## GIS - Task #8016

## Handling Exception in Python Scripts, Logging and Creating and Publishing to a Git Repository

03/04/2019 11:15 - Debojyoti Mallick

Status:	New	Start date:	03/04/2019	
Priority:	Normal	Due date:		
Assignee:	Debojyoti Mallick	% Done:	80%	
Category:		Estimated time:	0.00 hour	
Target version:		Spent time:	0.00 hour	

## Description

Need to know how to handle exceptions in python so that we know when things go wrong in a code even though the syntax is correct.

Logging will help us have a much more detailed and organised method of viewing outputs.

Logging in Python : <u>https://docs.python.org/3/library/logging.html</u> Handling Exceptions : <u>https://realpython.com/python-exceptions/</u>

### History

### #1 - 03/04/2019 16:18 - Debojyoti Mallick

- File LoggingExample.py added

- File blaaahblah.py added

Debojyoti Mallick wrote:

Need to know how to handle exceptions in python so that we know when things go wrong in a code even though the syntax is correct.

Logging will help us have a much more detailed and organised method of viewing outputs.

Logging in Python : <u>https://docs.python.org/3/library/logging.html</u> Handling Exceptions : <u>https://realpython.com/python-exceptions/</u>

Different Logging Objects with Examples.

The example below will show you the different levels of logging objects.				
import logging				
logging.debug('This is a debug message')				
logging.info('This is an info message')				
logging.warning('This is a warning message')				
logging.error('This is an error message')				
logging.critical('This is a critical message')				
logging.basicConfig(level=logging.DEBUG)				
logging.debug('This will get logged')				
logging.basicConfig(format='%(name)s - s - %(message)s')				
logging.warning('This will get logged to a file')				
logging.basicConfig(format='(process)d-%(levelname)s-%(message)s')				
logging.warning('This is a Warning')				
logging.basicConfig(format='%(asctime)s - s', level=logging.INFO)				
logging.info('Admin logged in')				
logging.basicConfig(format='(asctime)s - %(message)s', datefmt='%d-%b-%y %H:%M:%S')				
logging.warning('Admin logged out')				

the outputs for the following would look like this:

WARNING:root:This is a warning message ERROR:root:This is an error message CRITICAL:root:This is a critical message WARNING:root:This will get logged to a file WARNING:root:This is a Warning WARNING:root:Admin logged out

Attached a file where you can run and test the output for yourself.

Now using Logging in the script for connecting with TTN.

import time import ttn import sys import logging **Imported logging object** 

logging.basicConfig(level=logging.DEBUG) We are using the Logging object level of Debug

config = {

'apps': { 'ami\_rama': "ttn-account-v2.6UEpQT1kb58BCouvltr5HLOP7CGOwZsLTWYpTo2nK3I", 'baraka': "ttn-account-v2.eo-VseYHokva5d5h7J0b9b1fMQqWPU1dzOeBGXdqwrw"

```
},
'timeout': 10
```

```
}
```

def uplink callback(msg, client):

logging.debug("Values Received from appld: {}, devld: {}, wLevel: {}, wPressure: {}, temp: {}".format(msg.dev\_id, msg.app\_id, msg.payload\_fields.wLevel, msg.payload\_fields.wPressure, msg.payload\_fields.tempC))

### The Uplink\_ Callback value will have a different format and will be under the object of Debug.

The output would look something like this: DEBUG:root:Values Received from appld: catena4450\_07\_46, devld: baraka, wLevel: 0, wPressure: 0, temp: 31.97265625

In addition to this, we have also assigned a logging object when the devices are connected and disconnected.

for example : when the script is connected to the mqtt client :
logging.info('Connected to {}'.format(app\_id)) Here we use the object INFO instead of DEBUG
mqtt\_clients[app\_id] = handlers[app\_id].data()
mqtt\_clients[app\_id].set\_uplink\_callback(uplink\_callback)
mqtt\_clients[app\_id].connect()

and when it disconnects from the mqtt client : for name, mqtt\_client in mqtt\_clients.items(): logging.info('Closing connection to {}'.format(name)) mqtt\_client.close() INFO:root:Connected to ami\_rama INFO:root:Connected to baraka

INFO:root:Closing connection to ami\_rama INFO:root:Closing connection to baraka

# Handling Python Exceptions:

Exception type of error occurs whenever syntactically correct Python code results in an error. The last line of the message indicated what type of exception error you ran into.

We define the handlers dictionary eg. handlers =  $\{\}$ . In case of our script we have defined it along with the mqtt\_clients =  $\{\}$ .

We now use the try and except block in Python is used to catch and handle exceptions. Python executes code following the try statement as a "normal" part of the program. The code that follows the except statement is the program's response to any exceptions in the preceding try clause.

As an example we use the following block: try { "RUN THIS CODE" } except { "EXECUTE THIS CODE WHEN THERE IS AN EXCEPTION" } else { "Continue to connect to the mqtt client"

#### So the script looks like this:

for app\_id, access\_key in config['apps'].items():
try:
handlers[app\_id] = ttn.HandlerClient(app\_id, access\_key)
except Exception as err:
logging.debug(str(err))
logging.warning(err.\_state.details)
else:

1. using mqtt client logging.info('Connected to {}'.format(app\_id)) mqtt\_clients[app\_id] = handlers[app\_id].data() mqtt\_clients[app\_id].set\_uplink\_callback(uplink\_callback) mqtt\_clients[app\_id].connect()

### So the final script looks like this:

import time import ttn import sys import logging

logging.basicConfig(level=logging.DEBUG)

config = {
'apps': {
'ami\_rama': "ttn-account-v2.6UEpQT1kb58BCouvltr5HLOP7CGOwZsLTWYpTo2nK3I",
'baraka': "ttn-account-v2.eo-VseYHokva5d5h7J0b9b1fMQqWPU1dzOeBGXdqwrw"
},
'timeout': 10
}

def uplink\_callback(msg, client):

logging.debug("Values Received from appld: {}, devld: {}, wLevel: {}, wPressure: {}, temp: {}".format(msg.dev\_id, msg.app\_id, msg.payload\_fields.wLevel, msg.payload\_fields.wPressure, msg.payload\_fields.tempC))

handlers = {} mqtt\_clients = {}

for app\_id, access\_key in config['apps'].items():
try:
handlers[app\_id] = ttn.HandlerClient(app\_id, access\_key)
except Exception as err:
logging.debug(str(err))
logging.warning(err.\_state.details)
else:

 using mqtt client logging.info('Connected to {}'.format(app\_id)) mqtt\_clients[app\_id] = handlers[app\_id].data() mqtt\_clients[app\_id].set\_uplink\_callback(uplink\_callback) mqtt\_clients[app\_id].connect()

if len(handlers) == 0: sys.exit(0) else: time.sleep(600)

for name, mqtt\_client in mqtt\_clients.items():
logging.info('Closing connection to {}'.format(name))
mqtt\_client.close()

#### The output would look something like this... In case there are no errors:

INFO:root:Connected to ami\_rama INFO:root:Connected to baraka DEBUG:root:Values Received from appld: catena4450\_07\_46, devld: baraka, wLevel: 0, wPressure: 0, temp: 31.97265625 But there will always be a errors.... unamused.png So let us consider the output when there are network connectivity problems.

DEBUG:root:<\_Rendezvous of RPC that terminated with: status = StatusCode.UNAVAILABLE details = "Name resolution failure" debug\_error\_string = "{"created":"@1554288214.69000000","description":"Failed to create >subchannel","file":"src/core/ext/filters/client\_channel/client\_channel.cc","file\_line":2267,"referenced\_errors":[{"created":"@1554288214.6900000 00","description":"Name resolution failure","file":"src/core/ext/filters/client\_channel/request\_routing.cc","file\_line":166,"grpc\_status":14]]}"

WARNING:root:Name resolution failure

Leaving the script attached just for reference ( This is still a work in progress)

## #2 - 03/04/2019 16:20 - Debojyoti Mallick

- % Done changed from 0 to 30

#### #3 - 04/04/2019 16:54 - Debojyoti Mallick

- % Done changed from 30 to 50

#### #4 - 04/04/2019 16:55 - Debojyoti Mallick

Debojyoti Mallick wrote:

Debojyoti Mallick wrote:

Debojyoti Mallick wrote:

Need to know how to handle exceptions in python so that we know when things go wrong in a code even though the syntax is correct.

Logging will help us have a much more detailed and organised method of viewing outputs.

Different Logging Objects with Examples.

The example below will show you the different levels of logging objects. import logging logging.debug('This is a debug message') logging.info('This is an info message') logging.warning('This is a warning message') logging.error('This is an error message') logging critical('This is a critical message') logging.basicConfig(level=logging.DEBUG) logging.debug('This will get logged') logging.basicConfig(format='%(name)s - s - %(message)s') logging.warning('This will get logged to a file') logging.basicConfig(format='(process)d-%(levelname)s-%(message)s') logging.warning('This is a Warning') logging.basicConfig(format='%(asctime)s - s', level=logging.INFO) logging.info('Admin logged in') logging.basicConfig(format='(asctime)s - %(message)s', datefmt='%d-%b-%y %H:%M:%S') logging.warning('Admin logged out')

the outputs for the following would look like this:

WARNING:root:This is a warning message ERROR:root:This is an error message CRITICAL:root:This is a critical message WARNING:root:This will get logged to a file WARNING:root:This is a Warning WARNING:root:Admin logged out

Attached a file where you can run and test the output for yourself.

Now using Logging in the script for connecting with TTN.

import time import ttn import sys import logging **Imported logging object** 

logging.basicConfig(level=logging.DEBUG) We are using the Logging object level of Debug

### $config = \{$

'apps': {

```
'ami_rama': "ttn-account-v2.6UEpQT1kb58BCouvltr5HLOP7CGOwZsLTWYpTo2nK3I", 'baraka': "ttn-account-v2.eo-VseYHokva5d5h7J0b9b1fMQqWPU1dzOeBGXdqwrw" }.
```

```
'timeout': 10
```

def uplink\_callback(msg, client):

logging.debug("Values Received from appld: {}, devld: {}, wLevel: {}, wPressure: {}, temp: {}".format(msg.dev\_id, msg.app\_id, msg.payload\_fields.wLevel, msg.payload\_fields.wPressure, msg.payload\_fields.tempC))

The Uplink\_ Callback value will have a different format and will be under the object of Debug.

The output would look something like this: DEBUG:root:Values Received from appld: catena4450\_07\_46, devld: baraka, wLevel: 0, wPressure: 0, temp: 31.97265625

In addition to this, we have also assigned a logging object when the devices are connected and disconnected.

for example : when the script is connected to the mqtt client :
logging.info('Connected to {}'.format(app\_id)) Here we use the object INFO instead of DEBUG
mqtt\_clients[app\_id] = handlers[app\_id].data()
mqtt\_clients[app\_id].set\_uplink\_callback(uplink\_callback)
mqtt\_clients[app\_id].connect()

and when it disconnects from the mqtt client : for name, mqtt\_client in mqtt\_clients.items(): logging.info('Closing connection to {}'.format(name)) mqtt\_client.close()

So the output would look like this:

INFO:root:Connected to ami\_rama INFO:root:Connected to baraka

INFO:root:Closing connection to ami\_rama INFO:root:Closing connection to baraka

## Handling Python Exceptions:

Exception type of error occurs whenever syntactically correct Python code results in an error. The last line of the message indicated what type of exception error you ran into.

We define the handlers dictionary eg. handlers = {}. In case of our script we have defined it along with the mqtt\_clients = {}.

We now use the try and except block in Python is used to catch and handle exceptions.

Python executes code following the try statement as a "normal" part of the program.

The code that follows the except statement is the program's response to any exceptions in the preceding try clause.

As an example we use the following block: try { "RUN THIS CODE" } except { "EXECUTE THIS CODE WHEN THERE IS AN EXCEPTION" } else { "Continue to connect to the mqtt client"

## So the script looks like this:

for app\_id, access\_key in config['apps'].items():
try:
handlers[app\_id] = ttn.HandlerClient(app\_id, access\_key)
except Exception as err:
logging.debug(str(err))
logging.warning(err.\_state.details)
else:

1. using mqtt client logging.info('Connected to {}'.format(app\_id)) mqtt\_clients[app\_id] = handlers[app\_id].data() mqtt\_clients[app\_id].set\_uplink\_callback(uplink\_callback) mqtt\_clients[app\_id].connect()

#### So the final script looks like this:

import time import ttn import sys import logging

logging.basicConfig(level=logging.DEBUG)

config = {
'apps': {
'ami\_rama': "ttn-account-v2.6UEpQT1kb58BCouvltr5HLOP7CGOwZsLTWYpTo2nK3I",
'baraka': "ttn-account-v2.eo-VseYHokva5d5h7J0b9b1fMQqWPU1dzOeBGXdqwrw"
},
'timeout': 10
}

def uplink\_callback(msg, client): logging.debug("Values Received from appld: {}, devld: {}, wLevel: {}, wPressure: {}, temp: {}".format(msg.dev\_id, msg.app\_id, msg.payload\_fields.wLevel, msg.payload\_fields.wPressure, msg.payload\_fields.tempC))

handlers = {} mqtt\_clients = {}

for app\_id, access\_key in config['apps'].items():
try:
handlers[app\_id] = ttn.HandlerClient(app\_id, access\_key)
except Exception as err:

logging.debug(str(err)) logging.warning(err.\_state.details) else:

1. using mqtt client logging.info('Connected to {}'.format(app\_id)) mqtt\_clients[app\_id] = handlers[app\_id].data() mqtt\_clients[app\_id].set\_uplink\_callback(uplink\_callback) mqtt\_clients[app\_id].connect()

if len(handlers) == 0: sys.exit(0) else: time.sleep(600)

for name, mqtt\_client in mqtt\_clients.items():
logging.info('Closing connection to {}'.format(name))
mqtt\_client.close()

The output would look something like this... In case there are no errors:

INFO:root:Connected to ami\_rama INFO:root:Connected to baraka DEBUG:root:Values Received from appld: catena4450\_07\_46, devld: baraka, wLevel: 0, wPressure: 0, temp: 31.97265625 DEBUG:root:Values Received from appld: catena4450\_07\_46, devld: baraka, wLevel: 0, wPressure: 0, temp: 31.93359375 INFO:root:Closing connection to ami\_rama INFO:root:Closing connection to baraka

But there will always be a errors.... unamused.png So let us consider the output when there are network connectivity problems.

DEBUG:root:<\_Rendezvous of RPC that terminated with: status = StatusCode.UNAVAILABLE details = "Name resolution failure" debug\_error\_string = "{"created":"@1554288214.69000000","description":"Failed to create >subchannel","file":"src/core/ext/filters/client\_channel/client\_channel.cc","file\_line":2267,"referenced\_errors":[{"created":"@155428821 4.690000000","description":"Name resolution failure","file":"src/core/ext/filters/client\_channel/request\_routing.cc","file\_line":166,"grpc\_status":14]]}"

WARNING:root:Name resolution failure

Leaving the script attached just for reference (This is still a work in progress)

## Creating and Initializing a GIT repository to monitor changes to the script.

Download GIT 64bit for Windows on your machine and Run the Setup. (You can download the repository from : https://git-scm.com/download/win

Run the setup with PublicKey and install in directory : C:\Program Files\Git

Once Installed open SCM on Visual Studio Code. ( View > SCM)

A marker will appear on the top right corner of the SCM tab and you will get the option to Initialize GIT repository.

Click and select the path to the python file.

Now you will be able to monitor any changes you've made to your script. On the SCM panel, you will find changes which will show you a side by side view of the original and the changed script.

The next step is to push the repository to a server and you can do so by providing the PublicKey cat id\_rsa.pub.

Once this repository is pushed on the server it will allow everybody else to monitor the edits , clone the repository and make edits themselves.

The Public Key for this repository is as follows:

#### \*ssh-rsa

AAAAB3NzaC1yc2EAAAADAQABAAABAQChBSYJ3wl445ZVHR5Vka6ohMDl3dbEtFkg4eoqYIszOZAlk03GWzBDL+y8dX8ANgbJLI7VwAE1XB 696vd1+5mqZi5Mx81Nbc6qyMBpcnS4mvxnhV5Z0OT4+Pz0hztL3YH70jpanwLfne93WC1JQcgV7xidElerp9zZyq6f6tzNxHvpUWojQqemhP2lQ06 z97JL/aGIKBaBF7NR9d2CMapdbmXt9vb8lxiiwrD92rwEibTPn63aTXDMUOYAQ+V9bkB6wwtUKP1vbOCsOEBr6APC3UhHzr0En6wv+NRRQ+9 HEIT4wBM3r8Plp/7W1XVdgQe7h4jKwXi3MLCP+4YjPbm3 Dell@DESKTOP-O3NBA08 \*

#### #5 - 03/05/2019 11:14 - Debojyoti Mallick

- % Done changed from 50 to 80

Debojyoti Mallick wrote:

Debojyoti Mallick wrote:

Need to know how to handle exceptions in python so that we know when things go wrong in a code even though the syntax is correct.

Logging will help us have a much more detailed and organised method of viewing outputs.

Logging in Python : <u>https://docs.python.org/3/library/logging.html</u> Handling Exceptions : <u>https://realpython.com/python-exceptions/</u>

Different Logging Objects with Examples.

The example below will show you the different levels of logging objects.
import logging
logging.debug('This is a debug message')
logging.info('This is an info message')
logging.warning('This is a warning message')
logging.error('This is an error message')
logging.critical('This is a critical message')
logging.basicConfig(level=logging.DEBUG)
logging.debug('This will get logged')
logging.basicConfig(format='%(name)s - s - %(message)s')
logging.warning('This will get logged to a file')
logging.basicConfig(format='(process)d-%(levelname)s-%(message)s')
logging.warning('This is a Warning')
logging.basicConfig(format='%(asctime)s - s', level=logging.INFO)
logging.info('Admin logged in')
logging.basicConfig(format='(asctime)s - %(message)s', datefmt='%d-%b-%y %H:%M:%S')
logging.warning('Admin logged out')

the outputs for the following would look like this:

WARNING:root:This is a warning message ERROR:root:This is an error message CRITICAL:root:This is a critical message WARNING:root:This will get logged to a file WARNING:root:This is a Warning WARNING:root:Admin logged out

Attached a file where you can run and test the output for yourself.

Now using Logging in the script for connecting with TTN.

import time import ttn import sys import logging **Imported logging object** 

logging.basicConfig(level=logging.DEBUG) We are using the Logging object level of Debug

'apps': { 'ami\_rama': "ttn-account-v2.6UEpQT1kb58BCouvItr5HLOP7CGOwZsLTWYpTo2nK3I", 'baraka': "ttn-account-v2.eo-VseYHokva5d5h7J0b9b1fMQqWPU1dzOeBGXdqwrw" }, 'timeout': 10 }

def uplink\_callback(msg, client): logging.debug("Values Received from appld: {}, devld: {}, wLevel: {}, wPressure: {}, temp: {}".format(msg.dev\_id, msg.app\_id, msg.payload\_fields.wLevel, msg.payload\_fields.wPressure, msg.payload\_fields.tempC))

The Uplink\_ Callback value will have a different format and will be under the object of Debug.

The output would look something like this: DEBUG:root:Values Received from appld: catena4450\_07\_46, devld: baraka, wLevel: 0, wPressure: 0, temp: 31.97265625

In addition to this, we have also assigned a logging object when the devices are connected and disconnected.

for example : when the script is connected to the mqtt client :
logging.info('Connected to {}'.format(app\_id)) Here we use the object INFO instead of DEBUG
mqtt\_clients[app\_id] = handlers[app\_id].data()
mqtt\_clients[app\_id].set\_uplink\_callback(uplink\_callback)
mqtt\_clients[app\_id].connect()

and when it disconnects from the mqtt client : for name, mqtt\_client in mqtt\_clients.items():

logging.info('Closing connection to {}'.format(name)) mqtt\_client.close()

So the output would look like this:

INFO:root:Connected to ami\_rama INFO:root:Connected to baraka

INFO:root:Closing connection to ami\_rama INFO:root:Closing connection to baraka

# Handling Python Exceptions:

Exception type of error occurs whenever syntactically correct Python code results in an error. The last line of the message indicated what type of exception error you ran into.

We define the handlers dictionary eg. handlers = {}. In case of our script we have defined it along with the mqtt\_clients = {}.

We now use the try and except block in Python is used to catch and handle exceptions. Python executes code following the try statement as a "normal" part of the program. The code that follows the except statement is the program's response to any exceptions in the preceding try clause.

As an example we use the following block: try { "RUN THIS CODE" } except { "EXECUTE THIS CODE WHEN THERE IS AN EXCEPTION" } else { "Continue to connect to the mqtt client" } So the script looks like this:

for app\_id, access\_key in config['apps'].items():
try:
handlers[app\_id] = ttn.HandlerClient(app\_id, access\_key)
except Exception as err:
logging.debug(str(err))
logging.warning(err.\_state.details)
else:

1. using mqtt client logging.info('Connected to {}'.format(app\_id)) mqtt\_clients[app\_id] = handlers[app\_id].data() mqtt\_clients[app\_id].set\_uplink\_callback(uplink\_callback) mqtt\_clients[app\_id].connect()

So the final script looks like this:

import time import ttn import sys import logging

logging.basicConfig(level=logging.DEBUG)

config = {
 'apps': {
 'ami\_rama': "ttn-account-v2.6UEpQT1kb58BCouvltr5HLOP7CGOwZsLTWYpTo2nK3I",
 'baraka': "ttn-account-v2.eo-VseYHokva5d5h7J0b9b1fMQqWPU1dzOeBGXdqwrw"
 },
 'timeout': 10

}

def uplink\_callback(msg, client):

logging.debug("Values Received from appld: {}, devld: {}, wLevel: {}, wPressure: {}, temp: {}".format(msg.dev\_id, msg.app\_id, msg.payload\_fields.wLevel, msg.payload\_fields.wPressure, msg.payload\_fields.tempC))

handlers = {} mqtt\_clients = {}

for app\_id, access\_key in config['apps'].items():
try:
handlers[app\_id] = ttn.HandlerClient(app\_id, access\_key)
except Exception as err:
logging.debug(str(err))
logging.warning(err.\_state.details)
else:

```
1. using mqtt client
logging.info('Connected to {}'.format(app_id))
mqtt_clients[app_id] = handlers[app_id].data()
mqtt_clients[app_id].set_uplink_callback(uplink_callback)
mqtt_clients[app_id].connect()
```

if len(handlers) == 0: sys.exit(0) else: time.sleep(600)

for name, mqtt\_client in mqtt\_clients.items():
logging.info('Closing connection to {}'.format(name))
mqtt\_client.close()

The output would look something like this... In case there are no errors:

INFO:root:Connected to ami\_rama INFO:root:Connected to baraka DEBUG:root:Values Received from appld: catena4450\_07\_46, devld: baraka, wLevel: 0, wPressure: 0, temp: 31.97265625 DEBUG:root:Values Received from appld: catena4450\_07\_46, devld: baraka, wLevel: 0, wPressure: 0, temp: 31.93359375 INFO:root:Closing connection to ami\_rama INFO:root:Closing connection to baraka

But there will always be a errors.... unamused.png So let us consider the output when there are network connectivity problems.

DEBUG:root:<\_Rendezvous of RPC that terminated with: status = StatusCode.UNAVAILABLE details = "Name resolution failure" debug\_error\_string = "{"created":"@1554288214.69000000","description":"Failed to create >subchannel","file":"src/core/ext/filters/client\_channel/client\_channel.cc","file\_line":2267,"referenced\_errors":[{"created":"@1554288214.690 000000","description":"Name resolution failure","file":"src/core/ext/filters/client\_channel/request\_routing.cc","file\_line":166,"grpc\_status":14]]}"

WARNING:root:Name resolution failure

Leaving the script attached just for reference (This is still a work in progress)

Files			
LoggingExample.py	778 Bytes	03/04/2019	Debojyoti Mallick
blaaahblah.py	1.4 KB	03/04/2019	Debojyoti Mallick