|  |
| --- |
| **DSP Group Logo** |
| Chip Validation Python environment manual |
| Version 1.1 |
|  |

Table of Contents

List of Figures 3

List of Tables 4

Revision history 5

1 Introduction 6

2 Requirements 7

3 Working method 8

4 Python conventions 9

5 The environment 10

5.1 Files 10

5.1.1 init 10

5.1.2 lab\_equipment 10

5.1.3 devices 11

5.1.4 utilities 11

List of Figures

No table of figures entries found.

List of Tables

No table of figures entries found.

Revision history

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Description | Author |
| Version 1.0 | 22-Jan-17 | Creation | Bar Kristal |
| Version 1.0 | 1-Mar-17 | update | Bar Kristal |

# Introduction

This document’s purpose is to detail the using process of the python environment of the Validation team, which created by the team. The python environment is a collection of classes and functions that are needed in the daily work in the lab, including lab equipment remote control, writing to Excel, controlling the Lauterbach debugger, serial communication and etc.

The environment is written in Python 2.7.10, and should work properly in every version later than this, in the 2.7 series. In order to implement the environment on Python 3 series, there are some syntax changes that should be done in the environment code.

# Requirements

1. Python interpreter: Python 2.7.10 version and above (in the 2.7 series, not python 3)
2. Coding environment: Komodo IDE/ Windows Visual Studio/ PyCharm/ Note Pad+
3. In order to work with Lauterbach: T32 directory need to be saved in ‘c:\T32’, and to contain the python-api files, and the config.t32 file. These files can be found in G:\Chip\_Validation\T32\_python\_api.

# Working method

Python is a very simple scripting language, with an emphasis on readability and easy debugging, which reflected in the language syntax, indentations and an extensive range of study materials, guides and troubleshootings in the web. In addition, python is an Object-Oriented language, and the whole environment written here is based on this feature. Object oriented is, on one hand, a coding method which allows you to create a certain class, just like the classes ‘Int’, ‘Float’ or ‘Str’, and define its own attributes and methods. In our case, for example, we want to control several lab devices that are essential to our test, so every equipment type has its own class. When we want to work with a device, first we need to define a new attribute of the specific class, as follows: *device\_name = DeviceClassName(inputs)*. Now *device\_name* is an attribute of the class *DeviceClassName*, and we can call any of its function by the **dot** calling: *device\_name****.****do\_something().* Part of the object oriented method is that every class has it’s own funtions, however for similar classes, or for basic functions, the calling syntax can be uniform. For example: ‘Termotron3800’ and ‘VotschVT4002’ are both classes written to work with the oven types being used in our lab, respectively. The background communication with every device is absolutely different and doesn’t matter to the user, while the basic functions of the two are identical.

For example:

Initializing:

*thermotron = Termotron3800(address1)*

*vostch = VotschVT4002(address2)*

set oven temperatue:

*thermotron.set\_temp(25)*

*vostch.set\_temp(25)*

close connection with the device:

*thermotron.close()*

*vostch.close()*

As one can see, this method of programming is quite readable- in every line one can know which device is being used now, and what action it does. Also, the programming become very intuitive.

# Python conventions

Python conventions simplify the understanding and debugging of a code, and it is essential when working in a shared environment. For example, when one see a variable being used in the middle of the code, and is written with UPPER\_CASE\_LETTERS, he will immediately know that this is a global variable, and not a local one.

Partial list of the conventions:

* DEFINES\_AND\_GLOBALS: constant values and global variables are often written with upper case letters and underscores between words
* Classes: class is always written with a capital letter, followed with small letters. Two words class name is being separated with a capital letter too, without an indentation: ClassName.
* modules: modules, like libraries being imported, or files that had been written by this team members, should be written in lower case letters only, separated with underscore.
* functions: the same as for modules, only lower case letters and underscores.
* local\_variables: the same as for modules and functions.
* Initializing long lists or dictionaries- writing only two values in every line:

*new\_list = [val1, val2,*

*val3, val4,*

*val5, val6*

*val7, val8]*

* Spaces before after operators, after commas and etc:

*num = 1 + 1*

*list = [1, 2, 3]*

*if a == 5:*

*b += 1*

* NOTE: it is recommend to use [**PyCharm IDE**](https://www.jetbrains.com/pycharm/download/download-thanks.html?platform=windows&code=PCC), because it writes notes when your code syntax isn’t follow the convention rules.

# The environment

## Files

The python environment includes the following files:

* Init
* lab\_equipment
* devices
* utilities
* test1

### init

The init file is responsible of the initialization of the environment, including: import the necessary libraries, open a new log file (the name appears with the exact date), define constants (like GPIB address, folder name, etc..).

Every user should put his addresses and setting in the init.py file under his USER\_NAME, and in the test itself to initial his user\_name e.g. USER\_NAME = “Bar”.

### lab\_equipment

Lab\_equipment file gives basic operation functions, for controlling lab equipment, including:

Power supply, multimeter, electronic load, frequency counter, signal generator and ovens.

Specific devices being supported:

* Agillent34401A
* QL355TP Power Supply
* HP53131a Frequency Counter
* HP33120a Wave Generator
* KikusuiPLZ70UA electronic load
* VotschVT4002 oven
* Termotron3800 oven

### devices

devices module supplying different types of communication with DSP’s products: UART connection, Lauterbach debugger, RaspberryPi.

In the meantime, the UartDevice class is designed only for the DBM series, and when initializing an attribute of this class, it is necessary to insert the chip’s name (like ‘DBMD2’) as an argument, so the function “sync” will be adjusted to right version of the DBMx.

In order to work with the Lauterbach class, one should make sure he have the T32 directory in the path: ‘C:\T32’, and also the python-api files saved in this directory. The files can be found at G:\chip\_validation\T32\_python\_api.

RaspberryPi class sets up an ssh connection with the RaspberryPi, and lets us send commands to the RPI’s terminal. In addition, there are some functions intended for the DBMA7, that make use of the FW’s python environment.

### utilities

utilities module supplying general functions for the python environment, like opening a new text-log file or opening an Excel file.