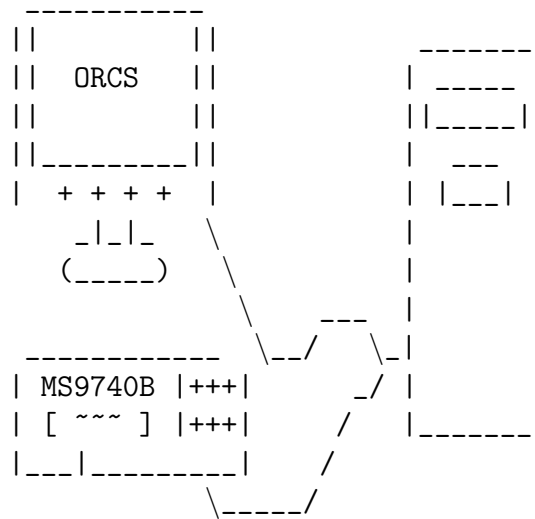


ORCS
Optical Spectrum Analyser
Remote Control System
- Operation Manual -

Emanuel Pegler
in Cooperation with
Lennart Leffers M.Sc.

November 2021



Contents

1	Introduction	3
2	Installation	4
3	Usage	5
3.1	Commands	6
4	Adding additional plugins	11

1 Introduction

ORCS is a system to control the Optical Spectrum Analyzer MS9740B by Anritsu with a remote computer. It was developed in cooperation with *Lennart Leffers M.Sc.* at the *HoT - Hannover Centre for Optical Technologie*. It is a plugin based console application written in python. In this manual the installation, the use of the system and how to easily add more functionality by writing plugins for the program will be explained.

2 Installation

For setting up the system you first need to install *Python 3*. For development *Python 3.9* was used and it is recommended to use this version or higher. The next step is to download all the Python frameworks needed. The frameworks and how to install them via pip can be seen in the table 1. If you have

Table 1: Frameworks

Framework	Version	Installation Command
PyVisa	1.11.3	pip install pyvisa
numpy	1.19.5	pip install numpy
pandas	1.2.2	pip install scipy
matplotlib	3.3.4	pip install matplotlib
scipy	1.7.2	pip install pandas

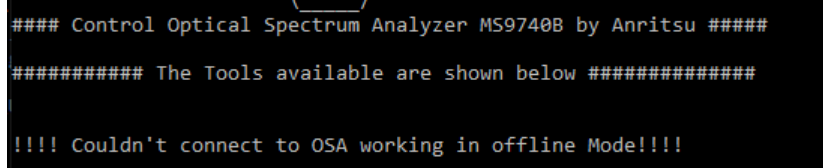
installed all the frameworks you can clone the codebase from the github repository: <https://github.com/manuelprogramming/OSA>. The last thing before you can start using ORCS is to create a saving folder and a json-file with a visa search term for connecting the system with the Optical Spectrum Analyser (OSA). The saving folder must be created in the root folder of the program. The name can be found in `.../OSA/bin/settings.json`. The name is set by default to `saved_data`. The json-file for the visa-search-term must be created in `.../OSA/bin/visa_search_term.json`. The program expects a dictionary with the folwing content:

```
{"visa_search_term": "Example-Visa-Search-Term"}
```

For more information about connecting the MS9740B check the Operation Manuel on the Anritsu website.

3 Usage

After installation you can simply run the main.py file in the main folder. If you are not connected to your MS9740B the following message will be displayed on the starting screen:



```
#### Control Optical Spectrum Analyzer MS9740B by Anritsu ####  
##### The Tools available are shown below #####  
!!!! Couldn't connect to OSA working in offline Mode!!!!
```

Figure 1: Starting Screen in Offline Mode

The program is still usable in offline mode but some functionality is not given. After starting the program will ask you to send a command. All commands can be found in table 2 below. The syntax of sending commands is straight forward and pretty simple just type in a command of the tool you want to use. A simple example would be:

SSI

With this command a single sweep is performed on the Anritsu but the data is not plotted. For plotting the data immediatly after sweeping ORCS also supports the usage of multiple commands at once. If using multiple commands you have to seperate the commands by a semicolon. An example is shown below

SSI;PLT;SVD

This example performs a single sweep and then plots the data. When the data is plotted a new window will be opened with the plotted data. Notice that the program will stop and only continue to run after closing the window of the plot. When you close the plot the data will be saved to the latest file created in the saving folder. If the folder is empty an error message will be displayed. If you're using whitespaces between the commands they will be stripped so

SSI ; PLT; SVD

is equally valid.

3.1 Commands

All command with a simple explanation can be found in this table.

Table 2: All valid commands for ORCS

Commands	Details
----------	---------

Basic Tools

EXIT/exit	Shuts down ORCS
SET?	Show the settings in the settings.json file
HLP	Opens this documentation of ORCS
OOM	Opens the Operation Manuel provided by Anritsu
PLT	Plots the data from the cache
SVD	Saves the data from the cache to the selected file. If no file is selected the latest csv file in saving folder is chosen. Displayes error message if no file exists perform "CNF". Only data with the same structure can be saved together.
CNF	Create new file in the saving folder

Commands	Details
PFF	Plots the data saved in the latest file in the saving folder
SLF	Opens a File Dialog and you can select (".csv", ".xlsx", ".xls", ".dat", ".DAT", ".txt")-Files for you to work with. Other Commands may have unexpected behavior if you choose non-empty files not created by ORCS

OSA Tools

IDN	Identify the connected OSA
ESR	Check Standart Event Status Register
CLS	Clear Event Register
SSI	Perform single sweep. The programm will stop while sweeping
SST	Stops the sweeping for the OSA
SSW	Set the start and stop wavelength set in the setting
GDA	Retrieves the data from given memory slot in the settings
MPT	Set the sampling points given in the settings

Commands	Details
VBW	Set the video band width given in the settings
SRT	Perform repeated sweep plus plotting. No more commands can be send! Stops sweeping after closing plotting window. !Sometimes ORCS breaks on closing the plot window! Just restart ORCS
OPT	Turns the light output ON or OFF accordingly
TSL	Selects the trace given in the memory slot in settings for measurering data

Change Settings

CPT	Changes the sampling points in the settings. Use "MPT" for setting the sampling points after changing it
CSTAW	Changes the start wavelength in the settings. Use "SST" for setting wavelength after changing it
CSTOW	Changes the stop wavelength in the settings. Use "SST" for setting wavelength after changing it
CMS	Changes the memory slot in the settings. Perform "GDA" to retrieve data from the memory slot
CSF	Changes the saving folder, where the data is stored and new files are created
CVBW	Changes the video band width in the settings, Use "VBW" for setting the video band width after changing it

Commands	Details
----------	---------

Analyser

FMA	Calculates the Moving Average from the cache data.
FSG	Filters the data from the cache with the Savatzki-Golay-Filter
MAX	Calculates the maxima of the given data with a peak finding algorithm
SFT	Calculates the shifts between the reference data and the cache data. Use "REF" before usage.
REF	Saves the data from the cache as the reference data for "SFT" calculations
REF?	Displays the reference data saved.

Dummy Data

DD1	Creates random dummy data for testing purpose
DD2	Creates dummy data from a file for testing purpose
DD3	Creates other dummy data from a file for testing purpose

4 Adding additional plugins

Since ORCS was build with a plugin structure adding new functionality to the system is fairly easy and can be done without changing the underlying functionality of the rest of the program. Here a step by step instruction will be provided. Some basic knowledge of python is definitely recommended.

So lets say we want to create a new tool for controlling the MS9740B. The first thing is to create a new python file in the `...\OSA\plugins` folder. For example *my_newtool.py*. Since all tools are dataclasses we need to import `dataclass`. Also every tool needs to have an attribute for the result. So we import a *BaseResult* class. And to add our tool to the system the *factory* is needed. At last since we want to create a tool for controlling the MS9740B we also need to import the *BaseAnritsu* class and the *test_anri_connection* decorator.

```
from dataclasses import dataclass
from result import BaseResult
from osa import factory
from osa.anritsu_wrapper import BaseAnritsu, test_anri_connection
```

After doing all the basic import we can write a class which is our new tool like shown in the example below. Note for a tool which doesn't control the MS9740B we don't need the attribute *anri*.

```
@dataclass
class MyNewTool:
    """
    Explanation of what the new tool can do
    """
    command: str
    result: BaseResult
    anri: BaseAnritsu

    @test_anri_connection
    def do_work(self) -> BaseResult:
        """Here you can add your functionality"""
```

If the tool has the *anri* attribute it can be used quite easy for querying and writing to the MS9740B like shown in the example below. Also every tool

with the *anri* attribute needs to have the decorator *test_anri_connection* so the program doesn't break in offline mode.

```
self.anri.query("EXAMPLE QUERY") # sends a query to the connected OSA
self.anri.write("EXAMPLE WRITE") # writes a command to the connected OSA
```

Also note that every tool has a *result* attribute, which has to be returned and displayed after calling *do_work* method. You can add values and messages like shown below. Also the result has a certain *ResultType*. This will determine how the data is stored and if it is for example possible to plot the *result.value*. Only *ResultType.arrayResult* can be plotted. Array results are a tuple of two numpy arrays. The two other ResultTypes are *ResultType.noneResult* and *ResultType.valueResult*. In our case we would have a *ResultType.valueResult*.

```
self.result.msg = "Done work successully " # The displayed message
self.anri.value = 23 # Some example result Value
```

After writing the tool class we need to define a function for registration of the tool in the factory. This function must be written in the same *my_newtool.py* file. The function is called *initialize* and looks like this:

```
def initialize() -> None:
    factory.register("my_newtool", MyNewTool)
```

So this finishes the module of the new plugin. Now the last thing needs to be done is to edit the *tools_data.json* file in *...\OSA\bin*. Here we add our filepath to the plugins list and a dictionary with the properties to the tools list. The command string and the *ResultType* is defined like shown in figure 2. The name given here must be the same as in the *initialize* method we defined earlier. The *result* and *anri* attributes are set to null at the beginning.

```
22     "plugins.analyser.load_ref",
23     "plugins.dummy_data.dummy_data",
24     "plugins.dummy_data.dummy_data_2",
25     "plugins.dummy_data.dummy_data_3",
26     "plugins.my_newtool"
27 ],
28 "tools": [
29   {
30     "name": "identify",
31     "command": "IDN",
32     "result": null,
33     "result_type": "ResultType.noneResult",
34     "anri": null
35   },
36   {
37     "name": "my_newtool",
38     "command": "MNT",
39     "result": null,
40     "result_type": "ResultType.valueResult",
41     "anri": null
42   },
43   {
44     "name": "standard_event_status_register",
45     "command": "FSR",
```

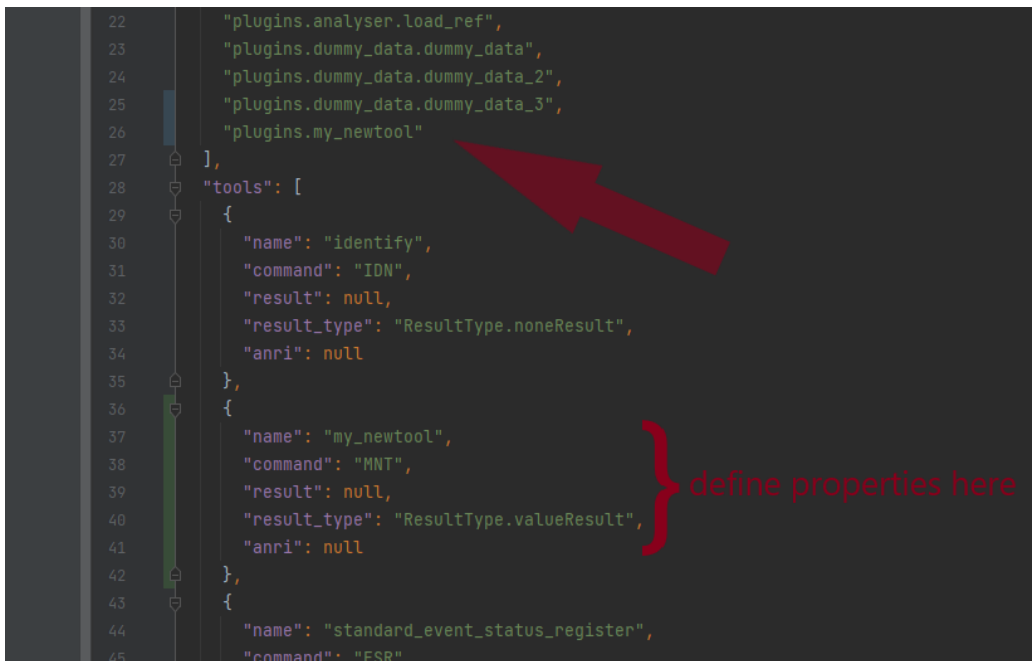


Figure 2: Example of the tools_data.json file after adding a tool