PyQSO Documentation

Release 1.1.0

Christian Thomas Jacobs, M0UOS

Apr 02, 2018

Contents

Contents:

CHAPTER 1

Introduction

1.1 Overview

PyQSO is a logging tool for amateur radio operators. It provides a simple graphical interface through which users can manage information about the contacts/QSOs they make with other operators on the air. All information is stored in a light-weight SQL database. Other key features include:

- Customisable interface (e.g. only show callsign and frequency information).
- Import logs in ADIF format, and export logs in ADIF or Cabrillo format.
- Perform callsign lookups and auto-fill data fields using the qrz.com and hamqth.com online databases.
- Sort the logs by individual fields.
- Print a hard-copy of logs, or print to PDF.
- Connect to Telnet-based DX clusters.
- Progress tracker for the DXCC award.
- World map with grey line and Maidenhead grid squares.
- Filter QSOs based on callsign (e.g. only display contacts with callsigns beginning with "M6").
- Remove duplicate QSOs.
- Basic support for the Hamlib library.

The source code for PyQSO, written in Python (version 3.x), is available for download from the GitHub repository.

1.2 Data storage model

Many amateur radio operators choose to store all the contacts they ever make in a single *logbook*, whereas others keep a separate logbook for each year, for example. Each logbook may be divided up to form multiple distinct *logs*, perhaps one for casual repeater contacts, another for satellite contacts, and another for DX'ing. Finally, each log can contain

multiple *records*. PyQSO is based around this three-tier model for data storage, going from logbooks at the top to individual records at the bottom.

Rather than storing each log in a separate file, a single database can hold several logs together; in PyQSO, a database is therefore analogous to a logbook. Within a database the user can create multiple tables which are analogous to the logs. Within each table the user can create/modify/delete records which are analogous to the records in each log.

1.3 Licensing

PyQSO is free software, released under the GNU General Public License. Please see the file called `COPYING` for more information. A copyright year range of the form YYYY-ZZZZ specifies every single year from YYYY to ZZZZ inclusive (for example, 2012-2017 means 2012, 2013, 2014, 2015, 2016, 2017).

1.4 Contact

If you have any comments or questions about PyQSO, please send them via email to Christian Jacobs, M0UOS, at christian@christianjacobs.uk. Bug reports and feature requests can be made via the issue tracker.

1.5 Structure of this documentation

The structure of this documentation is as follows. The section on Getting Started provides information on the PyQSO installation process through to creating a new logbook (or opening an existing one). The Log Management section explains how to create a log in the logbook, as well as the basic operations that users can perform with existing logs, such as printing, importing/exporting logs, and sorting. The Record Management section deals with the bottom layer of the three-tier model - the creation, deletion, and modification of QSO records in a log. The Toolbox section introduces the PyQSO toolbox which contains three tools that are useful to amateur radio operators: a DX cluster, a world map, and an awards progress tracker. Finally, the Preferences section explains how users can set up Hamlib support and show/hide various fields in a log, along with several other user preferences that can be set via the Preferences dialog window. A keyboard shortcuts list is also available for reference.

CHAPTER 2

Getting started

2.1 Demonstration

The screencast below demonstrates how to install, configure and use PyQSO (focussing on version 1.0.0 only). Detailed instructions are also available in the sections that follow.

2.2 System requirements

It is recommended that users run PyQSO on the Linux operating system, since all development and testing of PyQSO takes place there.

As the name suggests, PyQSO is written primarily in the Python programming language (version 3.x). The graphical user interface has been developed using the GTK+ library through the PyGObject bindings. PyQSO also uses an SQLite embedded database to manage all the contacts an amateur radio operator makes. Users must therefore make sure that the Python interpreter is installed and that any additional software dependencies are satisfied before PyQSO can be run successfully. The list of software packages that PyQSO depends on is provided in the `README.md` file.

2.3 Installation and running

Assuming that the current working directory is PyQSO's base directory (the directory that the `Makefile` is in), PyQSO can be run without installation by issuing the following command in the terminal:

```
python3 bin/pyqso
```

If the `pip3` package manager is available on your system then PyQSO can be installed system-wide using:

```
sudo make install
```

Once installed, the following command will run PyQSO:

```
pyqso
```

2.3.1 Command-line options

There are several options available when executing PyQSO from the command-line.

Open a specified logbook file

In addition to being able to open a new or existing logbook through the graphical interface, users can also specify a logbook file to open at the command line with the `-l` or `--logbook` option. For example, to open a logbook file called `mylogbook.db`, use the following command:

```
pyqso --logbook /path/to/mylogbook.db
```

If the file does not already exist, PyQSO will create it.

Debugging mode

Running PyQSO with the `-d` or `--debug` flag enables the debugging mode:

```
pyqso --debug
```

All debugging-related messages are written to a file called `pyqso.debug`, located in the current working directory.

2.4 Creating and opening a logbook

A PyQSO-based logbook is essentially an SQL database. To create a new database/logbook file, click `Create a New Logbook...` in the `Logbook` menu, choose the directory where you want the file to be saved, and enter the file's name (e.g. `my_new_logbook.db`). The new logbook will then be opened automatically. If you would like to open an *existing* logbook file, click `Open an Existing Logbook...` in the `Logbook` menu. Note that logbook files usually have a `.db` file extension.

Once the logbook has been opened, its name will appear in the status bar. All logs in the logbook will be opened automatically, and the interface will look something like the one shown in *figure:logbook*.

2.5 Closing a logbook

A logbook can be closed by clicking the `Close Logbook` button in the toolbar, or by clicking `Close Logbook` in the `Logbook` menu.

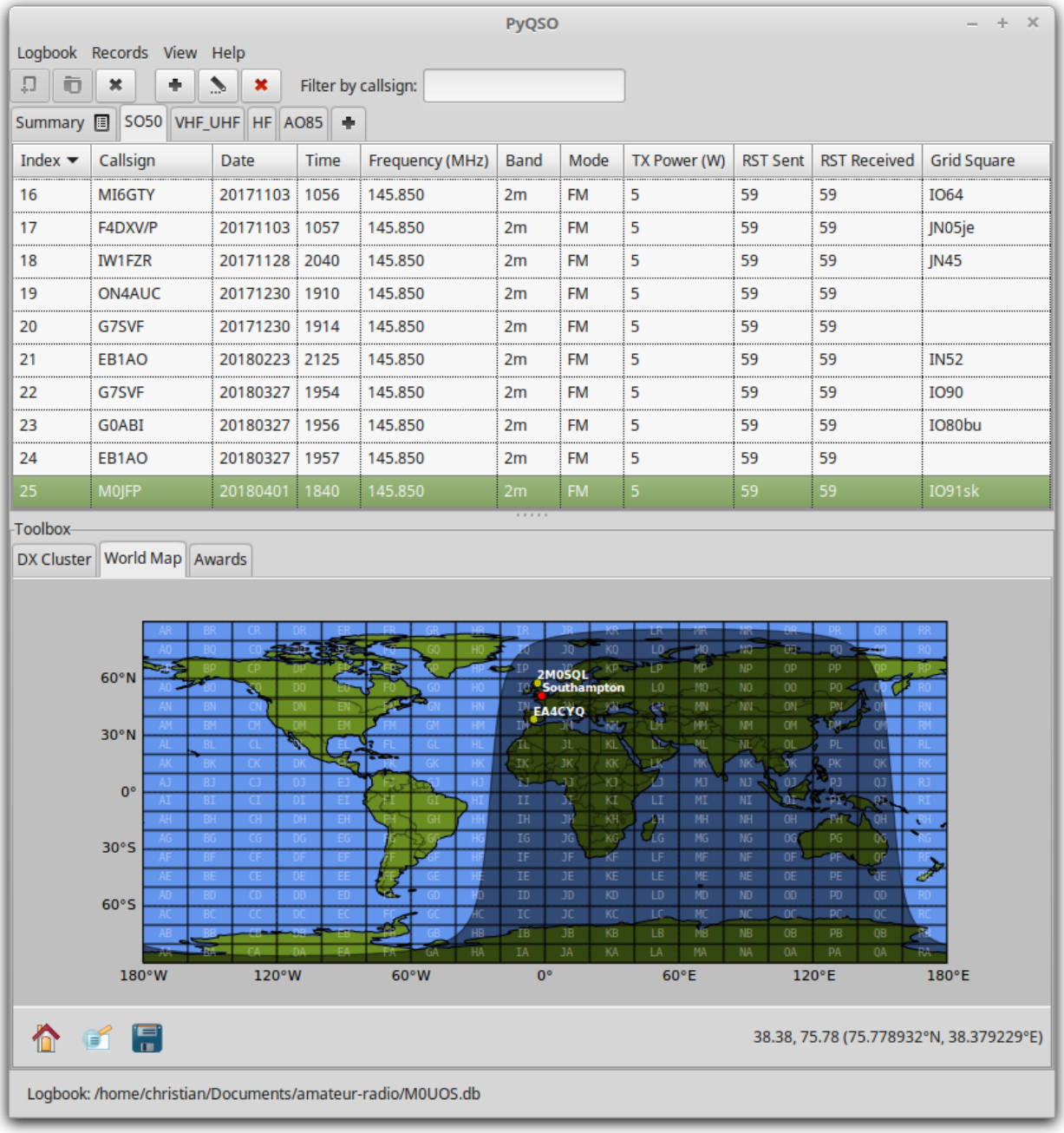

Fig. 2.1: The PyQSO main window, showing the records in a log called `SO50` (for contacts via the amateur radio satellite SO-50), and the World Map tool in the toolbox below it.

CHAPTER 3

Log management

Note 1: All the operations described below assume that a logbook is already open.

Note 2: Any modifications made to the logs are permanent. Users should make sure they keep up-to-date backups.

3.1 Creating a new log

To create a new log, click `New Log` in the `Logbook` menu and enter the desired name of the log in the dialog window that appears (e.g. repeater_contacts, dx, mobile_log). Alternatively, use the shortcut key combination `Ctrl + N`.

The log name must be unique (i.e. it cannot already exist in the logbook). Furthermore, it can only be composed of alphanumeric characters and the underscore character, and the first character in the name must not be a number.

Note: When logs are stored in the database file, field/column names from the ADIF standard are used. However, please note that only the following subset of all the ADIF fields is considered: CALL, QSO_DATE, TIME_ON, FREQ, BAND, MODE, SUBMODE, PROP_MODE, TX_PWR, RST_SENT, RST_RCVD, QSL_SENT, QSL_RCVD, NOTES, NAME, ADDRESS, STATE, COUNTRY, DXCC, CQZ, ITUZ, IOTA, GRIDSQUARE, SAT_NAME, SAT_MODE. Visit the ADIF website for more information about these fields.

3.2 Renaming a log

To rename the currently selected log, click `Rename Selected Log...` in the `Logbook` menu. Remember that the log's new name cannot be the same as another log in the logbook.

3.3 Deleting a log

To delete the currently selected log, click `Delete Selected Log` in the `Logbook` menu. As with all database operations in PyQSO, this is permanent and cannot be undone.

3.4 Exporting a log

While PyQSO stores logbooks in SQL format, it is possible to export individual logs in the well-known ADIF and Cabrillo formats. Select the log to export, and click `Export Log as ADIF...` or `Export Log as Cabrillo...` in the `Logbook` menu.

Note for contesters: Cabrillo records typically require contest QSO information in the form `CALL RST EXCH`, where `EXCH` denotes exchange information (e.g. a serial number or US state). No dedicated field exists in PyQSO to store exchange information so the RST fields should be used to store both the RST report *and* exchange information, separated by a space. The `RST Sent` field should therefore contain the RST and exchange information that you give to the other station (e.g. 59 001), and the `RST Received` field should contain the RST and exchange information that the other station gives you (e.g. 57 029). The export process asks for your callsign (this should be the callsign used during the contest) and the contest's name which can be selected from a drop-down list. If the contest name does not appear in this list, you may enter its name manually.

3.5 Importing a log

Records can be imported from an ADIF file. Upon importing, users can choose to store the records in a new log, or append them to an existing log in the logbook. To import, click `Import Log...` in the `Logbook` menu.

Note that each QSO record being imported must conform to the ADIF standard, otherwise the record will be ignored.

3.6 Printing a log

The log that is currently selected can be printed out on paper or printed to a PDF file by clicking `Print Log...` in the `Logbook` menu. Each page uses a landscape orientation to maximise the amount of QSO information per line. The following data is included: Index, Callsign, Date, Time, Frequency, Mode, RST Sent, and RST Received.

3.7 Filtering by callsign

Entering an expression such as `xyz` into the `Filter by callsign` box will instantly filter out all records whose callsign field does not contain `xyz`.

3.8 Sorting by field

To sort a log by a particular field name, click the column header that contains that field name. By default, it is the `Index` field that is sorted in ascending order.

CHAPTER 4

Record management

Note: Any modifications made to the records are permanent. Users should make sure they keep up-to-date backups.

4.1 Creating a new record (QSO)

A new QSO can be added by either:

- Clicking the `Add Record` button in the toolbar.
- Pressing `Ctrl + R`.
- Clicking `Add Record...` in the `Records` menu.

A dialog window will appear where details of the QSO can be entered (see *figure:edit_record*). Note that the current date and time are filled in automatically. When ready, click `OK` or press the Enter key to save the changes.

4.1.1 Callsign lookup

PyQSO can also resolve station-related information (e.g. the operator's name, address, and ITU Zone) by clicking the `Callsign lookup` button adjacent to the Callsign data entry box. Note that the user must first supply their qrz.com or hamqth.com account information in the preferences dialog window.

4.2 Editing a record

An existing record can be edited by:

- Double-clicking on it.
- Selecting it and clicking the `Edit Record` button in the toolbar.
- Selecting it and clicking `Edit Selected Record...` in the `Records` menu.

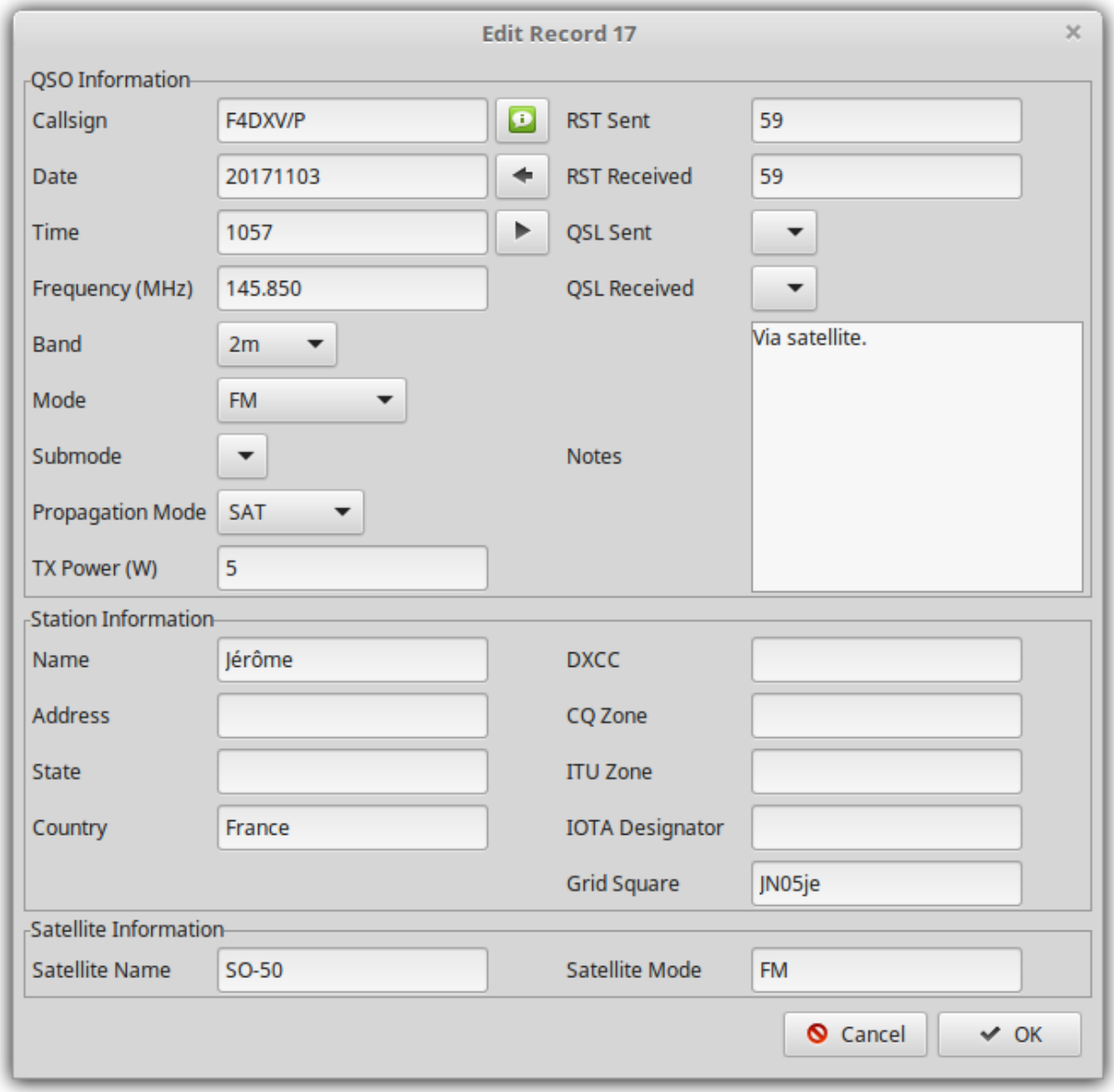

Fig. 4.1: Record dialog used to add new records and edit existing ones.

This will bring up the same dialog window as before.

4.3 Copying/pasting a record

An existing record can be copied and pasted by:

- Selecting it and right-clicking to bring up the popup menu.
- Selecting `Copy`.
- Right-clicking again and selecting `Paste`. This will duplicate the record, with the duplicate becoming the latest record in the selected log.

4.4 Deleting a record

An existing record can be deleted by:

- Selecting it and clicking the `Delete Record` button in the toolbar.
- Selecting it and pressing the `Delete` key.
- Selecting it and clicking `Delete Selected Record...` in the `Records` menu.

4.5 Removing duplicate records

PyQSO can find and delete duplicate records in a log. A record is a duplicate of another if its data in the Callsign, Date, and Time fields are the same. Click `Remove Duplicate Records` in the `Records` menu.

CHAPTER 5

Toolbox

The toolbox is hidden by default. To show it, click `Toolbox` in the `View` menu.

5.1 DX cluster

A DX cluster is essentially a server through which amateur radio operators can report and receive updates about QSOs that are in progress across the bands. PyQSO is able to connect to a DX cluster that operates using the Telnet protocol to provide a text-based alert service. As a result of the many different Telnet-based software products that DX clusters run, PyQSO currently outputs the raw data received from the DX cluster rather than trying to parse it in some way.

Click on `Connect to Telnet Server` then `New...` in the `Connection` menu, and enter the DX server details in the dialog that appears. If no port is specified, PyQSO will use the default value of 23. A username and password may also need to be supplied. Frequently used servers can be bookmarked for next time; bookmarked server details are stored in `~/.config/pyqso/bookmarks.ini`, where ~ denotes the user's home directory.

Once connected, the server output will appear in the DX cluster frame (see *figure:dx_cluster*). A command can also be sent to the server by typing it into the entry box beneath the server output and clicking the adjacent `Send Command` button (or pressing the Enter key).

5.2 World map

The world map tool (see *figure:world_map*) can be used to plot the QTH of your station and stations that you have contacted. It also features a grey line to check which parts of the world are in darkness. The position of the grey line is automatically updated every 30 minutes.

The user's QTH can be pinpointed on the map by specifying the QTH's location (e.g. city name) and latitude-longitude coordinates in the preferences. If the geocoder library is installed then these coordinates can be filled in for you by clicking the lookup button after entering the QTH's name, otherwise the coordinates will need to be entered manually.

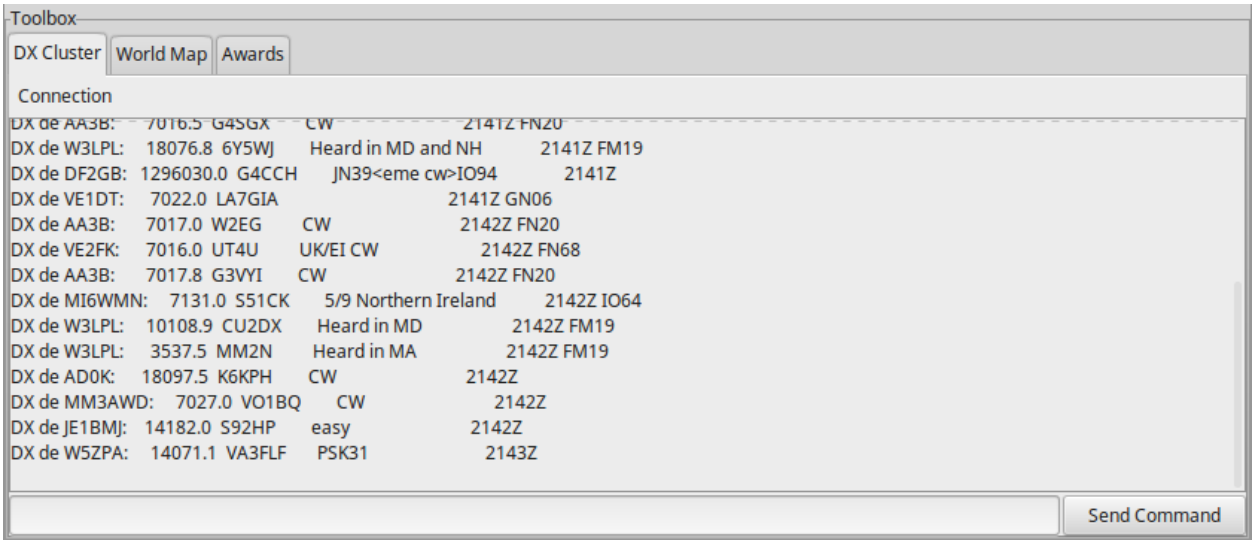

Fig. 5.1: The DX cluster frame.

The location of a worked station may also be plotted by right-clicking on the relevant QSO in the main window and selecting `Pinpoint` from the popup menu.

5.3 Awards

The awards progress tracker (see *figure:awards*) updates its data each time a record is added, deleted, or modified. Currently only the DXCC award is supported (visit the ARRL DXCC website for more information).

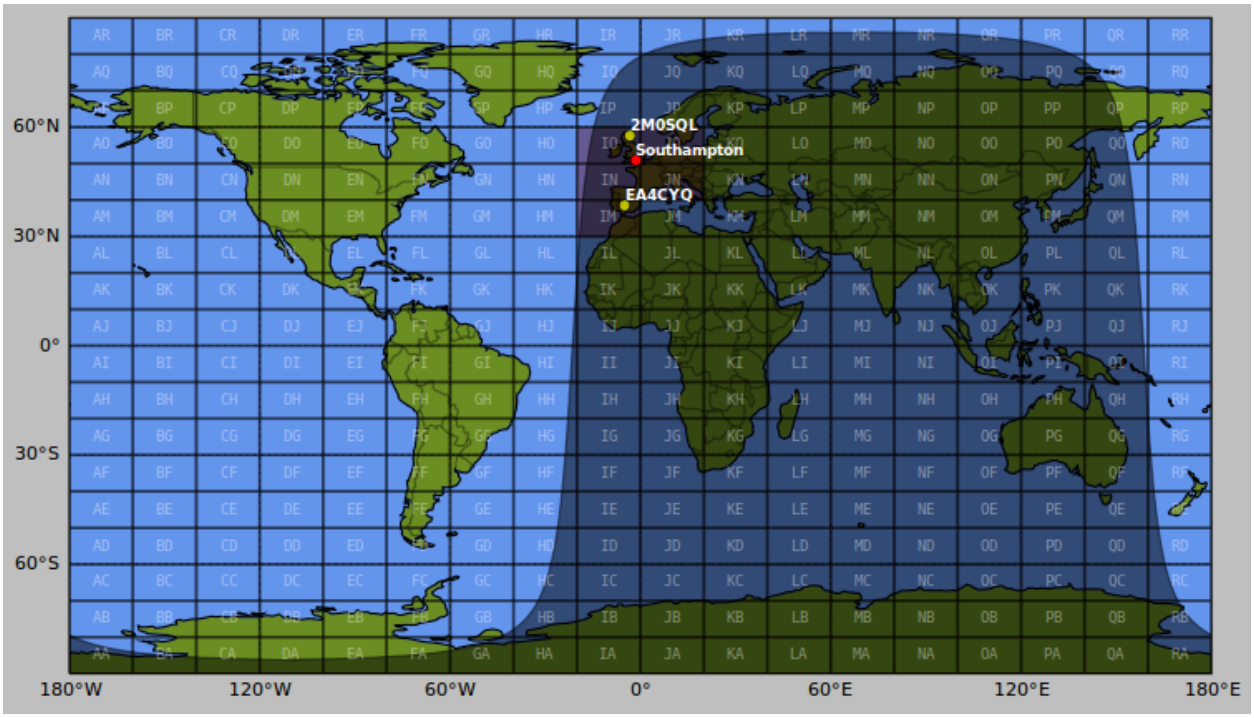

Fig. 5.2: The world map tool with the user's QTH (e.g. Southampton) pinpointed in red, and several other worked stations pinpointed in yellow. Worked grid squares are shaded purple.

DXCC Award

Modes	70cm	2m	6m	10m	12m	15m	17m	20m	30m	40m	80m	160m
Phone	2	7	0	0	0	0	0	2	0	2	0	0
CW	0	0	0	0	0	0	0	0	0	0	0	0
Digital	0	0	0	0	0	0	0	0	0	0	0	0
Mixed	2	7	0	0	0	0	0	2	0	2	0	0

Fig. 5.3: The award progress tracker.

CHAPTER 6

Preferences

PyQSO user preferences are stored in a configuration file located at `~/.config/pyqso/preferences.ini`, where ~ denotes the user's home directory.

6.1 General

Under the `General` tab, the user can choose to:

- Always show the toolbox (see the Toolbox section) when PyQSO is started.
- Display yearly logbook statistics on the Summary page when a logbook is opened (see *figure:summary*).
- Open a default logbook file.
- Keep the `Add Record` dialog window open after a new QSO is added, in preparation for the next QSO.

6.2 View

Not all the available fields have to be displayed in the logbook. The user can choose to hide a subset by unchecking them in the `View` tab. PyQSO must be restarted in order for any changes to take effect.

6.3 Records

The records tab comprises options concerning the Add/Edit Record dialog window. It allows users to:

- Use the UTC timezone when autocompleting the date and time fields.
- Choose whether the band should be automatically determined from the frequency field.
- Specify default values for the Power, Mode, and Submode fields.

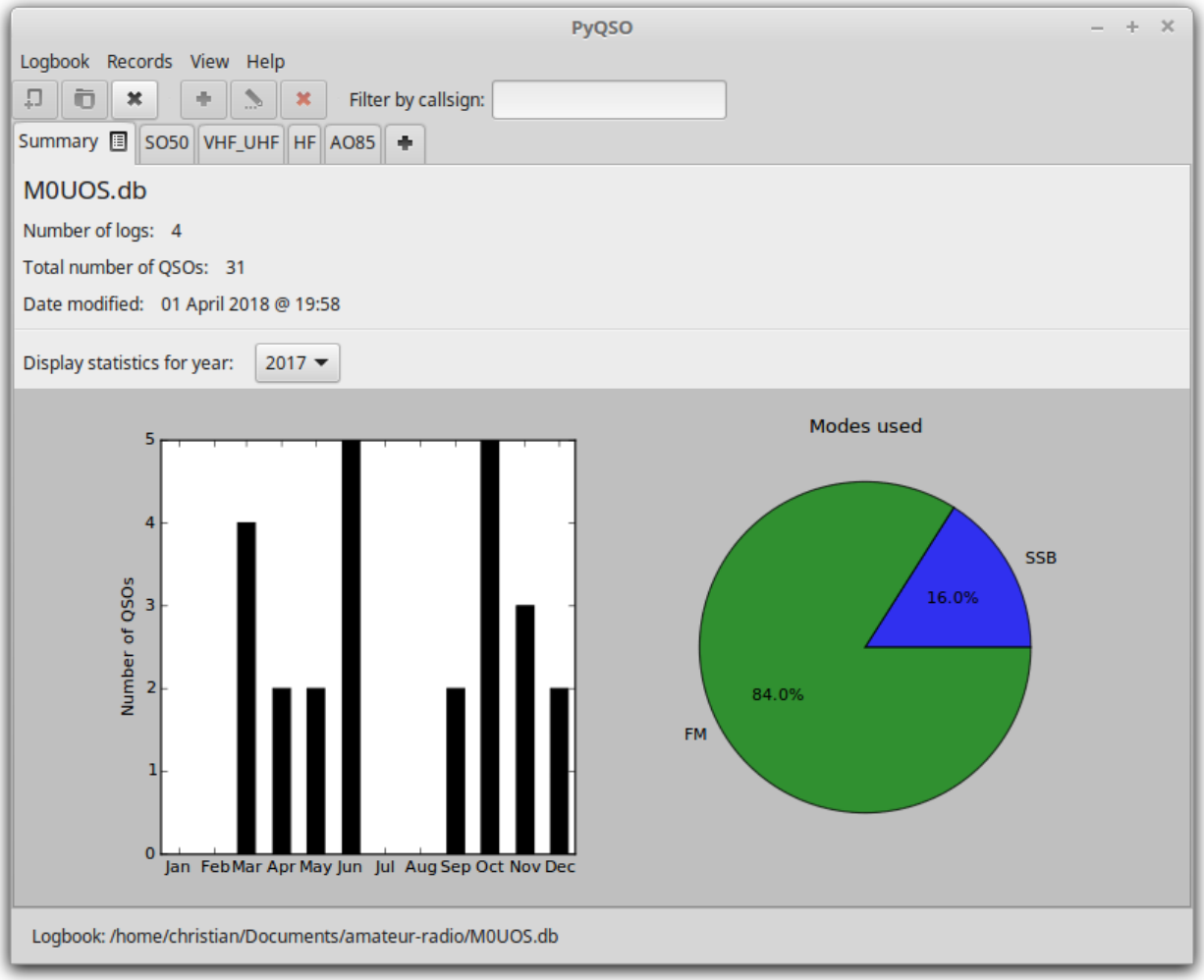

Fig. 6.1: The Summary page which appears after a logbook is opened. This presents some basic logbook statistics.

- Enter the QSO's frequency in a unit other than MHz (note that the frequency will always be presented in MHz in the main window, regardless of this preference).
- Specify the callsign lookup settings.

6.3.1 Callsign lookup

The user can enter their login details to access the qrz.com or hamqth.com database and perform callsign lookups. Note that these details are currently stored in plain text (unencrypted) format.

If the `Ignore callsign prefixes and/or suffixes` box is checked, then PyQSO will perform the callsign lookup whilst ignoring all prefixes (i.e. anything before a preceding "/" in the callsign) and the suffixes "P", "M", "A", "PM", "MM", "AM", and "QRP". For example, if the callsign to be looked up is F/MYCALL/QRP, only MYCALL will be looked up. If you get 'Callsign not found' errors, try enabling this option.

6.4 Import/Export

PyQSO currently supports the `NOTES` field in the ADIF specification, but not the `COMMENTS` field. When a user imports a log in ADIF format, they can choose to merge any existing text in the `COMMENTS` field with the `NOTES` field by checking the 'merge' checkbox. This way, no information in the `COMMENTS` field is discarded during the import process.

6.5 Hamlib support

PyQSO features rudimentary support for the Hamlib library. The name and path of the radio device connected to the user's computer can be specified in the `Hamlib` tab of the preferences dialog. Upon adding a new record to the log, PyQSO will use Hamlib to retrieve the current frequency and mode that the radio device is set to and automatically fill in the Frequency and Mode fields.

6.6 World Map

The user can pinpoint their QTH on the world map by specifying the latitude-longitude coordinates (or looking them up based on the QTH's name, e.g. city name) in the `World Map` tab. Maidenhead grid squares can also be rendered, with worked grid squares shaded, which is particularly useful for satellite operating.

CHAPTER 7

Keyboard shortcuts

Description	Shortcut
Open logbook	Ctrl + O
Close logbook	Ctrl + W
New log	Ctrl + N
Print log	Ctrl + P
Quit	Ctrl + Q
Add record	Ctrl + R
Edit record	Ctrl + E
Delete record	Delete

CHAPTER 8

pyqso package

8.1 Submodules

8.2 pyqso.adif module

class pyqso.adif.**ADIF**
Bases: object

The ADIF class supplies methods for reading, parsing, and writing log files in the Amateur Data Interchange Format (ADIF). For more information, visit http://adif.org/

is_valid(*field_name*, *data*, *data_type*)
Validate the data in a field with respect to the ADIF specification.

Parameters

- **field_name** (*str*) – The name of the ADIF field.
- **data** (*str*) – The data of the ADIF field to validate.
- **data_type** (*str*) – The type of data to be validated. See http://www.adif.org/304/ADIF_304.htm#Data_Types for the full list with descriptions.

Returns True or False to indicate whether the data is valid or not.

Return type bool

parse_adi(*text*)
Parse some raw text (defined in the 'text' argument) for ADIF field data.

Parameters **text** (*str*) – The raw text from the ADIF file to parse.

Returns A list of dictionaries (one dictionary per QSO). Each dictionary contains the field-value pairs, e.g. {"FREQ": "145.500", "BAND": "2M", "MODE": "FM"}.

Return type list

read(*path*)
Read an ADIF file and parse it.

Parameters **path** (*str*) – The path to the ADIF file to read.

Returns A list of dictionaries (one dictionary per QSO), with each dictionary containing field-value pairs, e.g. {FREQ:145.500, BAND:2M, MODE:FM}. If the file cannot be read, the method returns None.

Return type list

Raises **IOError** – If the ADIF file does not exist or cannot be read (e.g. due to lack of read permissions).

write(*records*, *path*)
Write an ADIF file containing all the QSOs in the 'records' list.

Parameters

- **records** (*list*) – The list of QSO records to write.
- **path** (*str*) – The desired path of the ADIF file to write to.

Returns None

Raises **IOError** – If the ADIF file cannot be written (e.g. due to lack of write permissions).

8.3 pyqso.auxiliary_dialogs module

8.4 pyqso.awards module

8.5 pyqso.blank module

8.6 pyqso.cabrillo module

class pyqso.cabrillo.**Cabrillo**
Bases: object

The Cabrillo class supplies methods for writing log files in the Cabrillo format (v3.0). For more information, visit http://wwrof.org/cabrillo/

write(*records*, *path*, *contest=''*, *mycall=''*)
Write a list of QSO records to a file in the Cabrillo format.

Parameters

- **records** (*list*) – The list of QSO records to write.
- **path** (*str*) – The desired path of the Cabrillo file to write to.
- **contest** (*str*) – The name of the contest.
- **mycall** (*str*) – The callsign used during the contest.

Returns None

Raises **IOError** – If the Cabrillo file cannot be written (e.g. due to lack of write permissions).

8.7 pyqso.cabrillo_export_dialog module

class pyqso.cabrillo_export_dialog.**CabrilloExportDialog**(*application*)
Bases: object

A handler for the Gtk.Dialog through which a user can specify Cabrillo log details.

contest
Return the name of the contest.

Returns The name of the contest.

Return type str

mycall
Return the callsign used during the contest.

Returns The callsign used during the contest.

Return type str

8.8 pyqso.calendar_dialog module

class pyqso.calendar_dialog.**CalendarDialog**(*application*)
Bases: object

Handler for a simple dialog containing a Gtk.Calendar widget. Using this ensures the date is in the correct YYYYMMDD format required by ADIF.

date
Return the date from the Gtk.Calendar widget in YYYYMMDD format.

Returns The date from the calendar in YYYYMMDD format.

Return type str

8.9 pyqso.callsign_lookup module

8.10 pyqso.compare module

pyqso.compare.**compare_date_and_time**(*model*, *row1*, *row2*, *user_data*)
Compare two rows (let's call them A and B) in a Gtk.ListStore, and sort by both date and time.

Parameters

- **model** (*Gtk.TreeModel*) – The model used to sort the log data.
- **row1** (*Gtk.TreeIter*) – The pointer to row A.
- **row2** (*Gtk.TreeIter*) – The pointer to row B.
- **user_data** – The specific column from which to retrieve data for rows A and B.

Returns -1 if Row B's date/time is more recent than Row A's; 0 if both dates and times are the same; 1 if Row A's date/time is more recent than Row B's.

Return type int

pyqso.compare.**compare_default**(*model*, *row1*, *row2*, *user_data*)
The default sorting function for all Gtk.ListStore objects.

Parameters

- **model** (*Gtk.TreeModel*) – The model used to sort the log data.
- **row1** (*Gtk.TreeIter*) – The pointer to row A.
- **row2** (*Gtk.TreeIter*) – The pointer to row B.
- **user_data** – The specific column from which to retrieve data for rows A and B.

Returns -1 if the value of Row A's column value is less than Row B's column value; 0 if both values are the same; 1 if Row A's column value is greater than Row B's column value.

Return type int

8.11 pyqso.dx_cluster module

8.12 pyqso.grey_line module

8.13 pyqso.log module

8.14 pyqso.log_name_dialog module

class pyqso.log_name_dialog.**LogNameDialog**(*application*, *title=None*, *name=None*)
Bases: object

A handler for the Gtk.Dialog through which a user can specify the name of a Log object.

name
Return the log name specified in the Gtk.Entry box by the user.

Returns The log's name.

Return type str

8.15 pyqso.logbook module

8.16 pyqso.menu module

8.17 pyqso.preferences_dialog module

8.18 pyqso.printer module

8.19 pyqso.record_dialog module

8.20 pyqso.summary module

8.21 pyqso.telnet_connection_dialog module

class pyqso.telnet_connection_dialog.**TelnetConnectionDialog**(*application*)

Bases: object

A handler for the Gtk.Dialog through which a user can specify Telnet connection details.

bookmark

Return True if a new bookmark should be created, otherwise return False.

Returns True if a new bookmark should be created, otherwise False.

Return type bool

host

Return the Telnet server's host name.

Returns The server's host name.

Return type str

password

Return the user's password.

Returns The user's password.

Return type str

port

Return the Telnet server's port number (as a string).

Returns The server's port number (as a string).

Return type str

username

Return the user's username.

Returns The user's username.

Return type str

8.22 pyqso.toolbar module

class pyqso.toolbar.**Toolbar**(*application*)
Bases: object

The toolbar underneath the menu bar.

set_logbook_button_sensitive(*sensitive*)
Enable/disable logbook-related toolbar items.

Parameters **sensitive** (*bool*) – If True, enable the 'new logbook' and 'open logbook' toolbar items. If False, disable them.

set_record_buttons_sensitive(*sensitive*)
Enable/disable record-related toolbar items.

Parameters **sensitive** (*bool*) – If True, enable all the record-related toolbar items. If False, disable them all.

8.23 pyqso.toolbox module

8.24 Module contents

Python Module Index

p

Index