JUMPING JIVE

Project ID: 730884

pySCHED: re-factoring SCHED from Fortran to Python

Deliverable:	D8.3
Lead beneficiary:	JIVE (Authour: Bob Eldering)
Submission date:	21-11-2019
Dissemination level:	Public

Contents

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 730884

1 Introduction

The program SCHED[1] was written in the early 1980s in order to provide a common, generalised user interface for scheduling VLBI observations. It does so by combining observing parameters, source catalogues and frequency setup catalogues, which describe the detailed settings at all different stations. This is by no means trivial, considering that all telescopes are different, in terms of location, architecture, hardware limitations, equipment and frequency coverage. The resulting schedule comes in the form of a so-called VEX file, for which an international standard was defined, a plain-text human-readable equivalent of a database. This file is sent to the stations, where the control computer parses the schedule and translates it into a series of commands to the telescope control system and the recording/transmitting equipment. The decades-old code base makes the program extremely hard to modify in order to adapt it to the modern-day demands of VLBI networks.

This document describes how we re-factored the SCHED code to be more easily maintainable (Section 2) and how we used the re-factored code base to add new features (Section 3). We have called the resulting program pySCHED.

2 Re-factoring

SCHED is written in Fortran 77, an old language with which few programmers or astronomers are familiar these days. Fortran 77 also lacks many features found in modern high-level languages, most importantly data structures. For these reasons we have decided to switch to a modern language for pySCHED. Out of the modern languages, we have chosen Python to be used in this project. Its broad acceptance in the astronomical community, extensive scientific libraries and ease of use make it a natural choice.

Rather than re-writing everything from scratch in Python, we have replaced specific bits of the Fortran code of SCHED. We have done this by calling the original Fortran code from within Python. We use a program called f2py[2] to assist us. F2py takes Fortran code and generates a Python extension module. This Python module exposes the Fortran functions as Python functions and the Fortran COMMON blocks as Python classes. Using f2py makes it possible to call SCHED functions from within Python code and replace specific bits of code while maintaining backwards compatibility with SCHED. This makes the transi-

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 730884

tion from using SCHED to using pySCHED easier.

2.1 Keyin reader

The SCHED distribution includes a set of catalogues of source positions and station configuration information, which are specified using the Keyin format. The specification of the Keyin format is informal, in practice the precise specification is the interpretation that SCHED gives a Keyin file when it is read. Fortran is not a language well suited to parsing and string handling, and parsers developed in Fortran can be hard to maintain and often (as here) do not follow a precise formal specification.

Therefore we have replaced the parser with a version written fully in Python. The parsing is done by recursive descent, while tokenising is done by regular expressions. This process is a lot easier to read and maintain than the state-machine process of the parser written in Fortran.

The output of the Python parser is in the JSON format[3]. Parsing large Keyin files (the source catalogue) can be rather slow. Therefore we store the JSON output of the parsing in a cache. Very fast parsers for JSON are readily available in Python.

2.2 COMMON blocks to Python classes

As said before, Fortran 77 does not support proper data structures. Where in a modern programming language one would use an array of structures containing data elements, SCHED uses a distinct array per data element. The grouping of these data elements is expressed in SCHED by using the same indexing into the corresponding arrays. These groupings match the SCHED catalogues closely. We have made Python catalogue classes, which make this grouping explicit. These catalogue classes all share methods to read and write from/to the COMMON blocks as exposed by f2py. This explicit grouping of data elements makes code that accesses these elements in Python easier to maintain. However, any time we call a Fortran function, we need to synchronise the Python catalogue classes with the Fortran COMMON blocks. When we call many Fortran functions in quick succession from within the Python code, this synchronisation can take up a lot of time. Therefore we have also created a method to access the COMMON blocks directly through the catalogue classes. Using this method is slightly slower per access to the data elements, but it eliminates the need to synchronise the catalogue objects when calling Fortran functions.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 730884

2.3 Translation of SCHED functions

Using f2py allows us to call Fortran functions from Python code, but we do not modify the SCHED Fortran code to call Python code. This means that when we want to replace a bit of Fortran code, we also need to translate all functions that call that bit of code. In the course of these developments, Fortran functions representing approximately 63 thousand lines of code have been translated to Python, while Fortran functions that are called from Python represent approximately 223 thousand lines of code.

3 New features

Using the re-factoring and transition to Python, described in the previous section, we have added many features to pySCHED. These features are described in this section.

3.1 Automatic catalogue updates

Since the catalogues are included in the distribution of SCHED, updating them can be a slow process. In scheduling the European VLBI Network[4] (EVN) experiments, the work-around for this problem has been to email the principal investigator (PI) up-to-date catalogue files. Obviously, this has the disadvantages that these updates are not available for the users in general and only available for the PI when the time to schedule the experiment has come.

PySCHED is still distributed with a set of catalogues, but it also downloads updates to the catalogues on start-up. This update process uses the git version control software[5], so the amount of data transferred to download updates is minimized. In case no internet access is available, the update process will simply be skipped. So pySCHED still functions normally without internet access. Updates to the catalogues can be shared with all pySCHED users quickly this way.

3.2 DBBC 2 support

The main motivation to start work on pySCHED was the lack of support for the digital base band converter (DBBC) version 2 in SCHED. The consequence of this lack of support is that the intermediate frequency (IF) parameters have to be specified manually for each station. PySCHED has full support for the the DBBC

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 730884

2 and can determine the IF parameters from the frequency catalogue entries. However, since this work-around for SCHED has been in effect for a couple of years, the frequency catalogues had never been updated for the new DBBC 2. We have taken the manual input of IF parameters for the last year of EVN experiments and computed the corresponding frequency catalogue entries. This catalogue update is distributed to the users through the automatic method described above.

3.3 VEX 2

VEX (VLBI EXperiment) is a standard for expressing VLBI experiment schedules. Schedules in VEX format are used by most VLBI telescopes. SCHED can create VEX schedules and this is its default output format. The current version of the standard is 1.5[6], which has been in use since the late 1990s.

An update to the standard is in progress and nearing completion[7]. This new version will be released as version 2. The main goal of this update is to support a wider range of VLBI data acquisition hardware as version 1.5 is not capable of properly describing the current generation of VLBI equipment. This is achieved by the introduction of a much more flexible description of equipment and how this equipment is connected together. This makes it much easier to specify schedules for data acquisition systems that consist of multiple parallel (digital) back-ends and/or recorders. But VEX 2 includes many additional improvements such as support for complex sampling, observation of spacecraft, specification of intent, etc.

PySCHED writes both a VEX 2 and VEX 1.5 file. The code generating the VEX files is written from scratch in Python. However, the Fortran code to write VEX 1.5 files is still included in pySCHED, because some users might depend on comments and specific formatting of the VEX file to encode information that could not be otherwise encoded in VEX 1.5. Most of these problems are solved in VEX 2.

3.4 Experiment intent for scans

Usually, during real-time correlation, multiple experiments are combined into one observing run. These experiments are combined during the scheduling phase and split manually during post-processing of the correlated data. We want to automate this process using the intent feature of VEX 2.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 730884

Therefore in pySCHED we have added a Keyin keyword: `SCANEXPS`. Using this keyword, the scheduler can assign a scan to one or more experiments (multiple experiments are possible when a calibrator scan is shared between two consecutive experiments). This will be reflected in the VEX 2 output by the intent keyword for scans. Using such a VEX file, the post-processing software can now automatically split the correlator output data into experiments.

3.5 TSCAL in frequency catalogue

The station catalogue has a keyword, `TSCAL`, this lets (py)SCHED know when the station measures system temperatures. The options for this keyword are “gap” and “cont” that indicate the system temperature measurements, or at least cal measurements, are done in the gap between scans or continuously during observing.

However for some EVN stations, the system temperature measurements method depends on the receiver installed in the telescope. Therefore pySCHED has added the same keyword to the frequency catalogue, which overwrites the station level `TSCAL` value when specified. This prevents the need to use a different station catalogue file depending on the receiver used.

3.6 Interactive plots

SCHED uses PGPLOT[8] to create both plots and the graphical user interface (GUI) to initiate these plots. Installing PGPLOT and linking it to SCHED can be quite a hassle, so for pySCHED we use modern GUI (Qt[9]) and plotting (Matplotlib[10]) libraries. This results in a more easily usable GUI to initiate the plots and more interactive plots.

3.7 Easy installation

Installing SCHED involves picking a make file from a selection of make files made available for different types of systems. In some cases no make file completely matches your system and some manual editing of the best match is required. PySCHED is registered as a PyPI[11] project. Which makes installing pySCHED an easy one-liner. For users who prefer to install Python libraries in separated work spaces, we profile a conda[12] environment file.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 730884

3.8 Command line arguments and readline

The typical usage of SCHED is to create a file with the input schedule and either run SCHED with input redirection from this file or run SCHED and use the `SCHEDULE` keyword to point to the input file. The second method requires typing the input file name on every start of SCHED, but it is the only way to make use of the "restart" feature available in the plotting control window. Additional commands have to be either, depending on the run method used, entered into the input file or added as keywords on the SCHED input.

In pySCHED we have added two features that make these basic interactions more convenient. The first is the addition of command line arguments, for example, to enable plotting or to set the input file name. The usage of old style VEX 1.5 printing (as discussed in Section 3.3) is also controlled with a command line argument. The second feature is the usage of the readline module. This module allows users to edit the pySCHED commands as they are typed in using for example the cursor keys. The readline module also comes with history facilities, the input history is saved between runs of pySCHED.

3.9 Programmable input

A more advanced way to provide input to pySCHED is to use the template command line argument. In this case the input to pySCHED is first processed by the template engine provided by the bottle[13] module. This allows the user to mix Keyin format input with Python code, such that complex schedules can be expressed succinctly.

4 Resources

The source code of pySCHED is available in multiple formats on GitHub:

- As zip file: `https://github.com/jive-vlbi/sched/archive/v1.3.0.zip`
- As gzipped tarball: `https://github.com/jive-vlbi/sched/archive/v1.3.0.tar.gz`
- The complete git tree: `https://github.com/jive-vlbi/sched.git`

Within the source code, the newly written Python code lives in the directory `src/pysched/`.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 730884

The installation and usage instructions are described in: https://github.com/jive-vlbi/sched/blob/v1.3.0/README.md

References

[1] http://www.aoc.nrao.edu/~cwalker/sched/sched.html

[2] https://docs.scipy.org/doc/numpy/f2py/

[3] http://www.json.org/

[4] https://www.evlbi.org/

[5] https://git-scm.com/

[6] http://www.vlbi.org/vex/

[7] https://safe.nrao.edu/wiki/bin/view/VLBA/Vex2

[8] http://www.astro.caltech.edu/~tjp/pgplot/

[9] https://www.qt.io/

[10] https://matplotlib.org/

[11] https://pypi.org/

[12] https://www.anaconda.com/

[13] https://bottlepy.org/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 730884