Auto QA/QC

Manual

Auto QA/QC is a NodeRed based automatic quality assurance and quality control framework.

Author: Tobias Korf

IBG-3: Agrosphere of Forschungszentrum Jülich GmbH

You can find the project at <https://github.com/ibg3/auto-qaqc>

Requirements

Auto QA/QC is executed as docker container so Docker is the only requirement for operation.

For development python 3 and a IDE of choice is needed.

Installation

1. Clone the example project.

git clone https://github.com/ibg3/auto-qaqc.git

1. Run a Docker-Container:

docker run --rm -it --name pynodered -v path/to/project:/data -p 127.0.0.1:1880:1880 ibg3/auto-qaqc:latest

1. NodeRed can be accessed via 127.0.0.1:1880 in browser

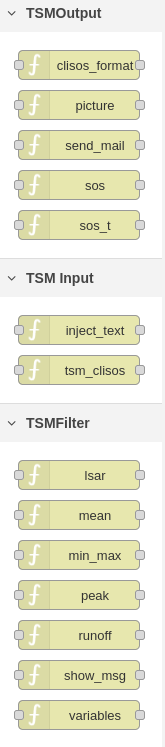
# Documentation

NodeRed works by passing through messages from one node to another. You will find several nodes on the left side (red circle) which can be used to alter these messages. Many of them come preinstalled but you can install new packages as well. Additionally, there are custom created nodes available which will be created by annotating the source code.

You can place nodes by drag and drop from the left panel. By connecting the input and output connectors you create flows (green circle).

To view debug output you can use the debug node which will print text in the debug panel (yellow circle).

## Custom nodes

All custom packages generated by the framework will be shown at the bottom of the left panel.

There are three default categories:

**TSMInput**

Category of Input Nodes, where data is retrieved from.

* tsm\_clisos will retrieve data from Sensor Observation Service specified in the node configuration

**TSMOutput**

Category of output Nodes where data can be exported to different systems

* clisos\_format will take data from processing node and converts it into sos output format
* picture will create a graph of specified variable
* send\_mail will send the data via mail
* variables will print a list of all variables in stream

**TSMFilter**

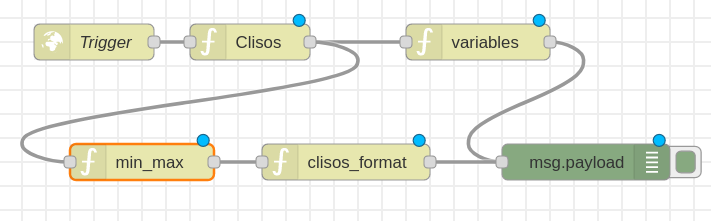
* min\_max will flag data according to node settings

Category of processing Nodes which will alter the data.

## Example flow

We want to apply a min-max filter to temperature variables *SoilTemperature\_0.05m\_Sensor1* and *SoilTemperature\_0.05m\_Sensor2* of the site WU\_B\_073.

Here is the example flow:



First of all we need a trigger which starts the flow. In this case the node "Trigger" is of type *http request* which will listen for a HTTP request at a specific URL. When it gets called the flow gets started and the GET parameters will be passed to the next node which is Clisos if type *tsm\_clisos* in this case. Note that the blue circle above the nodes indicate that there are changes made to them which are not yet deployed thus active. To activate them you need to press the deploy-Button on to top right cornet.

The output produces by this node will be sent so the *variables* and *min\_max* node.

*variables* will then print all available variables to the debug panel.

*min\_max* will flag the data according to the settings in the node itself. The output fill be formatted in clisos format in the *clisos\_format* node and then be outputted to debug panel as well.

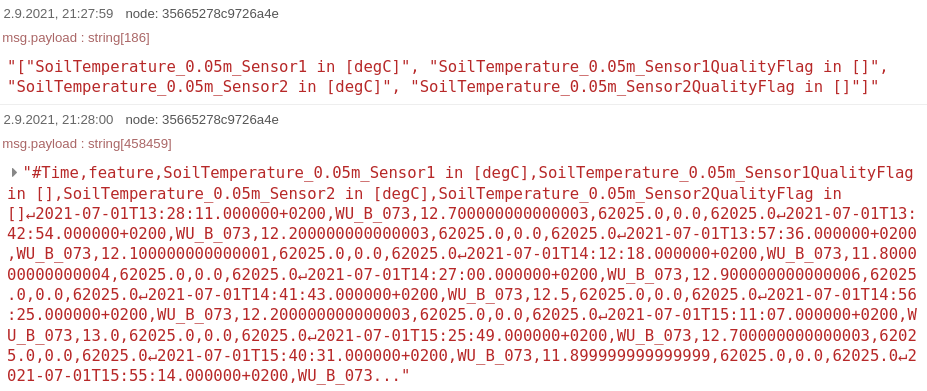
By double-clicking a node you can open its settings. Here you can configure the Nodes.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungThe node configuration of this example looks like this:

*Trigger (red) Clisos (green) min\_max (yellow)*

The output looks like this:



The first entry is from *variables* node showing the available variables and their flags.

The second entry shows the flagged data in clisos format which can be used to upload the data directly.

If you want to save the whole output you can use a node of type *file* and write it to the configured location.

**Note that files saved at /data location will be accessible from outside in your mounted folder.**

# Development of custom nodes

Node configuration for custom nodes can be specified via annotation.

You can add custom packages by annotating your Python Methods with 

annotation. (see <https://github.com/ghislainp/pynodered>)

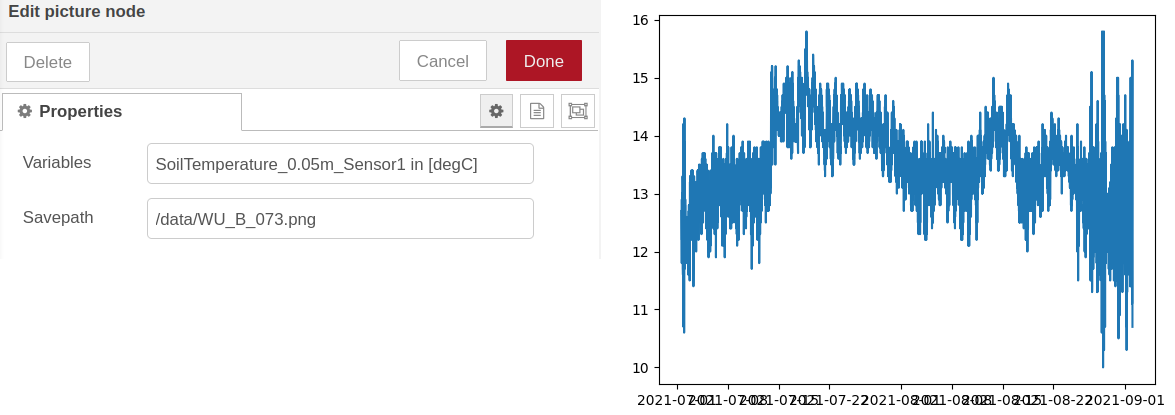
Here is the source code of the picture node.



As you can see the node name is determined by the method name. The @node\_red annotation is used to configure the behavior of the node. As you can see the node will be available in the *TSMOutput* category and there are three properties, one called *variable* which specifies the variable which should be plotted and one called *save* which specifies the save directory.

*msg* is the object which is passed from node to node. It is a dict. p*ayload* is the default entry of this dict and will be used by most build-in nodes as default input and output. You can use other entries as you wish. It is important that you return the msg object at the end. Otherwise the flow will stop at this node.

The *node* parameter contains the configuration of the nodes and can be accessed as shown.

The result looks like this: 

The shown configuration lead to the graph on the right side.