1 BrukerPy

Installation and operation manual the BrukerPy Python Software

1.1 About this manual

This manual describes the NIWA developed BrukerPy python program used to drive OPUS and Bruker instruments. Please read this document in its entirety before installing and using.

1.2 Version

This version of the manual is dated 3rd August 2017 and is relevant to BrukerPy version 2.0

1.3 Contact

For further information, comments or help please contact:

National Institute of Water and Atmosphere
Private Bag 50061
Omakau
Central Otago
New Zealand
Ph: 64-3-440 0055
alex.geddes@niwa.co.nz

1.4 Table of Contents

2 Introduction

BrukerPy is a Python program, developed at NIWA Lauder, to drive OPUS, where OPUS itself is used to operate the Bruker FTS instruments. BrukerPy is needed because making repeated measurements that require detailed scheduling is complicated within the limited macro functionality of OPUS. Driving OPUS with Python enables us to schedule measurements by solar zenith angle (SZA), time and location as well as optimizing the process to increase the useful measurement density. We can also incorporate a wide range of additional data, here refered to as auxiliary data, such as temperature, pressure and tracker status to simultaneously reduce the instances of bad spectra and increase the number of good spectra. This is done with a high degree of flexibility so that the user can tailor the program to their needs.

2.1 Development History

BrukerPy is the latest in a long line of Bruker FTS automation software. The current operation and structure goes back to the 1990s Denver University “ROAMS” OS/2 system. We retain the concept of running a list of OPUS XPMs (experiments) as a TASK file. TASKs are listed in a SCHEDULE file where they are scheduled in terms of fixed start times or SZA. Tasks and XPMs can also be manually run.

These concepts were used for a later VB6 (Windows up to XP) versions (Brukerlite and Autobruk). Moving to Windows7 required new software to be written. We chose Python.

The new Program, BrukerPy, can easily replicate the performance of previous automation plus additional functionality and a better user interface. It can be run on newer, 64 bit machines as well as 32 bit, and will hopefully have an easier upgrade path as Windows evolves.

3 Installation

3.1 Requirements

Brukerpy is a Python 2.7 program that therefore requires a Python distribution. A standalone executable was discussed but the work load in having to build the executable for different systems seemed unnecessary compared to the utility of having Python installed. Any distribution of Python can be installed, our preference is Anaconda but so long as you have access to the latest repositories it does not really matter.

Along with a standard distribution, several specific modules are required;

pyephem
pygubu
psutil

These can be installed using your distributions tools such as pip, or they can be installed manually from the versions supplied with BrukerPy. This is done by;

pip install <ephem wheel file>

For a 32 bit system the ephem install command might look like;

pip install ephem-3.7.6.0-cp27-none-win32.whl

Before installing BrukerPy itself, it would be wise to check the functionality of these libraries, at least trying to import them in to a Python session (not that the actual libraries are called *ephem*, *pygubu* and *psutil*). Now we can begin the installation

3.2 Installation and INI File

Copy the BrukerPy folder to your directory of choice, ideally the top level (C:/). You will now need to edit the ini file to suit your site and system, remembering that absolute paths are required. Here is an example of the ini file contents and their meanings;

[paths/files]	These are the paths you will need to change on install
datapath=C:\data\	Where you will store output data
macrofilespath=C:\brukerpy_v2.0\macro\	Where the OPUS macro is kept, shouldn't change other than the location of BrukerPy

taskspath= C:\brukerpy_v2.0\task\	Folder containing your tasks
schedulespath= C:\brukerpy_v2.0\schedule\	Folder containing your schedule
xpmpath= C:\brukerpy_v2.0\xpm\	Folder containing your OPUS XPMs
OPUSexepath=C:\OPUS_7.2.139.1294\	Path to the OPUS executable
python_log_path=C:\data\	Where the log from Python will be written
macrofile=runxpm3.mtx	Name of the macro, you won't need to change this
taskfile=	Empty
schedulefile=nir.txt	Schedule to be run
databasefile=task_database.txt	Dynamic schedule task database to be run if used
xpmfile=	Empty
OPUSexefile=OPUS.exe	OPUS executable name
simulator_xpm=simulator.xpm	Advanced simulator XPM, see section 6.2.7
[startup]	Default start up modes, 1 true, 0 false
run_daily_mode=0	These are the suggested initial values and are discussed later (6.2)
dynamic_schedule_mode=0	
schedule_status=0	
legacy_format_mode=0	
use_aux_flag_mode=0	
gui_test_mode=1	

simulator_mode=0	
[site]	Self-explanatory, edit at will
sitename=NIWA, Lauder NZ	
site_id=LA	
inst_id=01	
latitude=-45.04	
longe=169.68	
timezone=12	
pressure=1013.25	
temperature=12.1	
gui_test_xpm_time=300	
dynamic_margin_time=30	Margin time for dynamic mode (7.4.2)
[aux_properties]	Keep gather_aux at 0 until you are ready to use the auxiliary functions described later (4.6)
gather_aux=0	
labels= ["Time","Left","Right","Up","Down","Condition"]	
time_period=60	

Once finished it is suggested that you edit the last few lines of the BrukerPy main script. Comment out the following lines

sys.stdout= open(log_path+"\\gui_log.dat","w")
sys.stderr=sys.stdout

This is simply so that when you run *brukerpy_main.py* for the first time, from a console or spyder session, you will have the full stdout available straight away. This helps significantly when setting the *ini file* up for the first time. You should now be up and running in GUI test mode and should be able to run tasks and schedules provided they have been created. We will discuss tasks and schedules in more detail once we have described the code in more detail. In case you are unfamiliar with Python, here is a quick summary of how to run a script provided you've navigated to the BrukerPy directory in each case;

From command terminal;
Path/to/python.exe *brukerpy_main.py*

Or if your system variables have been updated just;
python *brukerpy_main.py*

From Python terminal;
execfile("brukerpy_main.py")

In Spyder;
Open *brukerpy_main.py* and press the green run button (F5)

3.3 Shortcut with Icon

The last thing left to do is to create a shortcut with the incredible icon.

On the desktop right click -> new -> shortcut

At the prompt, browse to and select brukerpy_main.py from your BrukerPy folder and at the next prompt, give the shortcut an appropriate name

Right click on the shortcut -> properties -> shortcut tab -> Change Icon...

Browse to and select the icon file from the BrukerPy folder, click ok, and apply. Once again in the shortcut tab, in the Target path enter your Python installation directory with Pythonw.exe instead of *Python.exe* (it is also sufficient if your Python path variable has been set up correctly just to enter *Pythonw* or *Python*) i.e.;

C:\Users\geddesag\AppData\Local\Continuum\Anaconda2\pythonw.exe
C:\Users\geddesag\Desktop\brukerpy_v1.5\brukerpy_main.py

This is the same as in the command line entering Python brukerpy_main.py. Pythonw is windowless Python, no additional terminal will be opened with Python. This is useful once you have ironed out all the settings and do not want to view the live *stdout* and *stderr*.

4 Structure

Now that we're up and running, let's pause and have a look at the overall structure of the code. BrukerPy is designed to be modular with functions being interchangeable. This is so that we can for instance, move to a http link instead of dde without significantly impacting the code. To this end BrukerPy consists of five scripts, an initialisation file (*ini file*) and a *ui file*. These are;

4.1 brukerpy.ini

Contains the default settings for the system, such as latitude, longitude, timezone, surface pressure, temperature and file structure. This should be edited upon installation to reflect the site, as described in the installation.

4.2 library.py

This is pretty self-explanatory, it contains a number of functions that are useful in the main operation. Most notably this library contains the functions that calculate the times at which tasks or XPMs should be submitted based on solar zenith angle. This requires the use of pyephem, which is therefore a requirement.

4.3 dde_client.py

This is a custom library that is built using ctypes to drive a dde link to another program. This should therefore be quite robust and removes any protracted installation procedures. Because the module is self-contained, it can easily be replaced as previously discussed.

4.4 exec_xpm.py

This is a standalone script that runs an experiment on OPUS. It can be integrated as a function but the primary reason for complete separation is the memory allocation of 32 bit Python is not ideal. Running this script as an entirely different process should be more stable on older platforms.

4.5 brukerpy_main.py

This is the top-level code that runs the GUI and controls job submission. Its overall flow is as follows: The other scripts and necessary Python libraries are imported and the GUI is built as an App. After the layout of the script is completed, the default schedule is calculated in a threaded process, after what has completed, all the daily data is displayed. Once the threaded process has been submitted the main loop of the GUI is initialised. The ins and outs of the main loop are described later.

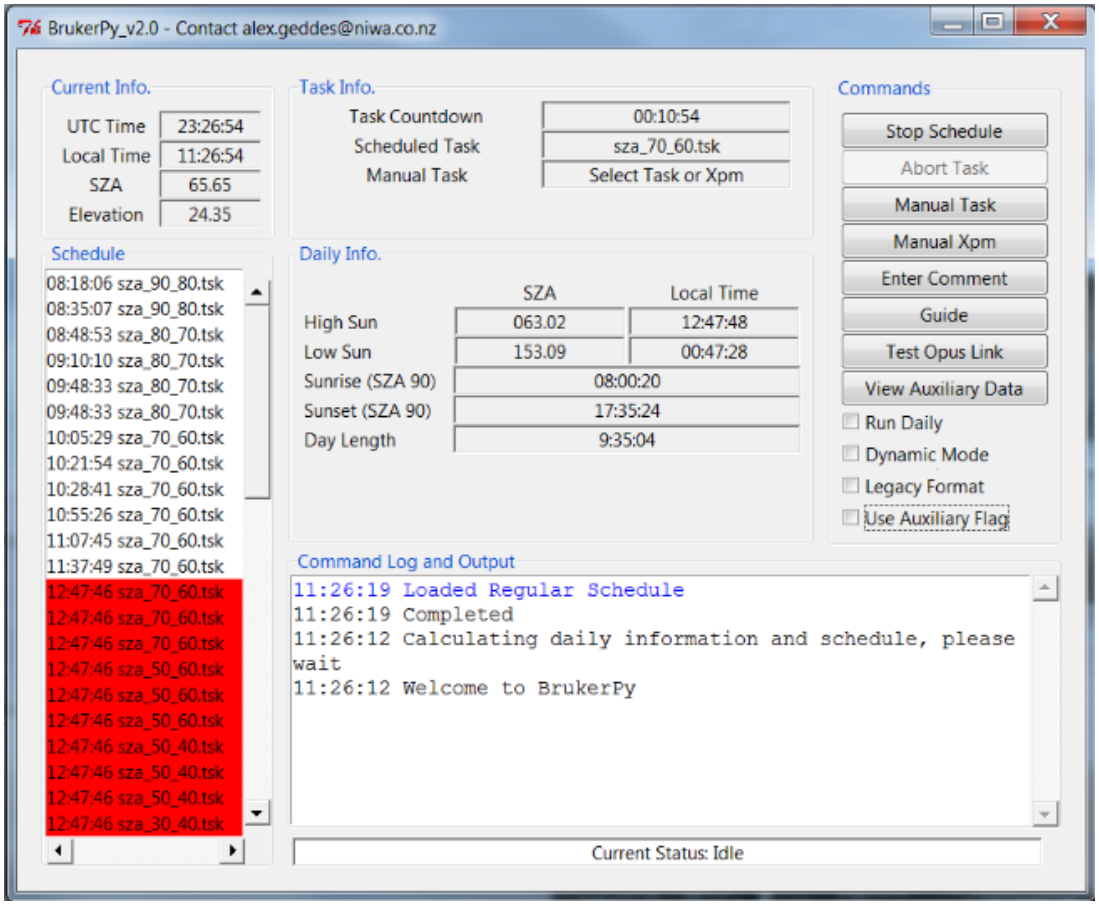
4.6 aux_data.py

This is a special script and is one that you are required to edit should you wish to incorporate any other data streams in to the operation of BrukerPy. A function is defined here called *gather_auxiliary_data* that when called returns a list of values and an integer flag value (1 being flag raised, do not run any tasks and 0 being all is well and BrukerPy can run tasks). This is further outlined in 7.6.

4.7 brukerpy.ui

This is a *pygubu* user interface file that defines the layout of the GUI. It is far simpler to use than by using *tkinter* directly. If the user wishes to adjust the layout of the GUI to suit their needs, open *pygubu* (in our case it is *C:/Anaconda/Scripts/pygubu-designer.bat*) and open the *brukerpy.ui*. Now you may adjust the layout of the objects but you must make sure that none are removed and that they maintain their id's. Make a backup of the ui file just in case.

5 Display



Here is the layout of the GUI on startup

5.1 Current Info.

This panel shows the current UTC time and local time. These are based on the machine time (local) and the timezone specified in the *ini file*. It is therefore important to maintain the machine time on your regions standard time, do not change to summer time for instance. The solar zenith angle (SZA) and solar elevation are also shown. These again rely on the location and timezone information in the *ini file*.

5.2 Task Info.

If the schedule is running, a countdown until the next task (or task window, discussed later) is shown along with the upcoming scheduled task. If the schedule is not running, the countdown is not shown. Lastly there is the manual task, which asks for a task or XPM to be selected. Once selected (later) the manually run task will be shown here.

5.3 Daily Info.

When BrukerPy is started, the daily information is calculated along with the schedule and useful information is displayed. This process can take a few moments so it is threaded out and a message is displayed to the user, hence the entry 'Calculating daily information and schedule, please wait'. The GUI still operates during this time but this has been added to increase clarity. The pieces of information are straight forward. The high and low sun SZA/time are displayed, obviously, the high sun is more relevant but the low sun is useful for polar sites. In addition, we show the sunrise and sunset times assuming that the SZA is 90 degrees. Lastly, we also show the hours of day light, again defined as the time between 90 degrees and 90 degrees (or low sun if applicable).

5.4 Schedule

The scheduled tasks with their relevant times calculated from the required SZA or as a time specific task. They are also colour coded;

- White – Scheduled
- Red – Cancelled due to invalid SZA or time, for instance scheduling a task to run at a SZA that doesn't occur, or a time that is before the previous task. This is an important point to consider when building schedules, SZA should be in order and time-specific tasks placed with caution.
- Orange – Skipped due to a task already being active
- Green – Running
- Blue – Completed

Clicking on one of the entries will display this information and additional facts such as start time and duration or counts (if relevant) in the output window.

5.5 Output

Text output displaying useful events such as when experiments start and finish, most recent line is highlighted blue. It also displays the current status of the gui, such as waiting for job, running scheduled task, running manual task etc.

6 Controlling BrukerPy

As there is quite a lot to discuss, the control panel gets its own section

6.1 The Control Panel

6.1.1 Start / Stop Schedule

Begins running through the schedule. If the schedule type has been changed, it will be reloaded. The schedule itself is determined in the *ini file*. The initial state can be set in the *ini file*, see section 6.2

6.1.2 Abort Task

This is only available once a task has been started. It aborts the remaining experiments within the current tasks**. Note it will not abort the currently active XPM on OPUS**. The user must wait until the live experiment has finished as telling OPUS to abort often crashes OPUS.

6.1.3 Manual Task and Xpm

Opens a browser to the default task or XPM directory. The task or XPM is then run once it has been selected. Note you can choose either an XPM or task with either button, just the directory is different.

6.1.4 Enter Comment

This opens a new window for the user to write a comment that will appear in the log file as well as in the output window

6.1.5 Test OPUS link

This tests that the dde link to OPUS is working. It will return the instrument being used by OPUS. This works regardless of whether an experiment is being run as the XPM is a separate process entirely. In GUI test mode you will be rewarded with a reminder that you are in GUI test mode.

6.1.6 Guide

A summary of the GUI and the controls is displayed in a new window

6.1.7 View Auxiliary Data

This opens a new window that displays the latest available auxiliary data, it will refresh automatically at the ini specified rate or can be updated by pressing refresh data. This is disabled if auxiliary data is disabled in the ini.

6.1.8 Run Daily Check Box

Enabling this means that BrukerPy will reload the schedule at 1am and then begin taking measurements accordingly. This allows for continuous automation which may or may not be desirable, hence the check box. If unchecked, the schedule will still be reloaded at 1am, but the GUI will wait for the command to begin.

6.1.9 Dynamic Mode Check Box

This changes the scheduling mode to a more dynamic, observation dense, mode. It requires the use of the *task_database* file rather than the *schedule file* and will be discussed more fully later

6.1.10 Legacy Format Check Box

When ticked the output file format from OPUS in the style adopted by Dan Smale at Lauder, which uses the year represented by a letter of the alphabet relative to a specific year. This will clearly be redundant in several year's time (R is 2017). When unticked, the format is AABB_XXX_YYYYMMDDHH.Z where AA and BB are the site and instrument identifiers as noted in the *ini file*, XXX is the first three characters of the XPM name YYYY,MM,DD and HH are the year month day hour that the measurement was made and Z is the repeat measurement suffix.

6.1.11 Use Auxiliary Flag Check Box

If set, the flag provided by the *gather_aux_data* function will be used to halt the running of tasks. If not set, auxiliary data will still be gather but the flag not used.

6.2 The Ini File

The Initial values of various parameters can be set in the *ini file* in the startup section;

6.2.1 run_daily_mode

If equal to 1, the *run daily check box* is active upon start and the program will refresh at 1am as described above and continue to run. Set equal to 0 to disable

6.2.2 dynamic_schedule_mode

If equal to 1 the *dynamic mode check box* will be set to true upon start and the dynamic schedule will be loaded as above. Otherwise set to 0 to be in regular operation at start up.

6.2.3 legacy_format_mode

If equal to 1 the *legacy format mode check box* is set to true and the output files will have the legacy file format applied to them, set to 0 for the startup mode to be standard form.

6.2.4 use_aux_flag_mode

If equal to 1 the use *auxiliary flag* check box is set to true at startup, if 0 it is set to false.

6.2.5 schedule_status

If set to 1, the schedule will begin running automatically once started, if set to 0, it will wait to be started with the Stop/Start Schedule button

6.2.6 gui_test_mode

If set to 1, the GUI will not send commands to OPUS and will instead run in a dummy mode for testing purposes. It will still run schedules, tasks and XPMs but these are place holders with a sleep command in them that will generate an output file after the *gui_test_xpm_time* (seconds)

specified in the *ini file*. This is an immensely useful tool for getting things going. Otherwise, set this value to 0

6.2.7 simulator_mode

This is a more advanced development tool that has not been thoroughly tested on different OPUS versions. It allows the user to test the GUI with OPUS but without a physical instrument. The user must go into OPUS and change the optical bench to the simulator. Then, with the *simulator.xpm*, run XPMs or tasks as normal. This is best left set to 0 and the user should contact the author if attempting.

7 Principles of Operation

Here we will go over some of the key operations and decisions the GUI makes and the various operating modes.

7.1 Solar Zenith Calculation

The SZA calculation underpins the scheduling process so it's important to understand its workings. An open source module called *pyephem* is used to calculate the position of the Sun based on time, location, surface pressure and temperature. These last two variables are worth noting, the module does indeed take into account atmospheric refraction if pressure is specified. This is why estimates of surface pressure and temperature are included in the *ini file*. Another key consideration when using this module on its own is the dependence on UTC time, you must specify the time in terms of UTC and then correct it afterwards. This code is also easily adaptable to output the lunar zenith angle and may well be a future option in the GUI.

http://rhodesmill.org/pyephem/

7.2 Communicating with Opus via a DDE Link

Communication with OPUS is achieved with a dynamic data exchange (DDE) link. This is a interprocess communication tool available on windows machines that dates back to the release of Windows 2.0 in 1987. It is an easy to learn and reliable method of communication between programs. In the case of OPUS, it allows us to run text based commands in OPUS and return a result. This allows us to check for instance the version of OPUS and the instrument identification, as used in the *Test Opus Link* command. In running experiments, we use the same principles to run a OPUS macro which we then populate with additional text sent with the same DDE link.

As far as we are aware, Bruker will continue to support DDE communication to OPUS and as such so will we. However the drawback on of our dependence on it is that we are tied to Windows operating systems. When a Linux version of OPUS is widely used we will need to develop an additional communication method, most likely http which is the other main means

of communication with OPUS. Given that we have made BrukerPy modular, it should only mean that we change the *exec_xpm.py* file and so should be a smooth transition.

7.3 Tasks

A task is a series of XPMs to be run by OPUS successively. This is achieved by creating a task file beforehand which BrukerPy will then read in and submit the XPMs sequentially when the task is run. A typical task file is shown below;

```
sza_20_30.tsk
For new tracker mirrors 2011..
7       Number of XPMS to run
lmshb163.xpm lm
l5shb143.xpm l5
l2shb123.xpm l2
l3shb033.xpm l3
l4shb043.xpm l4
lnshb073.xpm ln
lcsmb14a.xpm lc
```

After the 3 lines of header we have a list of XPMs in the first column and two characters in the second column. These are all irrelevant and are historical (and so backwards compatible with old software), only lines that contain the text ".xpm" are used and what comes after that is ignored. It is advised that you standardize your task files to the following;

```
#HEADER
#HEADER
l5shb143.xpm
l2shb123.xpm
l3shb033.xpm
l4shb043.xpm
lnshb073.xpm
lcsmb14a.xpm
```

Once a format is settled upon you could alter the *read_task* function in the library to better reflect your choice but this is not necessary if you make sure to not include ".xpm" in any non XPM lines like the header.

7.4 Scheduling

7.4.1 Normal Mode

In normal mode (non-dynamic) the code reads in the schedule file specified in the GUI and calculates the task running times based on the SZA and if it's an am or pm task (A or P in the file) or takes the observation time directly. It will then go through each task and decide if it is possible due to the SZA or if it is poorly organized time wise. Here is an example;

```
Z(T) Deg&a/p(time)
Z 87.5 A    sza_90_80.tsk
Z 68.0 A    sza_70_60.tsk
Z 67.0 A    sza_70_60.tsk
Z 57.0 A    sza_50_60.tsk
Z 55.0 A    sza_50_60.tsk
Z 25.0 A    sza_20_30.tsk
T 12:00:00  sza_50_40.tsk
T 12:30:00  sza_50_40.tsk
T 13:00:00  sza_50_40.tsk
Z 25.0 P    sza_20_30.tsk
Z 30.0 P    sza_20_30.tsk
```

The first column denotes which task type, Z or T, meaning zenith angle or time. Where zenith angle tasks are scheduled to coincide with that specific angle and time measurements are fixed in time. The second column denotes the value for this time or angle. If the measurement is zenith type, the third column indicates whether it is in the AM or PM by A or P and the last column is the name of the task to be run. A header of one row is expected, otherwise the schedule is read line by line and only reads lines that begin with T or Z. These tasks must be present in the *taskpath* folder specified in the *ini file* for them to run. This example shows the limitations of normal mode. The tasks are only run at specific times during the day, therefore in practice we will include several of the same task at slightly differing SZA's to ensure that data is taken. As a result, you end up with a long schedule file that isn't as efficient as it could be.

7.4.2 Dynamic Mode

Dynamic mode is smarter. Instead of a list of all the individual tasks to be run a shorter list of task windows are defined in the *task_database file* also specified in the *ini file*. Simply put, the scheduler ascribes a given task to a window rather than a specific SZA or time. These windows are calculated for the entire day with time windows taking precedence. Whilst within each window the task will be run as many times as possible provided there is more than a set margin of minutes left in the window, this margin is configurable in the *ini file*, at *dynamic_margin_time=minutes*. If there isn't then the next window can begin early if the nex task is not a timed window. If the task has been able to run once before in the windows, the duration it took will be used to determine if it can be run again within the windows. In all cases, tasks will not be allowed to overrun on to the next window if the next window is time specified. Here again is an example;

```
#Task Data Base
#Type( Z or T)  start_value stop_value Taskname
Z 90 80 sza_90_80.tsk
Z 80 70 sza_80_70.tsk
Z 70 60 sza_70_60.tsk
Z 60 50 sza_50_60.tsk
Z 50 40 sza_50_40.tsk
Z 40 30 sza_30_40.tsk
Z 30 20 sza_20_30.tsk
T 12:00:00 15:00:00 sza_50_40.tsk
T 15:00:00 16:00:00 ooofti_run.tsk
```

Again we have the Z or T notation in the first column. The second and third column are the upper and lower limits of each window in either SZA or Time space. The last column is the task to be run within the window. Note that there is no AM or PM at this point. The dynamic schedule assumes that you will want to run all SZA tasks in both the AM and PM which is often the case. Do not specify SZA's higher than 90, and if you want to run a task all day, specify the SZA limits to be 90 and 0. Two header rows are expected.

7.5 Experiment / Task Execution

When the countdown hits zero or the manual task/XPM is selected a thread is started that handles the process starting and monitoring required. The thread will then spawn an independent process.

- Thread – It starts the process and monitors it until completion, at which point it will start a new process for the next XPM or finish.
- Process – This submits the macro to the OPUS via dde with the correct input for the given XPM. It then polls OPUS for the completion of this XPM. Polling is done because the file building done by OPUS is inconsistent, the file can be created at the start or end of the XPM and its size varies throughout. The polling is done through the DDE by querying the macro id that was issued upon execution.

This is deliberately excessive and was done due to hardware limitations. The PC used (Windows XP, 32 Bit) and 120M used during the development phase were outdated and did not perform consistently. This meant that separating the GUI as much as possible from the XPM running process was essential. On more modern machines and instruments a lot of the divorcing that was done here should be irrelevant. In which case the whole procedure of submitting a macro to OPUS and polling its status can be done within a single thread from the main program. This was previously done and is easily doable. However, it is unnecessary and BrukerPy runs smoothly on other machines as is.

7.6 Auxiliary Data

A significant advancement is the ability to use auxiliary data in the decision-making process of the GUI. For instance the user can add in solar tracker information, i.e. if the tracker is locked, and run tasks accordingly. As mentioned earlier in section 4.6, the aux_data.py file contains a

single function, *gather_auxiliary_data* that the user must customise to return a list and a flag after which the user must set up the labels of the returned data in the ini file in the *aux_properties* subsection and specify the *time_period* (in seconds) at which to take measurements.

The use of the flag can be disabled at will as described in section 6.1 and the user can disable all auxiliary functions by setting *gather_aux=0*. This is recommended for initial set up. Here are some examples of what the *gather_auxiliary_data* function might look like;

```
def gather_auxiliary_data():
    """Takes the most recent line from a txt file, tail is defined in the library, it extracts the last
    line"""
    data=array(tail("aux_data.dat").split(),dtype=float)
    if data[0]>1020:
        suspend_flag=1
    else:
        suspend_flag=0
    return data,suspend_flag
```

or slightly more advanced;

```
def gather_auxiliary_data():
    “””Extract data from the last line of a data file hosted on a server”””
    now=datetime.datetime.now()
    year=str(now.year)
    month="%02d" % now.month
    day="%02d" % now.day
    todayspath=year+month+day
    f=urllib.urlopen("http://10.10.0.100:8080/logs/"+todayspath+"/solar.txt").read()
    lastline=(f.split("\r\n")[-2]).split()
    if lastline[-1]=="locked":
            flag=0
    else:
            flag=1
    return lastline[:],flag
```

There are some other basic examples and dummy versions within the *aux_data.py* for inspiration. Reading the last line of a file is perhaps the simplest way to proceed. All auxiliary data should be being logged separately by its own program, so accessing this data should be straight forward. Brukerpy makes no attempt to save or store any of this data as it is merely interpreting the already gathered and recorded data.

8 BrukerPy Output

After running BrukerPy you will have a directory for the day within your *datapath* directory (from the *ini file*) encoded YYYYMMDD. Within this folder you may have a number of files depending on how many XPMs have been run and whether or not BrukerPy has refreshed at midnight. In addition to any OPUS files generated by OPUS, these files are;

8.1 gui_log.dat and exec_log.dat

These two log files are the stdout and stderr of the *brukerpy_main.py* process and the *exec_xpm.py* processes respectively. They are copied to this directory at 1am otherwise they will be in the *datapath* directory. You can freely ignore these, they are used to diagnose faults and bugs within the code. However, if something goes wrong it is important to save them. They will also not be overwritten as they are in append mode, again aiding their usefulness in debugging the program.

8.2 yyyymmdd.log

This is the log file produced by BrukerPy, an example of which is shown below;

** Clear AM, thin high cloud about PM, BrukerPy, 10cm cell **
08:35:58 Welcome to BrukerPy
08:35:58 Calculating daily information and schedule, please wait
Dynamic Schedule Mode
Log file for 20170803 at NIWA, Lauder NZ
Site Info: latitude -45.04, Longitude (EAST) 169.68, Timezone 12
Daily Info: Sunrise 07:57:50, Sunset 17:37:38, Day Length 9:39:48, High Sun 62.5065419868 @ 12:47:41

Todays Schedule and Log
0 07:57:50 nir.tsk Scheduled
1 17:37:38 Complete Scheduled

Console output and User Comment
08:36:01 Completed
08:36:01 Loaded Dynamic Schedule
08:36:08 Schedule Started
08:36:08 Running Job - nir.tsk
08:36:08 counts = 1.0
08:36:08 Starting Xpm - nir.xpm (1/1)
08:38:42 Xpm Complete - nir.xpm Duration 154 Secs.
08:38:47 Job Complete - nir.tsk Duration 159 Secs.
08:38:47 Running Job - nir.tsk
08:38:47 counts = 2.0
08:38:47 Starting Xpm - nir.xpm (1/1)
08:41:22 Xpm Complete - nir.xpm Duration 154 Secs.
08:41:27 Job Complete - nir.tsk Duration 159 Secs.
08:41:27 Running Job - nir.tsk
08:41:27 counts = 3.0

It contains the loaded schedule information along with the commands issued by BrukerPy and their completion. A space denoted by '**' is reserved at the top of the file for useful QAQC comments that are entered manually. Additionally, user comments made through BrukerPy are prefaced by '##'. Shown here is BrukerPy operating in dynamic mode with one task for the whole day, this task containing one XPM.

8.3 Sample

Here is an example directory listing for a typical day at Lauder, making Near Infrared measurements;

Name	Date modified	Type	Size
20170803.log	4/08/2017 9:11 a.m.	Text Document	39 KB
exec.log	3/08/2017 5:04 p.m.	Text Document	35 KB
gui.log	4/08/2017 1:00 a.m.	Text Document	8 KB
la06_nir_2017080308.0	3/08/2017 8:38 a.m.	0 File	22,673 KB
la06_nir_2017080308.1	3/08/2017 8:41 a.m.	1 File	22,673 KB
la06_nir_2017080308.2	3/08/2017 8:44 a.m.	2 File	22,673 KB
la06_nir_2017080308.3	3/08/2017 8:46 a.m.	3 File	22,673 KB
la06_nir_2017080308.4	3/08/2017 8:49 a.m.	4 File	22,673 KB
la06_nir_2017080308.5	3/08/2017 8:52 a.m.	5 File	22,673 KB
la06_nir_2017080308.6	3/08/2017 8:54 a.m.	6 File	22,673 KB
la06_nir_2017080308.7	3/08/2017 8:57 a.m.	7 File	22,673 KB
la06_nir_2017080308.8	3/08/2017 8:59 a.m.	8 File	22,673 KB
la06_nir_2017080309.0	3/08/2017 9:02 a.m.	0 File	22,673 KB
la06_nir_2017080309.1	3/08/2017 9:05 a.m.	1 File	22,673 KB
la06_nir_2017080309.2	3/08/2017 9:07 a.m.	2 File	22,673 KB
la06_nir_2017080309.3	3/08/2017 9:10 a.m.	3 File	22,673 KB
la06_nir_2017080309.4	3/08/2017 9:13 a.m.	4 File	22,673 KB
la06_nir_2017080309.5	3/08/2017 9:15 a.m.	5 File	22,673 KB
la06_nir_2017080309.6	3/08/2017 9:18 a.m.	6 File	22,673 KB
la06_nir_2017080309.7	3/08/2017 9:21 a.m.	7 File	22,673 KB
la06_nir_2017080309.8	3/08/2017 9:23 a.m.	8 File	22,673 KB
la06_nir_2017080309.9	3/08/2017 9:26 a.m.	9 File	22,673 KB

All three log files are present as well as OPUS files corresponding to each XPM that was successfully run. These files can then be reopened in OPUS or with other tools (such as NIWAs Python Opus file handler program) for further inspection before being processed.

9 Basic User Guide

Assuming you are now set up, all your task files are to your liking and the schedule file is complete, here is a quick guide to the day to day operation of BrukerPy, ideal for newcomers who won't have to know the fine detail.

9.1 Getting Started

- Is the PC clock correct? Always local time, never adjusted by day light savings?
- Start OPUS by **double clicking on the OPUS icon**. After hitting enter when the introductory pop ups appear you should be left with the regular OPUS screen.
- Check the instrument ID by navigating to the optics setup and service tab
- Launch BrukerPy double clicking on the BrukerPy icon. Wait while it loads up
- Hit Test OPUS link, it should return the same instrument name as is shown in OPUS
- Good to go

9.2 Closing down

- Wait for the XPMs to finish in OPUS.
- Close BrukerPy by clicking the cross in the corner, nothing fancy
- Likewise for OPUS

9.3 Run a Schedule

Once BrukerPy has loaded, make sure that the dynamic mode check box is what you want it to be. Then, if the schedule is not already running, click the Start Schedule button. A countdown should then appear in the task countdown box. Feel free to walk away but it is a good idea to check that a task physically runs when the countdown is reached. You do this by checking to see if OPUS us doing anything (There should be a green bar at the bottom of the screen with text in) with no additional windows whilst brukerpy says it is running an XPM.

9.4 Run a Manual Task or XPM

Firstly, you must wait until any running tasks or XPMs have finished. You may also abandon a task with the Abort Task button but you must still wait for the current XPM to finish. Once ready, the Manual XPM and Task buttons will become active. Click on one of them and you'll be directed to the default XPM and task directories. Select the task or XPM you wish to run and hit OK. The task or XPM will now be executed. You'll be able to run an additional task or XPM once this one has finished, and do not worry if you accidently press one of these buttons, just close the directory window.

9.5 Leave a Comment

Simply press the comment button and type in your comment before hitting enter.

9.6 FAQ and Troubleshooting

9.6.1 Brukerpy says it is still running an XPM but OPUS isn't doing anything

Tricky fault to debug remotely, usually caused by orphaned processes. copy the *gui_log.dat* and *exec_log.dat* from the *Python_log_path* directory in the ini to the author as well as the daily log stored in the *data path/todaysdate folder*. It should in the mean-time be fixed by restarting BrukerPy.

9.6.2 When running an XPM, a window appears in OPUS and the macro fails to load

This is most likely due to a problem with your *xpmpath* in the ini file and the XPM names in your task file. Ensure that they are the absolute path and that the XPM in question exists in that directory.

9.6.3 The schedule has started a task but I want to do a manual measurement

No problem, if you cannot wait for the task to finish, press the abort task button and wait for the current XPM to finish. Once it has done so, begin your manual measurement. You may also stop the schedule so that this doesn't happen again by pressing the button. But make sure you restart it once you are finished.