WIMBORNE MODEL TOWN

2016-17 ENHANCEMENTS

INSTALLATION SPECIFICATION

Issue 1.0

25th June 2017

Table of Contents

Index of Tables

1 INTRODUCTION

This specification provides information for future developers of the Minster Functions and On-site Webserver at Wimborne Model Town.

Where possible, the system has been designed to be as easy as possible to operate, maintain and upgrade.

This document should be read in conjunction with the User Manual.

2 RELATED DOCUMENTS

2.1 Development Documents

The following documents were generated as part of this development.

Table 2.1: Related Documents

No	*Document*	*Iss*
1	Wimborne Model Town – 2016-17 Enhancements - Requirements	3
2	Wimborne Model Town – 2016-17 Enhancements - Design	1
3	Wimborne Model Town – 2016-17 Enhancements – User Manual	1

2.2 Referenced Documents and Resources

The following documents and resources are useful for supporting development.

Table 2.2: Related Documents

No	*Resource*	*Ref*
1	Python Tutorial	https://docs.python.org/2/tutorial/index.html
2	Python Documentation	https://docs.python.org/2/index.html
3	Raspberry Pi GPIO Tutorials	http://raspi.tv/
4	Python Lists	https://docs.python.org/2/glossary.html#term-list
5	The apscheduler Python Library	http://apscheduler.readthedocs.io/en/latest/index.html

No	*Resource*	*Ref*
6	Readonly Root with OverlyFS	https://www.raspberrypi.org/forums/viewtopic.php?f=66&t=173063
7	How To Autorun A Python Script On Raspberry Pi Boot	http://www.raspberrypi-spy.co.uk/2015/02/how-to-autorun-a-python-script-on-raspberry-pi-boot/
8	Etcher	https://etcher.io/
9	The Pi Hut	http://thepihut.com/blogs/raspberry-pi-tutorials/17789160-backing-up-and-restoring-your-raspberry-pis-sd-card

3 ABBREVIATIONS

• ESD	-	Electrostatic Sensitive Device
• IDE	-	Integrated Development Environment
• MOSFET	-	Metal Oxide Silicon Field Effect Transistor

4 PROJECT ARCHIVE

A copy of all project documentation, code and software environments may be found on a CD in the Model Railway Tool Store along with the Spares.

5 HARDWARE INSTALLATION

Installation of the hardware is covered in the User Manual and this document is confined to work that will only need to take place if the system has been revised and upgraded; an activity that might trigger a need to strip the system out or redesign some elements.

5.1 Electrostatic Sensitive Devices

CAUTION Electronic devices used in the system are sensitive to static electricity. Precautions should be taken when handling these devices in order to prevent permanent damage.

The Raspberry Pi Zero, the Relay Board and the MOSFET Driver all contain devices that could be damaged if they are exposed to static electricity. Static Electricity is readily generated when people walk over dry floors, rub against furniture or simply rub against their clothing. Although rarely noticed, the level generated is sufficient to damage devices if precautions are not taken. It is therefore recommended that no work be undertaken on these devices without taking one of the following actions:

1. If possible an ESD workstation be employed. An ESD workstation includes an Antistatic Mat, a conductive wrist band and a means of earthing the mat and band, usually via a three-pin mains plug with a resistive earth pin.
2. It is recognised that the WMR Model Railway building is not conducive to setting up an ESD workstation, so maintainers should endeavour to earth themselves by touching an earthed metal object at intervals during the work, especially after it has been necessary to leave the work area and return. A good compromise would be to use a wrist band clipped to a convenient earth point.

5.2 System Layout

Full details of the physical and electrical layout of the system may be found in the User Manual.

6 SOFTWARE DEVELOPMENT AND INSTALLATION

6.1 Development

6.1.1 Software Environment

6.1.1.1 Minster Functions

The writing of the software may be performed on any computing device, but will be simplified if suitable tools are available, see below. Debugging must be performed on the target device, with suitable hardware attached (see Section 6.1.3 below) in order that the Python Interpreter may import all the necessary libraries and the software functions may be observed in operation. During the original development a Raspberry Pi Model 3 running the Raspbian Operating System, was used for most of the development.

6.1.1.2 Webserver

The writing of the software may be performed on any computing device that has suitable web development tools as described below. Code may also be produced on any type of computer.

Debugging must be performed in conjunction with a deployed webserver. During the original development initial testing was carried out by uploading the code to a commercial webserver before porting it to the Raspberry Pi 3 running the Raspbian Operating System.

6.1.2 Development Tools

6.1.2.1 Minster Functions

Raspbian includes all of the tools necessary to write and develop the program. The original development was carried out using the IDLE Integrated Development Environment, but other development toolsets may be used if desired.

6.1.2.2 Webserver

Suitable web development tools may be found on almost any platform and writing of the code may be achieved with nothing more than a simple editor, such as the one provided in Raspbian or a more capable editor, such as Bluefish, which is available on many Linux Distros. In Windows, other tools are available, but care must be taken if a high-end tool, such as Dreamweaver, is employed, to ensure that unnecessary features do not generate ‘bloated’ code.

6.1.3 Hardware Environment

6.1.3.1 Minster Functions

The hardware in this system was developed and assembled before final debugging was carried out, since this proved to be feasible in the time available. However, this is not essential, and a breadboard with push-buttons and LEDs could be employed for future enhancements.

6.1.4 Development Backup

A backup image of the Development SD card may be found on a CD in the archive.

6.1.5 Detailed Description of Software Functions

6.1.5.1 Minster Functions

A detailed description of the key features of the software is provided below. The purpose of this is to allow future developers, who may not be experienced Python programmers, to understand how the more advanced features work.

Links to further research are also provided, where appropriate.

6.1.5.1.1 General

The Minster Functions code was written using Python Version 2.7.9. For simple code constructs and approaches, a good start would be this Python Tutorial. The full documentation for Python 2.7 may be found in the Python Docs and may be needed to interpret the detailed descriptions provided below.

Similarly, The RPi.GPIO Library was used to control the input and output functions and a series of Tutorials may be found at Alex Eames site.

Considerable use has been made of global variables; a practice not thought highly of by professional programmers. This technique was employed to make it relatively easy to reconfigure the code, should something change that did not require major redevelopment.

6.1.5.1.2 The mpg321 MP3 Player

The mpg321 MP3 Player is a shell based player which has a number of useful features, many of which are used in this software. The documentation for this tool may be accessed from any Linux computer, which has the tool installed, by typing:

```
 man mpg321
```

in a shell.

The main advantages of this tool for this development were:

- Ability to playback the file in mono format (-m argument).
- Ability to pass a list of files that the tool would play in sequence.
- Ability to loop the playback of the files, so that the music will continue forever, or until stopped (-l argument).
- Ability to direct the output of the player to a specific hardware device. For this software, the speakers in the tower were connected to the audio jack on the Minster Pi 3 and the speaker in the Chancel was connected to an USB Audio Adaptor (-a hw:0,0 argument).

6.1.5.1.3 Python Lists

Lists are a standard feature of Python and they may be studied in detail in the Python Lists documentation. They are mentioned here because it was necessary to concatenate or 'string together' two or more lists in the functions that were passing arguments to the MP3 player described above. List may be recognised in the code because they are bounded by square brackets, eg:

```
 ['mpg321', '-l', '-m', '-a', 'hw:1,0']
```

A good example of this technique in use may be found in the mp3_player_start() function where a list called mpg_list (reproduced above), which contains the call to mpg321 and all its arguments apart from the file(s) to play is concatenated with a list containing all the files in the playlist, see the next Section.

6.1.5.1.4 The glob Library

Also used in this code is the glob Library, which is used to determine the pathnames of the MP3 files to play. It generates a list which is then concatenated with the command and its arguments as described above.

6.1.5.1.5 The Advanced Python Scheduler (APS) Library

The APS Library (apscheduler) is employed to generate trigger events at certain times of the day. This is a very efficient piece of code that basically asks the Linux kernel to wait for a scheduled time and then generates a call to a function to carry out specific tasks.

The schedule is defined in the function chimes_schedule() and basically generates events every 15 mins during the opening hours of the Model Town (eg on the hour and at 15, 30 and 45 minutes past the hour). Provision is made in the function extended_hours_trigger() to include late hours between 5 pm and 10 pm, so the schedule includes events until that time. The functions calculate_hours(), check_silent_hours(), hours(), calculate_quarters() and quarters() then determine if the chimes should sound when triggered and how many.

The apscheduler capability is also used in the wedding_sequence_start() function, but this time it provides a fixed delay of 12 minutes from the point when it is invoked. The scheduled delay is specified at the end of the function, but the execution of that function is completed in a matter of milliseconds due to the use of the subprocess Library (see Sections 6.1.5.1.6 and 6.1.5.1.7 below) which allows the music and chimes to be started immediately so that control may be handed back to the program.

6.1.5.1.6 The GPIO.add_event_detect() Method

This method deserves a special mention as it is part of the process used to allow the software to reliably ring the hours and the quarters while simultaneously playing music or a wedding sequence in the Chancel of the Model Minster.

Once the program has started the functions described in the previous Section all run in the the main thread of the program. When a User operates one of the switches, the event definition associated with the GPIO pin connected to the switch (the first argument in the method) calls the function defined in the second argument in the second (remaining thread) and so the execution is not blocked.

Reference should be made to the Tutorials provided by Alex Eames at raspi.tv for more information about how this feature is used.

6.1.5.1.7 The Subprocess Management Library

The Subprocess Management Library allows for interprocess communication and in this software, the subprocess.call() and subprocess.Popen() methods are used to launch the MP3 player (mpg321) described above. This allowed the single thread environment available to the called function to be extended by using the multi-tasking capabilities of the underlying Linux Operating System.

The main difference between the two methods is that subprocess.call() launches the player as a foreground task and subprocess.Popen() as a background task.

So when a function invokes the player using subprocess.call(), the method does not give control back to the calling function until the MP3 file invoked has completed playing. This method is therefore only used in the main thread to play the hours and quarters and in the second thread to play short messages before launching into the long or non-stop pieces.

If the calling function invokes the player using subprocess.Popen(), the method only takes control until the player responds with its ID. The function is then free to continue processing within the second thread because the player has started a new thread under the main operating system, eg not in Python.

Functions that use this capability store the ID of the spawned player in a global variable, so that it may be terminated if a user operates the appropriate switch.

6.1.5.1.8 The Wedding Sequence Functions

The Wedding Sequence is managed by the wedding_sequence_start() function. This has an extra consideration to take into account, the fact that there is only one Wedding Sequence function (only one can be called from the GPIO.add_event_detect() instance, but it has to manage playing music and a wedding service to the Chancel and the wedding bells to the tower. At the same time, these two instances of mpg321, must be launched and release the thread as soon as possible, so that other events may be detected and processed.

The function achieves this by passing two files to the tower mpg321 process. The first is 5 minutes of silence and the second is the peals of changes and rounds that make up the wedding bells. The silence gives time for the wedding march and vows to complete and the Fugue to begin, before the bells sound.

6.1.5.1.9 Audio Recordings

The following audio recordings have been collected together and are stored on a removable USB Memory Stick plugged into a USB port on the Minster Pi 3. The files are as follows:

- Chimes – Recorded chimes from the real Minster of:
    - Quarter Jack pairs. There are two of these:
        - One pair to reproduce the first and subsequent pair of chimes in a sequence, eg at 15 mins; one pair, at 30 minutes two pairs and so on.
        - One pair to reproduce the last pair of chimes, which ring longer. These ring longer, because the chime is not ‘quenched by the subsequent strike’.
    - Hours. As above there are two of these:
        - One chime to reproduce the first and subsequent chimes in a sequence.
        - One chime for the last in the sequence.
- Playlists 1 to 6, each containing:
    - A short message announcing the identity of the playlist.
    - The music to be played in the chancel with the MP3 Player.
- Change Rings containing:
    - A short message announcing that ‘Change Rings are Enabled’.
    - A single file of Change Rings.
- Wedding Sequence containing:
    - A short message announcing that ‘The Wedding Sequence is Enabled’.
    - Three files; the Wedding march, the Vows and Widor’s Toccata and Fugue.
- Wedding Bells containing:
    - Approximately 5 minutes of silence.
    - A single file of Changes and Rounds (the Wedding Bells).
- Messages containing:
    - Voice recording (Late Hours Enabled’.
    - Voice recording (Late Hours Disabled’..

6.2 Deployment

6.2.1 Software Environment

The deployed image on the SD Card is the same as that used during development, except that the system is run in ‘read-only root’ mode. Details of how this may be done are given in the Raspberry Pi Forums entry Readonly Root with OverlyFS. This involves marking some areas of the system (such as the swap partition) as Read-only in fstab and also running a script to create a protected run-time environment.

6.2.2 Auto-running the Code at Boot-up

There are a number of ways to auto-run code when the Raspberry Pi boots up and for this development the method described at ‘How To Autorun A Python Script On Raspberry Pi Boot’ was chosen. This is undoubtedly the simplest approach and simply involves writing a short script to invoke the software, setting the Operating System to log in automatically and then adding a line to the file /etc/profile.

6.3 Backing Up and Restoring SD Cards

There are several methods of backing up SD Cards. Two are described below:

6.3.1 Backing Up

6.3.1.1 From Within Raspbian

Boot the system in the normal way, but interrupt execution of the minster_bells program by pressing CTRL-C several times. Start the Pixel Desktop by typing ‘startx’ at the prompt and pressing <RET>.

Once the desktop is running, insert a blank SD Card into a USB to SD Card adaptor and select Accessories – SD Card Copier from the main Menu. Follow the on-screen instructions.

Note: This method can be a bit slow because the Pi has a relatively slow processor, which has to manage the copying at the same time as running the GUI.

6.3.1.2 From a Linux Computer

6.3.1.2.1 Identify the Device

In order to carry out this operation, it is necessary to identify the mounted SD Card. Insert the SD Card into a USB to SD Card adaptor, but do not plug it into the Linux machine yet.

Issue the following command in a shell:

```
df -h
```

```
Filesystem      Size  Used Avail Use% Mounted on
udev            3.9G     0  3.9G   0% /dev
tmpfs           794M  9.5M  785M   2% /run
/dev/sda1        58G   18G   38G  32% /
tmpfs           3.9G   38M  3.9G   1% /dev/shm
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           3.9G     0  3.9G   0% /sys/fs/cgroup
/dev/sdb1       286G  160G  112G  59% /home
tmpfs           794M   36K  794M   1% /run/user/1000
```

Insert the SD Card into a USB slot and re-issue the df command.

```
df -h
```

```
Filesystem      Size  Used Avail Use% Mounted on
udev            3.9G     0  3.9G   0% /dev
tmpfs           794M  9.5M  785M   2% /run
/dev/sda1        58G   18G   38G  32% /
tmpfs           3.9G   38M  3.9G   1% /dev/shm
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           3.9G     0  3.9G   0% /sys/fs/cgroup
/dev/sdb1       286G  160G  112G  59% /home
tmpfs           794M   36K  794M   1% /run/user/1000
/dev/sdc2       6.8G  5.4G  1.1G  84% /media/terry/7f593562-9f68-4bb9-a7c9-2b70ad620873
/dev/sdc1        63M   21M   43M  34% /media/terry/boot
```

Two new devices are now visible sdc1 and sdc2. These are the two partitions on the Raspbian SD Card; it is the device that is of interest (eg, sdc) not the full partition name.

6.3.1.2.2 Writing to an Image

To backup up an SD Card in Linux, issue the following commands in a shell to backup an SD card:

```
sudo dd if=/dev/sdc of=~/SDCardBackup.img
```

Note1: The source of the backup, (in the example sdc) should be changed to suit the Linux installation in use.

Note2: The destination may be anywhere on the Linux installation and it is suggested that the filename reflects the contents and the date of the backup.

Note 3: The dd command does not show any feedback so wait until the command prompt re-appears.

6.3.2 Restoring

6.3.2.1 Using Etcher

Etcher is a cross-platform graphical SD Card writing tool which takes much of the complexity out of writing an image to an SD Card. The tool may be downloaded from the Etcher Website.

6.3.2.2 From a Linux Computer

After establishing the identity of the device it needs be unmounted. However, this must be done partition by partition by using the partition number as well. Issue the following commands for each partition:

```
sudo umount /dev/sdc1
sudo umount /dev/sdc2

etc….
```

Write the original image to the SD Card:

```
sudo dd bs=4M if=~/SDCardBackup.img of=/dev/sdc
```

Note1: The bs=4M option sets the 'block size' on the SD card to 4Meg. If you get any warnings, then change this to 1M instead, but that will take a little longer to write.
Note2: Wait until the command completes because ‘dd’ provides no feedback on progress.

Before ejecting the SD card, ensure that the Linux machine has completed writing to it by issuing the command:

```
sudo sync
```

See The Pi Hut for more information, including details of how to backup and restore SD cards on Windows or a Macintosh.

ANNEX A

DESIGN COMPROMISES

Hardware

The original design utilised a Raspberry Pi Zero for the Minster Functions. Although this is a single-core computer, has less memory and is clocked at a lower speed than the Pi 3, it was adequate for the job, except in one aspect. Unlike the bigger Pi computers, the Pi Zero only has one USB port, which is connected to the same kind of chipset as a mobile phone. These USB ports are not designed to be expanded into multiple instances through a hub; they will work but there are limitations.

The Minster Functions needed two audio channels and the original idea was to use two USB Audio Adaptors, because the Pi Zero has no audio jack. This proved impractical, because the two adaptors caused conflicts triggering unreliable behaviour and crashes on occasion.

Software

In the Requirements Specification each switch event was intended to be accompanied by a vocal message. However, it was found that the The GPIO.add_event_detect() Method had a serious limitation in that the detect RISING event functionality was in fact a detect HIGH event capability; a problem which necessitated more complex debounce circuits to slow down the rise time. This limitation made it very difficult to include the playing of very short MP3 files and the result was that the message was played multiple times. Only the Extended Hours switch is therefore provided with these messages because, unlike the other switches, there is no other indication that the event has occurred.