Introduction

This document is written as a general reference to the project. Specific references, such as the one relating to the database, are written on their own separate file or in-script. It's written following the ISO-9001 structure so it is affordable knowledge-wise for everyone who reads it

Vocabulary

Server: a computer which manages the access to resources or the execution of applications. Usually used in a net of computers and not standalone.

Command: simplified instruction which allows an action to be performed just by inputing it along some parameters, if any.

UNIX shell: command interpreter used by UNIX-based operating systems (example gratia, GNU/Linux) which holds the ability to control the core of the computer by the use of commands.

(Information) Package: set of data which is sent across the net

Ping (command): UNIX shell command which sends four packages to a certain server and checks the arrival of these both in answering and fulfilling terms (how fast it answers back and how many packages get across).

Ping (verb): using the ping command with a concrete server.

Ping (noun): measurement of answer speed of a server when pinged. It's given in milliseconds (a ping of 60 means that the pinged server has sent the ok-signal with a 60 milliseconds delay).

Probe (verb): originally, to introduce a device inside something to measure something or record information (e.g. dropping a stone tied up with a rope into a river to measure its depth). Pertaining to our case, pinging a server in order to check the connectivity.

Probe (noun): device used for probing. Pertaining our case, a script which pings continuously a server and records the information.

CRUD: acronym of Create, Read, Update, Delete. Set of data managing operations from which the first is designated to data reading, the second to data retrieving, the third to data altering and the fourth to data deleting.

Query: simplified command which allows a CRUD operation to be performed

SQL: acronym of Structured Query Language. An organized set of queries which allow to the CRUD operations to be applied in a database.

Database: element in the memory of a computer which allows for the storing and retrieval of data. For this purpose, they use SQL.

(Database) Connection: the means of connecting to a database, whether those are in-script or outside it is indifferent.

Object-oriented Programming: computer programming model that is organized around objects instead of functions.

Object: maximum wrapping level on Object-oriented programming. It holds both values (object parameters) and functions (object methods). In-script this wrapping level is called class, and instancing a class creates an object of said class.

Database connection object: Object created in order to stablish a connection in-script with the actual database. It stores all changes and they must be committed before terminating the connection.

Cursor object: Object from a Database connection object which executes queries written in SQL.

Python: high-level, interpreted, object-oriented programming language oriented to code readability. The version used here is the 3.8. Worth noting that all datatypes are objects, and will be referred as such.

“str” object: Python object consisting in a chain of characters. It has different levels of nesting: single quotes <- double quotes <- two double quotes <- three double quotes

f-string: str object which can hold parameters into it between braces ({loreipsum}).

List object: kind of object which stores separate values in sequence.

Mapping object: kind of objects which maps values further than what a list would do (given the mapping of a list is index-element). The referral value is called key, whereas the referred value is called value (therefore any mapping is of the form key-value). On Python there's only the Dictionary object, which allows mappings of anything-anything.

Object constructor: Method which allows for the instancing of a class (thus the object creation). On Python there can only be one, which has as parameters all object parameters.

Class inheritance: given a certain class, way to allow a second class to get all the parameters and methods of said class.

Method decorator: a denotation of the kind @kindofdecorator which states certain circumstances about the method which they are above.

@classmethod decorator: decorator that states that the method is used to create an instance of the object instead of just using the constructor.

@staticmethod decorator: decorator that states that the method isn't to take any parameters from the class.

Json: acronym for JavaScript Object Notation. A way to store data for quick reference or to be inserted into a database. It is translated into a mapping object when loaded in a programming language.

Programming exception: when a script on any stage cannot go further due to some circumstances (e.g. calling a method from an object which lacks it) it'll send an instruction to raise the error. If not treated in-script, this will stop the execution altogether.

Try-catch block: script block which attempts to execute some code (try) and then, if an exception is raised it checks if said exception has a block of code assigned (catch). On Python the try block is written under a “try:” and the catch block is written “except ExceptionName:”.

CSV file: being CSV the acronym for Comma Separated Values, file which allows for the organization on a table manner of values. A CSV file row will look like “data1;data2;data3;data4;…”

- # Introduction to the problem:

Let’s say we have in a project a set of servers. These servers are a core part of the project, and as such they have to be maintained. One of these things we have to maintain is their net connection but it’d be utterly impractical to send someone over, or even send an email or call them on the phone, each time the connection fails. For this, the ping probe is born.

- # Explanation of the solution:

All the documentation related to the methods per se is either in-script or on a side file to this.

Our PingProbe object is designed to have some processes executed through a UNIX server shell. This way, in order to start probing a server we just have to execute the “StartProbe.py” file. The probe parameters are written on a json file, “probeparameters.json” in order to be able to just load the object from the file. As per the version 1.0.0, the project has the following files:

1. PingProbe.py
    . File containing the PingProbe class, along all the methods to be used by the different scripts in the project.
2. StartProbe.py
    . File containing the script which, when executed, starts the probing.
    . It also registers the probe in the database were it to not be registered.
3. Error.py
    . File which contains the Error class. This file is here only to fulfill the one-class-per-file rule.
4. CustomExceptions.py
    . File which contains the customized exceptions for the scripts
    . This file has multiple classes, which clearly contradicts the previously stated rule. Even so, this is done in order to keep order within the project gathering all the customized exceptions in a single file
5. ChangeProbeAddress.py
    . File which needs two values to be called on execution, being the first the json file where the probe is stored into, and the second the value of the new address.
    . This is made so the address can be changed externally.
6. ChangeProbeRatio.py
    . File which needs two values to be called on execution, being the first the json file where the probe is stored into, and the second the value of the new ratio.
    . This is made so the address can be changed externally.
7. probeparameters.json
    . File where the parameters which make up the probe are stored
8. Probe.db
    . Sample database for the sake of showing how the project is intended to work with a database, which doesn’t need to be local
9. CreationQueries.sql
    . SQL file containing the queries used to created the sample database, which also show its structure.

10. *_dump_csv.csv
    - The * is the concatenation of the dominion the probe belongs to and the name of the probe with a "_" between them
    - This csv is used as sorts of a true local database.
    - At every ending of each probing process (read this as every time the process is terminated) a backup is created, then the data is dumped into the database and the file is cleared.
11. Backup_dump_*.csv
    - The * is the date the backup belongs to
    - This csv is the backup off the aforementioned
12. Readme.pdf
    - This file
13. Readme-database.pdf
    - Information pertaining the database, plus on how the ideal usage of the probe would be.
14. Thresholds.json
    - JSON file noting the thresholds, their codes and their status

Now that we know a bit about the files let's go in-depth on how this works. The probe has five parameters, the dominion it belongs, the name of the probe, the address it targets, the ratio (seconds it takes between probing attempts) and the status (which lets us know if it's active or inactive, plus any additional states defined by the user).

For probing, we ping the server in the address, and then we record its average ping. Then, we classify it according to a threshold which has a status and a code associated. Then the data is dumped into a csv file. The first row shows the columns of the data to be dumped: date, time, ping, status, code.

The whole status-code thing is to allow both a quick reference and detailing for a ping event. Example gratia. If I define that if the ping goes over the threshold of the 300ms or if there is no connection at all need an urgent fix because the project crumbles, I can define them as "Fatal connectivity issue, needs urgent fixing" and "No connection" respectively, both with the code "black", when searching for issues with the probe I can see all intervals with this kind of errors just by searching for the code.

There’s also this issue regarding the ping thresholds. Usually we’d define it in-code but in order to allow a simple implementation they are loaded from the file in (14). Now, its structure is simple: numbers as strings form a sort of index. There needs to be a full sequence without holes, else an error will be raised and the script will stop. Each of these indexes hold the same kind of structure, the lower level number of the threshold (the lowest number that can be in the threshold), the higher-level number (same, but with the highest one), the code and the status. Replicating the structure will suffice when defining custom thresholds.

Now the file indicated in (2) is used to start the prove via command. It’ll check if the probe is registered and if it isn’t it’ll register it. If it was registered, it’ll check if the database-stored information is coherent with the loading JSON in (7). If it isn’t, it updates the database accordingly. Only then it changes the probe status to “active”, and then starts probing. It has to be forcefully stopped and, when done, it backups the dump csv (10) into (11), then dumps the data in the database and clears the dump csv. In the end, it switches the probe to “inactive”.

The (5) and (6) files exist to allow on visual-level implementations and the such to do a change to the probe parameters via command line, and not doing a factual change via a text editor, example gratia.