MRT

User's Manual

1.4.6

Table of Contents

Introduction

MRT (Mr T) was born out of a request for a very lightweight, completely text based, Morse application that could provide the core capability of connecting to a KOBServer Wire to send and receive code, supporting a local key and sounder, and allowing input from the keyboard. The hope was to be able to run on small devices like the Raspberry Pi Zero W, Inovato Quadra, and the like, in order to provide a low cost solution for American Telegraph displays. It was also requested that this be able to be started at system start-up and run (basically unattended) in the background.

Preface

Though lightweight, MRT provides a lot of functionality, some of which it tailored specifically to use in displays, and not available in MKOB.

This user’s manual covers the key and unique features of MRT, while referring to other documentation for details on common functionality.

Related Components

pykob Package

MRT uses the **pykob** package of modules. The pykob package consists of a number of modules that provide different aspects of handling code, the physical key and sounder, internet connection to a KOBServer instance, system audio to simulate a sounder or provide tone, configuration and logging capabilities, and more. When updating MRT, it is advisable to get the latest (associated) release of pykob as well.

MKOB (Morse Key on Board)

MKOB is the full, graphical, user application. For most interactive use, MKOB is probably the application to use.

There is a full, illustrated, User’s Guide that describes MKOB and its operation. Please refer to the **MKOB User’s Guide** for descriptions of the core functionality, creating and using configurations, interfacing a key and sounder, using the *Virtual Closer*, and more.

Telegram

Telegram is an application specifically designed for use in displays. While MRT can be used in displays, its focus is in being lightweight and text (only) based. Telegram, on the other hand, is graphical, and presents a full screen interface with no exposed controls. Its interface appears as a telegram form that fills the entire screen. Received messages are displayed on the form. The form is cleared, or scrolled off, at the end of each message, or after a configurable period of idle time. The appearance of the form is configurable without changing any code.

Utilities

The pykob package also includes a number of utility applications. The README file included in the package contains a short description of each of the utility applications.

Quick Start

Get started with MRT by using **MKOB** or **Configure** to set up your configuration and then launch MRT.

MRT is run from the command line and accepts a number of command-line options. Use the `--help` option to get a description of the available options.

When run, MRT will display incoming code as text. It accepts text typed on the keyboard and converts it to code, which is sent to the wire (if connected). It also listens to an attached key and sends the code to the wire, as well as decoding and displaying it.

User’s Guide

Mr T automatically connects to the wire when it is started and it remains connected while it is running. There is no option to connect/disconnect during operation.

The exception to this is if wire number 0 is specified. In that case, MRT does not connect to a wire, and can be used for local displays or practice. In addition, a **Selector** can be used to switch configurations that can use different wires, including 'no wire' (see the Selector section for details).

It monitors the wire and the local key. If the local key is opened, keyed code will be sent to the wire. While the local key is closed, Mr T will sound, as well as decode and display, code received from the wire.

The station name of the sending station is displayed when the sending station changes.

Keyboard Commands

MRT automatically connects to the wire, either from the configuration or specified on the command line. There are four keyboard commands:

1. Ctrl-Z - Provide help.
2. '~' - Open the key.
3. '+' - Close the key.
4. Ctrl-C - Exit

If the key is open (either physically or by using '~'), other text typed will be sounded, displayed, and encoded and sent to the wire if connected.

(Simple, right!)

Exiting

Mr T runs continuously once started. To stop, enter a Ctrl-C (^C) on the keyboard or kill the process.

Send/Sound a File

Mr T can process and send/sound the contents of a text file. The file is specified using the `--file` command-line option. Along with the `--file` option, the `--repeat` option will cause the sending/sounding of the file to be repeated continuously with a configurable delay between each iteration.

Send/Sound a Recording

Mr T can process and send/sound a PyKOB Recording. The recording is specified using the `--play` command-line option. Along with the `--play` option, the `--repeat` option can be used. It causes the sending/sounding of the recording to be repeated continuously with a configurable delay between each iteration.

For those not familiar with a PyKOB Recording, one can be created using MKOB or MRT and it stores the actual code (timings). Therefore, it can be used to store and replay a session, keeping the fist of all of the participants.

Using a Selector (switch or panel)

Mr T supports using a Selector (hardware) to select between 4 different configurations (or 16, depending on the hardware). The Selector Switch attaches to a serial port and uses the hardware-handshake signals: CTS, DSR, DCD, RI to select either 1 of 4 or 1 of 16 configurations to use. The selector can be changed, and MRT will change to the selected configuration 'on-the-fly'.

The configurations are specified in a Selector Spec. The format of a Selector Spec is described below.

The `--Selector`/`--selector` command-line option is used to specify the Selector Spec file. The option also identifies the port (COM/tty) that the selector switch is connected to. The special value **SDSEL** can be used to automatically search for a *SilkyDESIGN* Selector Switch and use it.

The PyKOB Repository on GitHub contains a **hw** folder that has a schematic and other resources for a selector switch.

Selector Specification (Spec)

There is an example spec file included in the Documentation folder: mrt_sel_spec_example.mrtsel

JSON Format

The specification file is a JSON format file, as used by JavaScript, Python, and many other programming languages. Familiarity with JSON can be helpful, but is not required to be able to create or modify a specification file to meet your needs. It is suggested that you start with the example file, as that will give you everything you need, and you should be able to simply modify some of the content.

File is JSON Format Map/Dictionary

The top entries are:

- "type": It can have a value of:
    1. "1OF4" - Hardware that selects 1 of 4 options
    2. "BINARY" - Hardware that uses the 4 handshake signals to select 0-15 (1 of 16 options)
- "selections": This is an array/list with an entry for each possible selection
- Each element of the array/list is a map/dictionary with two entries:
    1. "desc": The description of the selection. This is displayed in the terminal window when the selector is switched to that selection
    2. "args": The value is an array/list with the command line arguments that would be used when executing MRT (if you did it from the command line)
        - Each argument is a string in quotes (even if it’s a number like the 'wire'). Arguments are separated by a comma. The configuration environment used to run the main MRT is the base for the selections, so arguments specified are applied to it to run the selection. Therefore, only a wire, or possibly a wire and a few additional options might be all that is needed as the "args" value

The elements are in order. The first is #1 (for a '1OF4' type), second is #2, etc.

If there are fewer elements than needed (less than 4 or less than 16), the rest are defaulted to running MRT with the configuration of the parent (less the '--selector port spec', of course). If a Selector Switch can’t be found, and the `--selector` option is used (rather than `--Selector`), MRT will keep trying to find a selector in the background and will use selection 1 until a selector is found.

The only restriction is that you cannot specify '--selector' in these arguments.

Note: JSON strings must be enclosed within quotes *"text"* (not apostrophes *'text'*). The values are case-sensitive.

Scheduled Feed Functionality

Mr T can be configured to send messages at a specified time, or after a certain period of idle time. The **Scheduled Feed** capability has options to break-in even if the wire is busy, only send if the wire is idle, or send after a period of inactivity. It can send a message selected randomly from a collection of messages. The messages support delays and getting content from environment variables.

The messages, times, and options are configured in a SchedFeed Specification. The following section describes the SchedFeed Specification.

The `--schedfeed` command-line option is used to specify the SchedFeed Spec file.

Scheduled Feed Specification

Having Mr T send code on a schedule (using the `--schedfeed sf_spec_file` option) requires a 'mrtsfs' scheduled-feed specification file (sf_spec_file) to control when and what MRT sends.

There is an example spec file included in the Documentation folder: mrt_schfd_example.mrtsfs

JSON Format

The specification file is a JSON format file, as used by JavaScript, Python, and many other programming languages. Familiarity with JSON can be helpful, but is not required to be able to create or modify a specification file to meet your needs. It is suggested that you start with the example file, as that will give you everything you need, and you should be able to simply modify some of the content.

File is JSON Format Map/Dictionary

The top entry is:

- "specs": The value is a list of one or more specifications.

Each specification is a Map/Dictionary with the following keys:value

- **condition**:*time* The 'condition' is one of:
  1. "at" - Send AT this **time**, even if wire is busy (breaks in if needed). **time** = 24-Hour Local Time
  2. "at_ii" - AT this **time** IF the wire is IDLE. If not idle at this time the message is not sent (it is skipped). **time** = 24-Hour Local Time
  3. "when_i" - Send WHEN the line is IDLE, at/after this **time**. **time** = 24-Hour Local Time
  4. "idle" - Send if the line remains IDLE for a given **period of time**. **time** = Minutes of idle time
- 'msg':*message text* The (single) message to be sent when the **condition** is satisfied.
- or
- 'msgs':*List of message texts* Two or more messages, from which one will be selected to send, randomly, when the condition is satisfied.

The **message text** is the message to be sent, just as you would enter it using the keyboard sender. Two special control sequences are supported within the text.

Message Text Control Sequences

Control sequences are in the form: «**control**»

Where '«' is Unicode U+00AB (Left-double-angle-quote) and '»' is Unicode U+00BB (Right-double-angle-quote) and **control** is:

- **\$***var* : A `$` followed by the name of an environment variable. The value of the environment variable will be used in the text.
- **P***seconds* : Pause for **seconds** before continuing to send. The value of **seconds** can include fractional values (3.5 = Three and a half seconds)

Though not considered **Control Sequences**, the other two **Special** characters used within the message text are:

- **~** : Open the key
- **+** : Close the key

These must be included in the text unless you know that the key will be open when the message is to be sent, and/or you want to leave the key open after sending the message.

Note: JSON strings must be enclosed within quotes *"text"* (not apostrophes *'text'*). The values are case-sensitive.

Example

It is suggested that you refer to the `mrt_schfd_example.mrtsfs` file, as it is tested. However, for convenience, the following should help:

```
{
    "specs": [
        {
            "at":906,
            "msg":"~ «$OFFICE» ON +     ~ AR «$OFFICE» +"
        },
        {
            "at_ii":1207,
            "msg":"~ «$OFFICE» CHECKING IN= AR «$OFFICE» +"
        },
        {
            "when_i":1950,
            "msg":"~ «$OFFICE» DONE +     ~ 30 «$OFFICE» +"
        },
        {
            "idle":10,
            "msgs":[
                "~ I «$OFFICE» 37 SEEING WHO IS THERE. = 73 «$OFFICE» +",
                "~ I «$OFFICE» INVITING YOU TO JOIN IN. = K «$OFFICE» +",
                "~ I «$OFFICE» WELCOME TO OUR WIRE. = «P3.2» WE HOPE YOU
ARE ENJOYING YOURSELF. K «$OFFICE» +",
                "~ I «$OFFICE» 77 PLEASE COME VISIT «$SITE_NAME» AND SEE
US IN ACTION. = K «$OFFICE» +"
            ]
        }
    ]
}
```

Let’s say that the environment contains an environment variable named *OFFICE* with a value of *"SM, Seattle"* and *SITE_NAME* with a value of _"TELEGRAPH MUSEUM".

This example will…

- At 9:06 local time, whether the wire is busy or not:
    - Open the key
    - Send the message: "SM, SEATTLE ON"
    - Close the key
    - Open the key
    - Send the message: "AR SM, SEATTLE"
    - Close the key
- At 12:07 local time, if the wire is idle: (if the wire is busy, this message will not be sent)
    - Open the key
    - Send the message: "SM, SEATTLE CHECKING IN= AR SM, SEATTLE"
    - Close the key
- Starting at 19:50 (7:50 PM) local time, wait for the wire to become idle, then:
    - Open the key
    - Send the message: "SM, SEATTLE DONE"
    - Close the key
    - Open the key
    - Send the message: "30 SM, SEATTLE"
    - Close the key
- At any time, if the wire is continuously idle for 10 minutes:
    - Pick one of the following 4 messages to send. Once sent, it will wait for 10 more minutes of idle time.
        - (open key) "SM, SEATTLE SEEING WHO IS THERE. = 73 SM, SEATTLE" (close key)

- (open key) "SM, SEATTLE INVITING YOU TO JOIN IN. = K SM, SEATTLE" (close key)
- (open key) "SM, SEATTLE WELCOME TO OUR WIRE. = " (pause 3.2 seconds) "WE HOPE YOU ARE ENJOYING YOURSELF. K SM, SEATTLE" (close key)
- (open key) "SM, SEATTLE= 77 PLEASE COME VISIT TELEGRAPH MUSEUM AND SEE UP IN ACTIN. = K SM, SEATTLE" (close key)

Command-Line Options

Mr T is controlled by command-line options in addition to the MKOB/PyKOB Configuration being used (a command-line option allows specifying which configuration to use, rather than using the global configuration). It is suggested that the `--help` command-line option be used to get the up-to-date description of the options available. The following list is provided for convenience.

```
MRT 1.4.6
usage: MRT.py [-h] [-a sound] [-A sounder] [-g gpio] [-P serial] [-p
portname]
              [-S station] [-c wpm] [-t wpm] [--config config-file]
              [--logging-level logging-level] [--record
filepath|['A'|'AUTO']]
              [--senderdt] [--file text-file-path] [--play recording-path]
              [--repeat delay] [--schedfeed feedspec-path]
              [--Selector port specfile-path] [--selector port specfile-
path]
              [wire]

Morse Receive & Transmit (Mr T). Receive from wire and send from key. The
Global configuration is used except as overridden by options.

positional arguments:
  wire                  Wire to connect to. If specified, this is used
rather
                        than the configured wire. Use 0 to not connect.

options:
  -h, --help            show this help message and exit
  -a sound, --sound sound
                        'ON' or 'OFF' to indicate whether computer audio
                        should be used to sound code.
  -A sounder, --sounder sounder
                        'ON' or 'OFF' to indicate whether to use sounder
if
                        'gpio' or `port` is configured.
  -g gpio, --gpio gpio  'ON' or 'OFF' to indicate whether GPIO (Raspberry
Pi)
```

```
                        key/sounder interface should be used. GPIO takes
                        priority over the serial interface if both are
                        specified.
  -P serial, --serial serial
                        'ON' or 'OFF' to indicate whether a Serial
key/sounder
                        interface should be used. GPIO takes priority over
the
                        Serial interface if both are specified.
  -p portname, --port portname
                        The name/ID of the serial port to use, or the
special
                        value 'SDIF' to try to find a SilkyDESIGN-
Interface,
                        or 'NONE'.
  -S station, --station station
                        The Station ID to use (or 'NONE').
  -c wpm, --charspeed wpm
                        The minimum character speed to use in words per
                        minute.
  -t wpm, --textspeed wpm
                        The morse text speed in words per minute. Used for
                        Farnsworth timing. Spacing must not be 'NONE' to
                        enable Farnsworth.
  --config config-file  Configuration file to use. The special value
'GLOBAL'
                        will use the global (un-named) configuration. The
                        special value 'NEW' will use a new (defaults)
                        configuration.
  --logging-level logging-level
                        Logging level. A value of '0' disables DEBUG
output,
                        '-1' disables INFO, '-2' disables WARN, '-3'
disables
                        ERROR. Higher values above '0' enable more DEBUG
                        output.
  --record filepath|['A'|'AUTO']
                        Record the session to a PyREC recording file. The
file
                        is 'filepath' if specified or is auto-generated if
                        'AUTO'.
  --senderdt            Add a date-time stamp to the current sender
printed
                        when the sender changes.
  --file text-file-path
                        Key a text file when started. Code will be sent if
                        connected to a wire.
  --play recording-path
```

```
                        Play a recording when started. Code will be sent
if
                        connected to a wire.
  --repeat delay        Used in conjunction with '--play' or '--file',
this
                        will cause the playback or file processing to be
                        repeated. The value is the delay, in seconds, to
pause
                        before repeating.
  --schedfeed feedspec-path
                        Schedule feeds that send from within MRT. The
option
                        value is a path to a feeds specification file. See
the
                        MRT Users Guild for a description of the format of
the
                        specification.
  --Selector port specfile-path
                        Use a PyKOB Selector to run MRT with different
options
                        based on the MRT Selector Specification file
                        'specfile-path' and the current selector setting
of a
                        selector connected to port 'port'. Exit with an
error
                        if the port cannot be found (the selector is not
                        available). SEE: '--selector' to specify a
selector,
                        but run normally if the port cannot be found. The
                        special 'port' value of 'SDSEL' can be used to
look
                        for a SilkyDESIGN Selector Switch rather than
using a
                        specific port.
  --selector port specfile-path
                        Same as '--Selector' except that MRT will run
normally
                        if the selector port cannot be found/used.
```

Running at Start-Up on Linux

There have been requests to have something that automatically runs by just turning it on. This can be done on a small Linux box like the Inovato Quadra or a Raspberry Pi Zero-W.

Here are instructions for the Quadra, but they can easily be applied to the RPi…

1. Script in /home/quadra: RunMRT.sh

```
#!/bin/sh
cd /home/quadra/PyKOB
/home/quadra/.local/bin/python3.11 MRT.py > MRT.out 2>/dev/null &
```

2. Make it executable:

```
$chmod +x RunMRT.sh
```

3. Create a crontab for the quadra user and have this run on reboot:

```
$crontab -e
```

(pick the editor you want to use)

-at the end after the comments-

```
@reboot /home/quadra/RunMRT.sh
```

4. Reboot:

```
$sudu reboot now
```

-After Reboot-

5. Check that it's running:

```
$ps -aux | grep MRT
```

should list something like this:

```
quadra      1504 21.8  2.4 461672 50412 ?        Sl   21:12   0:14
/home/quadra/.local/bin/python3.11 MRT.py
quadra      2915  0.0  0.0   5904   648 pts/0    S+   21:13   0:00 grep
MRT
```

6. If you need to kill it, use `kill -9` with the process number from above:

```
$kill -9 1504
```

The script runs the command in the background (using the '&' at the end). That is needed to allow the crontab processing to continue past that command. The script also causes the command error output to be thrown away, as there is a lot of it from the ALSA subsystem. It sends the regular output to 'MRT.out' in the PyKOB directory. Another way to avoid the ALSA errors is to create a named configuration that has the 'System Sound' disabled (`-a OFF`). This keeps the audio module from being loaded, and therefore removes (at least most of) the ALSA errors to be avoided. The regular output could also be thrown away, but it can be nice to be able to check that it is doing something. To check on it, tail the MRT.out file.

Index

Colophon

This document reflects the features of MRT Version 1.4.6 Edit date: 12-10-2024

This document is authored using **asciidoc**, and the pdf version rendered through **asciidoctor-pdf**, to create a manual that is more readable and more pleasing to the eye.

Dedication

The team would like to thank everybody who through comments, criticism and suggestions has helped to make the MRT application and this document better and more usable.

Special thanks go out to Les Kerr, who's vision and effort created the original MorseKOB application, and who initiated the Python version.