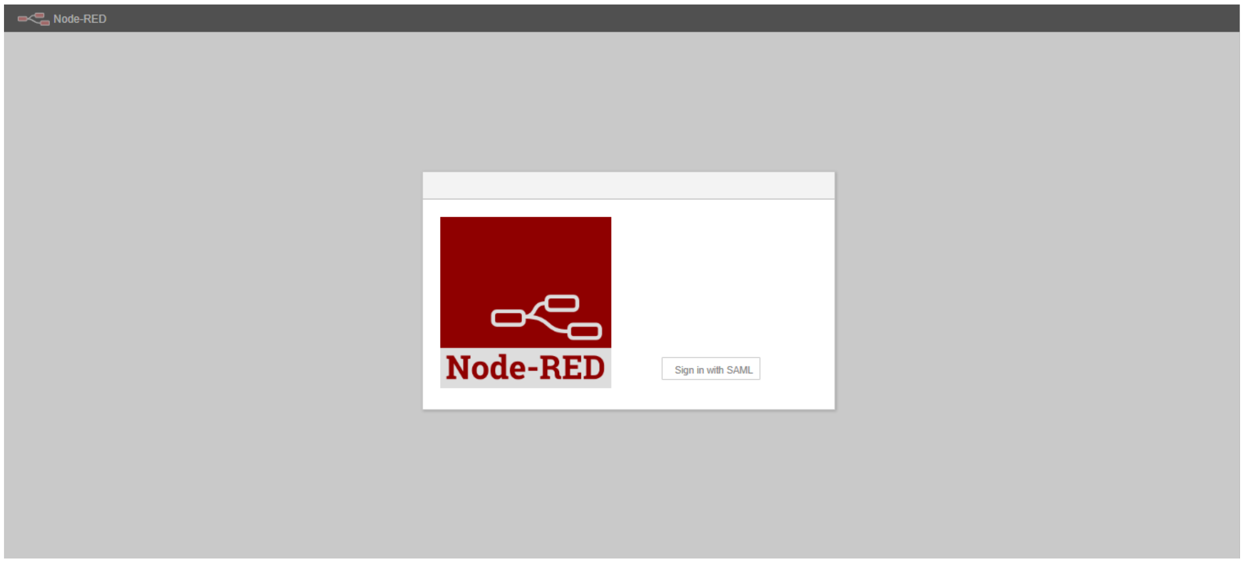


Fig. 1: Node-RED Sign In page.

4.3 Node-RED Editor

The editor window is where all the work will be done. It contains four components: header, palette, workspace and sidebar.

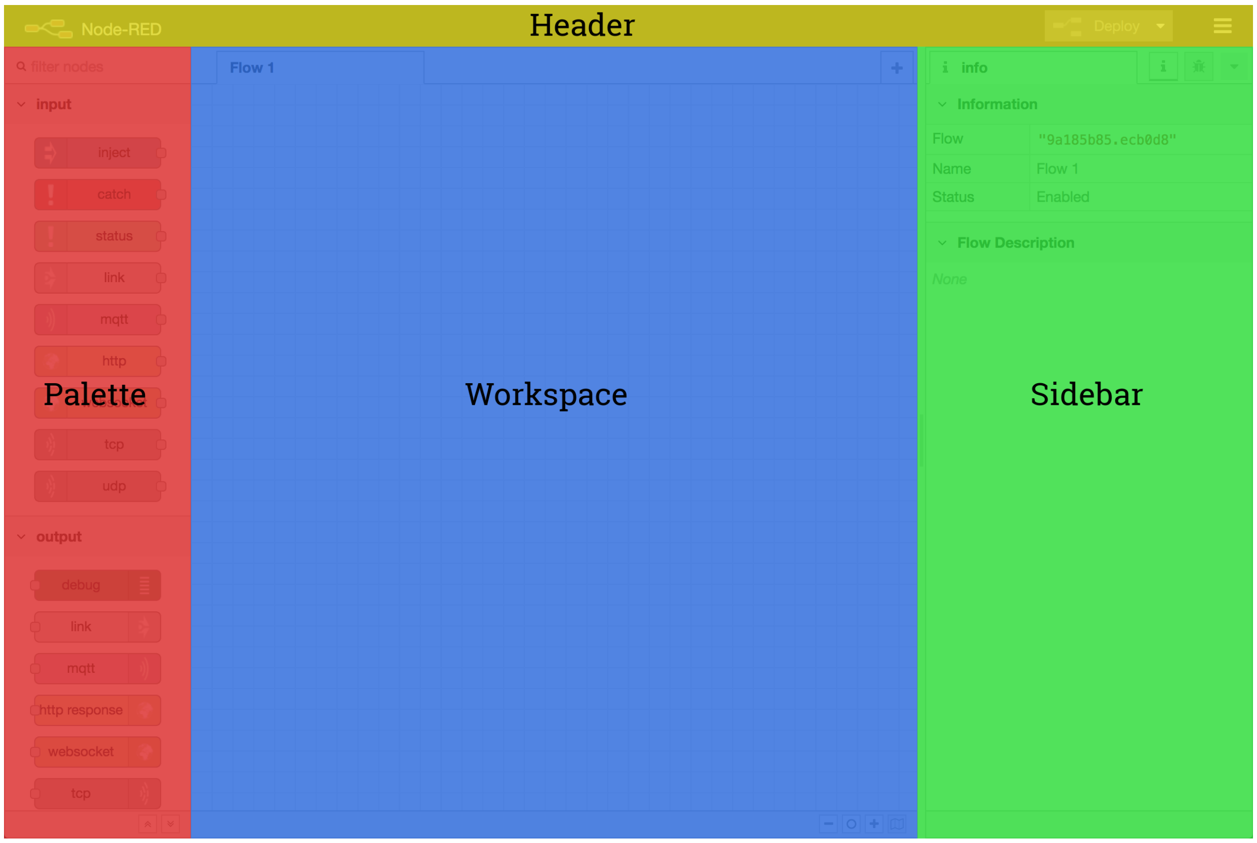


Fig. 2: Node-RED Components.

4.3.1 Header

The header contains the deploy button, main menu, and, if user authentication is activated, the user menu.

Deploy Button

The deploy button is used to deploy flows once you have finished creating or editing them. It is useful to remark that you must always deploy a flow after editing it so the changes are applied.

Main Menu

The main menu contains many actions, such as hiding components from the menu, importing or exporting flows, searching for flows, actions for flows (add, rename and delete) and subflows (create and select), managing palette, settings, keyboard shortcuts and a link to Node-RED's webpage.

User Menu

The user menu, if the user is authenticated, shows your username and the option to logout.

4.3.2 Palette

The palette contains all of the nodes that are installed and available to use. These nodes are organized into a number of categories, which can be expanded or collapsed by clicking its header.

The entire palette can be hidden by clicking the toggle button that is shown when the mouse is over it or by pressing **Ctrl+p**.[1]

[1] - Palette (https://nodered.org/docs/user-guide/editor/palette/)

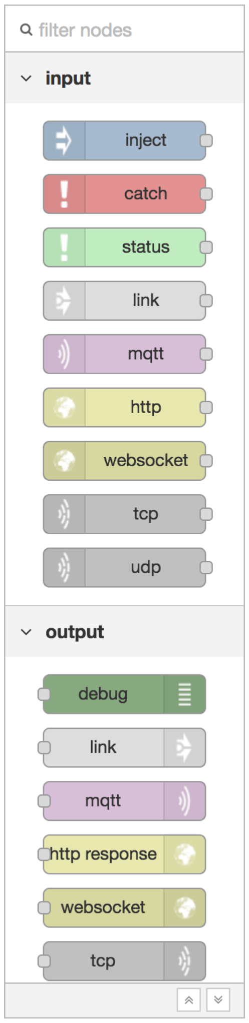
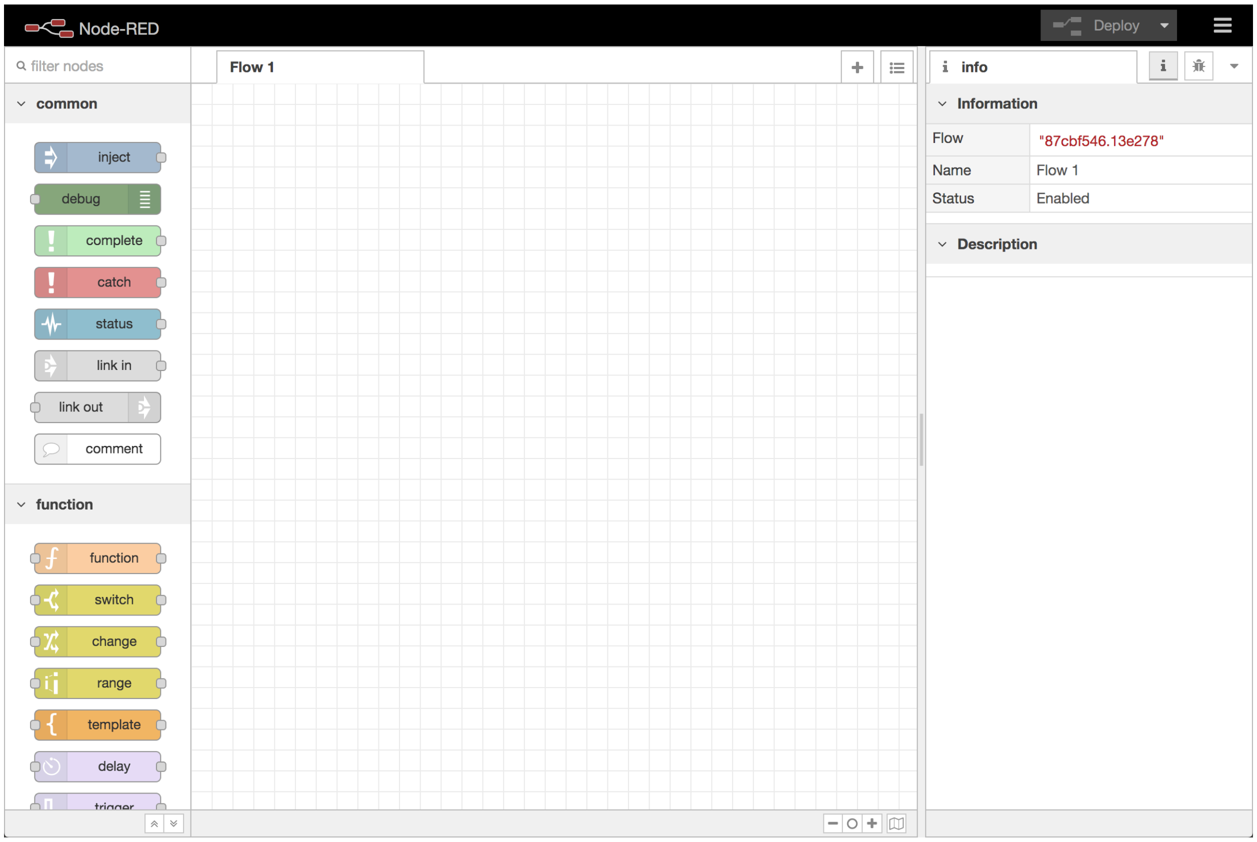


Fig. 3: Node-RED Editor Palette.

4.3.3 Workspace

The main workspace is where flows are developed by dragging nodes from the palette and wiring them together.

The workspace has a row of tabs along the top; one for each flow and any subflows that have been opened. [2]



Flow	"87cbf546.13e278"
Name	Flow 1
Status	Enabled

Fig. 4: Node-RED Editor Workspace.

View Tools

The footer of the workspace contains buttons to zoom in and out as well as to reset the default zoom level. It also contains a toggle button for the view navigator.

To zoom in, either click the **+** button inside the view navigator or press **Ctrl+=**.

To zoom out, either click the **-** button inside the view navigator or press **Ctrl+-**.

To reset the zoom, either click the **O** button inside the view navigator or press **Ctrl+0**.

The view navigator provides a scaled down view of the entire workspace, highlighting the area currently visible. That area can be dragged around the navigator to quickly jump to other parts of the workspace. It is also useful for finding nodes that have been **lost** to the further edges of the workspace.

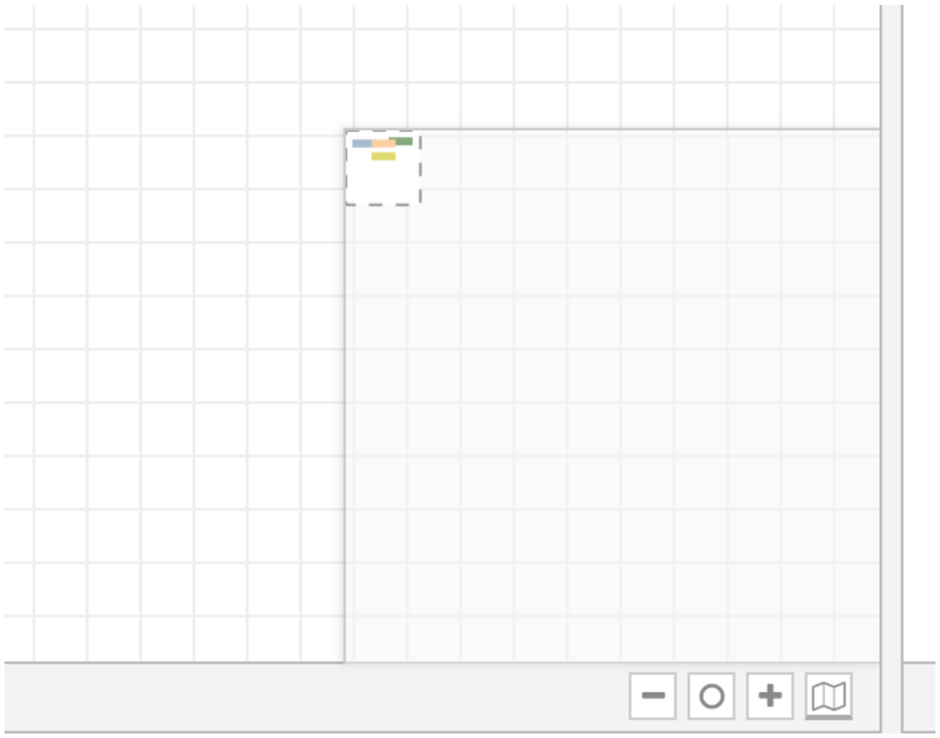
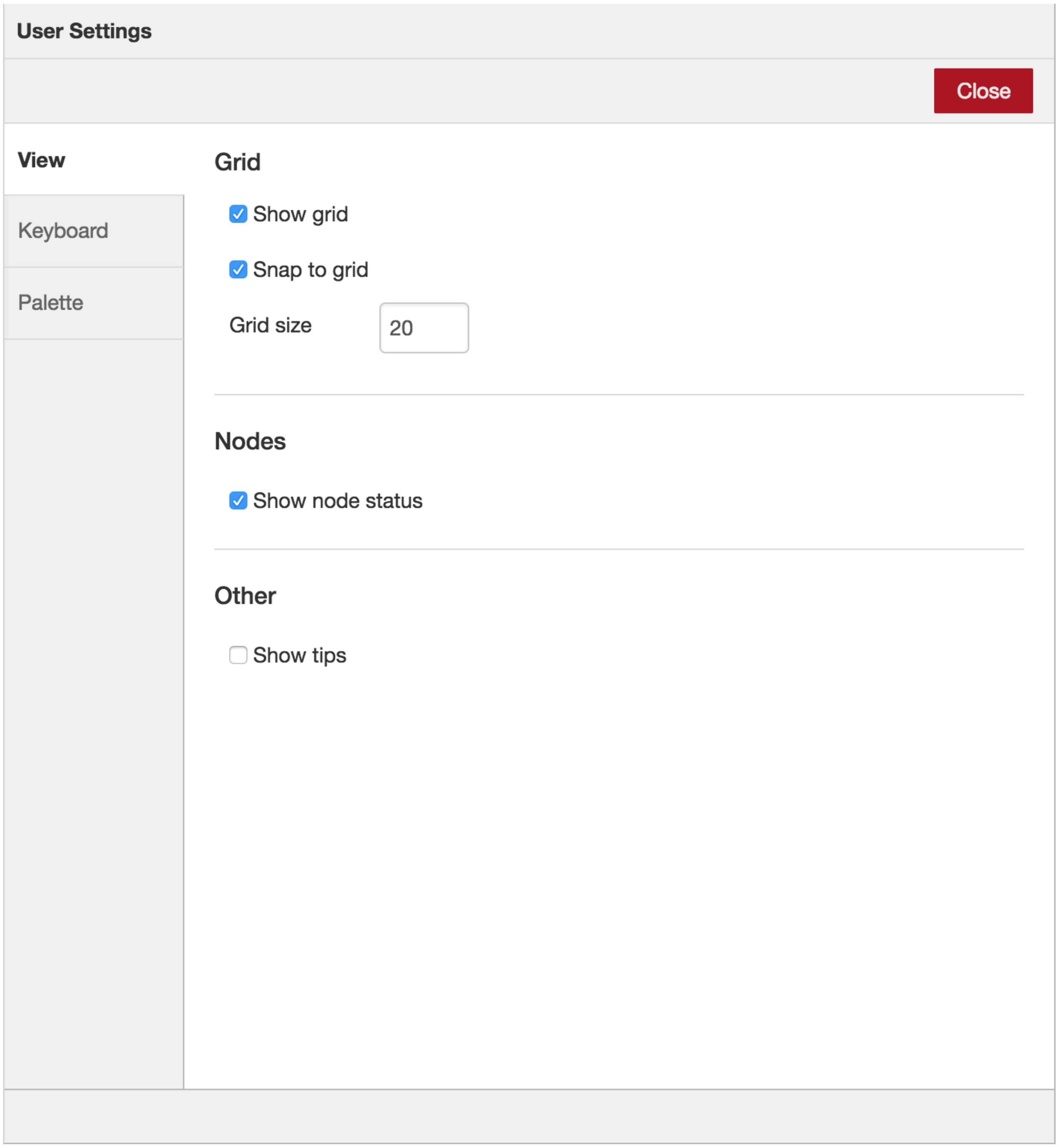


Fig. 5: Node-RED Editor Workspace Navigator Active.

Customising the view

The workspace view can be customised via the **View** tab of the User Settings dialog.

To activate the User Settings dialog, press **Ctrl+,**.


User Settings

Close

View

Keyboard

Palette

Grid

[✓] Show grid

[✓] Snap to grid

Grid size: 20

Nodes

[✓] Show node status

Other

[ ] Show tips

Fig. 6: Node-RED Editor User Settings Dialog.

[2] - Workspace (https://nodered.org/docs/user-guide/editor/workspace/)

4.3.4 Sidebar

The sidebar contains panels that provide a number of useful tools within the editor.[3]

- **Information**

View information about nodes and their help info

- **Debug**

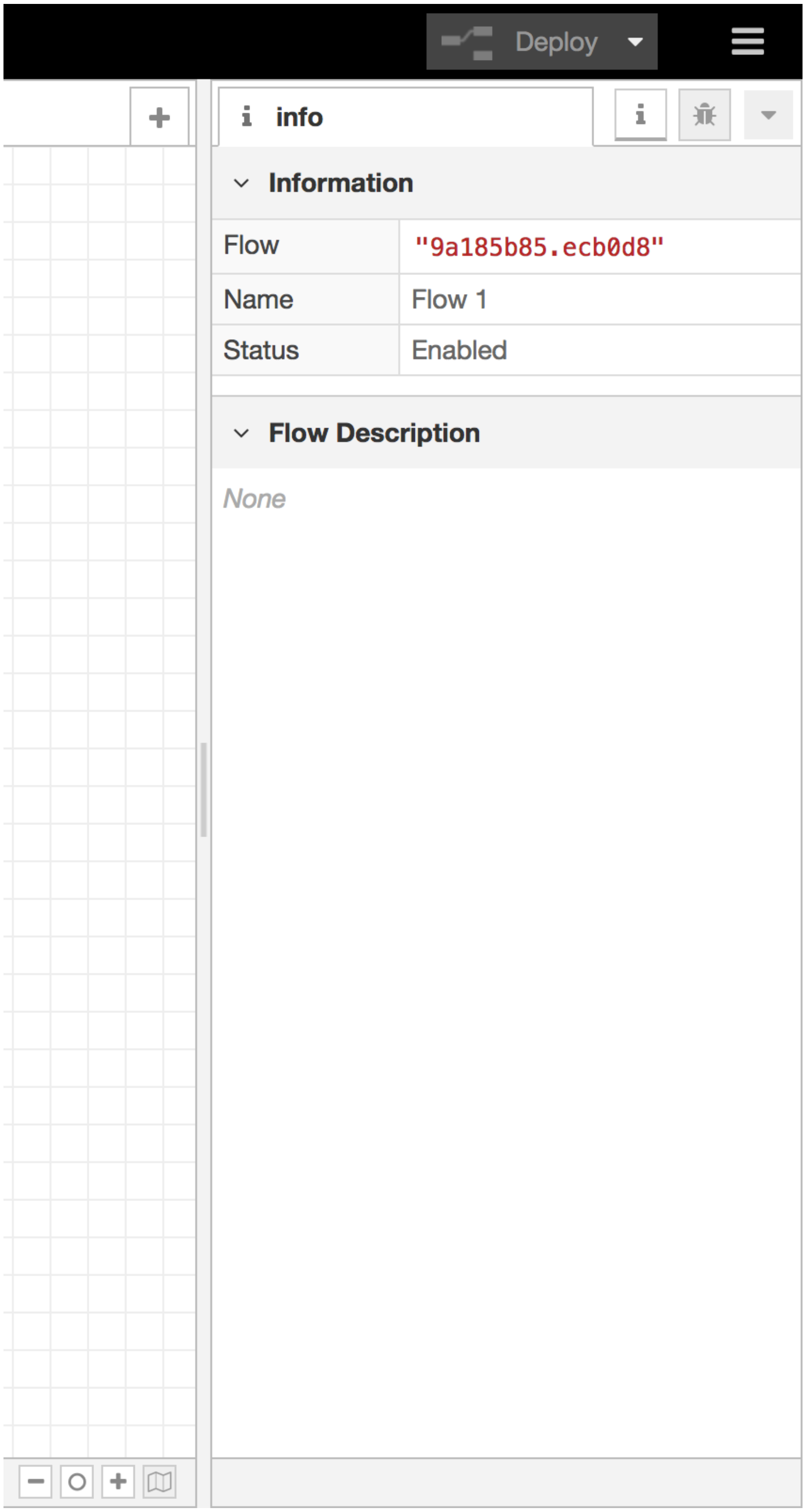
  View messages passed into debug nodes

- **Configuration Nodes**

  Manage configuration nodes

- **Context Data**

  View the contents of the context variables



Information

Flow	"9a185b85.ecb0d8"
Name	Flow 1
Status	Enabled

Flow Description

None

Fig. 7: Node-RED Editor Sidebar.

Some nodes contribute their own sidebar panels, such as node-red-dashboard (https://flows.nodered.org/node/node-red-dashboard).

The panels are opened by clicking their icon in the header of the sidebar, or by selecting them in the drop-down list shown.

The sidebar can be resized by dragging tis edge across the workspace.

If the edge is dragged close to the right-hand edge, the sidebar will be hidden. It can be shown again by selecting the **Show sidebar** option in the View menu, or using the **Ctrl+Space** shortcut.

4.4 Flow, Subflows, Nodes and Messages

Coming soon - Work in progress

4.5 OpenFlow/OpenRPA Nodes

Coming soon - Work in progress

4.5.1 RPA Detector

Coming soon - Work in progress

4.5.2 RPA Workflow

Coming soon - Work in progress

4.5.3 SMTPServer In

Coming soon - Work in progress

4.5.4 Workflow In

Coming soon - Work in progress

4.5.5 Workflow Out

Coming soon - Work in progress

4.6 Flow Examples

Coming soon - Work in progress

4.6.1 Dummy Integration OpenRPA-OpenFlow-NodeRED

Coming soon - Work in progress

4.6.2 Using OpenFlow Forms

Coming soon - Work in progress

4.6.3 AI Image Recognition

Coming soon - Work in progress

4.6.4 Email Receive, Send

Coming soon - Work in progress

4.6.5 Creating a HTTP endpoint

Coming soon - Work in progress

4.6.6 Excel Read

Coming soon - Work in progress

4.6.7 MongoDB Entities

Coming soon - Work in progress

Chapter 5

License

Python Module Index