WildlifeSystems - biodiversity technologies

Ed Baker

2024-08-08

Contents

About

This book explains the technologies developed as part of WildlifeSystems and how they can be implemented in real-world scenarios.

Support

WildlifeSystems nodes were originally developed and used as part of the Leverhulme Trust funded Automated Acoustic Observatories project at the University of York. Additional development is undertaken as part of the Urban Nature Project at the Natural History Museum, London.

Chapter 1

Biodiversity Technologies

1.1 What are Biodiversity Technologies?

Biodiversity technologies are tools that allow researchers and others to study, monitor and conserve biodiversity. These tools can be used to monitor a wide range of species, habitats, and ecosystems, potentially in (near) real-time and/or from a great distance away.

1.1.1 What has enabled Biodiversity Technologies?

The development of biodiversity technologies has been enabled by the rapid advances in sensor technology, data processing, and communication networks. These technologies have made it possible to collect, store, and analyse large amounts of data from a wide range of sources, including remote sensors, cameras, and acoustic recorders.

This has been accompanied by a decline in the cost of these technologies (e.g. Figure 1.1), making them more accessible to researchers, conservationists, and others interested in monitoring biodiversity.

These technologies can potentially create vast datasets, and in order to adhere to the FAIR principles (Findable, Accessible, Interoperable, Reusable), it is important for the data to be accompanied by an appropriate suite of data standards, such as those developed by TDWG.

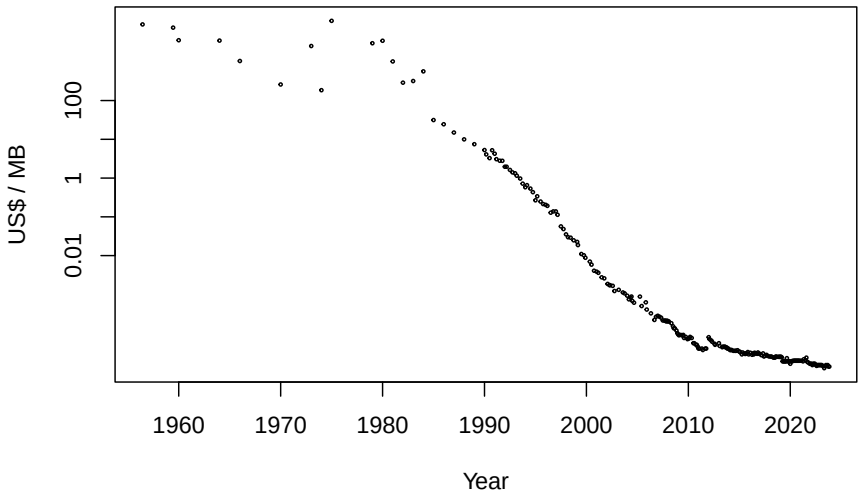

Figure 1.1: The decline in storage costs per megabyte in consumer hard drives.

1.2 Overall Philosophy

1.2.1 Leverage what already exists

Many of the tools that are needed to monitor biodiversity already exist, and the goal of WildlifeSystems is to bring these tools together in a coherent and integrated way. This will allow researchers and others to build on existing technologies to develop new tools and applications that can be used to monitor biodiversity in a wide range of environments.These tools are designed to be flexible, scalable, and easy to use, and to provide a platform for the development of new technologies and applications.

1.2.2 Innovate where necessary

"Technology is stuff that doesn't work yet." — Bran Ferren

Where suitable technological solutions already exist, use them. Limit technological research to what needs to be new. Most of the time it's OK to use existing technology, and focus development on the biodiversity related problem you're trying to solve. Older technologies are often better understood and more stable. Some of the technologies WildlifeSystems leverages to innovate in the biodiversity sphere are given in Table 1.1.

Table 1.1: Age of technologies underpinning WildlifeSystems.

Technology	Year Introduced	Age in 2024
Ethernet	1973	51
BASH	1989	35
Linux	1991	33

1.2.3 Plan for heterogeneity

Any network of devices will become heterogeneous given sufficient time. The primary driver of this shift to heterogeneity is continuous technological innovation combined with obsolescence of technical components.

Secondary drivers are changes in the purpose, capabilities, and/or scale of the network over time.

1.2.4 Open Source

1.3 Structure of Wildlife Systems

1.3.1 Packages

Chapter 2

Sensor Networks

Chapter 3

Environmental Sensors

3.1 How sensors work

3.1.1 Temperature

3.1.2 Humidity

3.1.3 Air Pressure

3.1.4 Gases

3.1.4.1 Heated Gas Resistance

Chapter 4

Sensors in WildlifeSystems

The WildlifeSystems platform comes with support for some popular existing environmental sensors, although there are many on the market and the range available is subject to constant change. The modular nature of WildlifeSystems allows for new sensors to be easily integrated if the need arises.

As many physical packages contain multiple sensors (e.g. the supported BME680 can monitor temperature, humidity, air pressure and air quality) there is a conceptual difference between a "device" and the one or more sensors on a device.

4.1 Devices included in the base system

The Raspberry Pi does not come with environmental sensors, however there are several onboard sensors that are used to monitor the operation of the hardware, to prevent crucial components from overheating, including the temperature of the CPU and GPU chips. WildlifeSystems provides access to these sensors through the `sensor-onboard` package, as well as providing some *software sensors* that report the free memory and free SD storage available. These can be useful for detecting and resolving possible issues on a sensor node before a serious problem arises.

4.2 Installing sensor support

Support for sensors is installed as part of the node installation process, however it is possible to install the `sensor-control` abstraction layer onto any Raspberry Pi system using the command below.

```
wget -O - https://raw.githubusercontent.com/Wildlife-Systems/sensor-control/main/instal
```

This will install the `sensor-control` and `sensor-onboard` scripts into the system, as well as installing a small of number of supporting packages if they are not already installed.

4.3 Reading data from a device

The sensor read command, `sr`, can be used to list the available devices on a given sensor.

```
sr onboard list
```

The sensor read command, `sr`, can also be used to read data from a named sensor. Not specifying a sensor is equivalent to reading all sensors on that device (i.e. using the wildcard"*").

```
sr onboard
sr onboard *
sr onboard onboard_gpu
```

This will give a JSON string listing information about each sensor (or just the specified sensor), and the current reading. This information can be presented in a more human readable form by piping the output to the program `jq`, a command line JSON processor.

```
sr onboard | jq
```

4.4 The sensor reading process

The sensor reading process in WildlifeSystems has five steps.

1. The `sr` command identifies which device script to route the request to.
2. The sensor script calls the `sc-prototype` script to obtain a template ("prototype") of the JSON request.
3. The sensor script accesses the relevant sensor(s) and populates the values in a template for each sensor reading, before returning a JSON array of populated readings to `sr`.
4. The `sr` script populates additional information for each reading, providing the `node_id` and a `timestamp` for each.
5. `sr` returns the finalised JSON array to the user.

4.5 Installing new sensors

Chapter 5

Implementing new sensors

New sensors should be implemented as new packages (i.e, a new GitHub repository). Packages have a standard format, and should be named `sensor-<sensor-name>` where `<sensor-name>` is short, descriptive of the sensor (e.g. model number), and unique within the WildlifeSystems ecosystem.

The structure of a basic package is given below.

```
.
+-- inst
|   +-- files that are not part of the package structure
|       used during installation (e.g. 3rd party scripts
|       to be copied to /usr/bin/)
|
+-- sensor-<sensor-name>
|     Executable to read sensor and print JSON of readings
|     (copied on install to /usr/bin/)
|
+-- <sensor-name>
|     Configuration file
|     (copied on install to /var/aao/sensors/)
|
+-- install
|     Bash file run once on install - used to install packaged
|     dependencies, etc.
```

As many people implement various sensors on the Raspberry Pi, it is likely that some sort of solution is already available, that can be tweaked to output readings in the standard format required. However, you must ensure that any licensing conditions are met. In particular, an open license is required if submitting your sensor package to WildlifeSystems for inclusion in the ecosystem.

5.1 Reading the sensor

File `sensor-<sensor-name>` provides the functionality to read the protoype JSON, populate the JSON with sensor readings, and print the output.

The file can be written in any programming or scripting language, but to prevent overhead consideration should be given to minimising the number of new packages installed.

5.1.1 In bash

```
#!/bin/bash

# Read the prototype JSON
JSON=$(sc-prototype)

echo -n "["

# Code to read the sensor value into GPU_TEMP

# Use `jq` to modify JSON
SENSOR=$(echo $JSON | jq ".sensor |= \"onboard_gpu\" | .measures |= \"temperature\" |
echo -n $SENSOR

echo "]"
```

5.1.2 In Python

```
#!/
import os
import json

# Read the prototype JSON
stream = os.popen('sc-prototype')
output = stream.read()

# Pre-populate with sensor metadata
temperature = json.loads(output)
temperature["sensor"] = "bme680_temperature"
temperature["measures"] = "temperature"
temperature["unit"] = "Celsius"
```

```
# Code to read sensor and output in variable `sensor_reading`
temperature["value"] = sensor_reading

# Output the JSON in an array
print("[",json.dumps(temperature),"]")
```

5.2 Setting the environment

The file `<sensor-name>` in the package specifies information that modify the environment of the Raspberry Pi (e.g. if the i2c interface should be enabled) before the `sensor-<sensor-name>` script is run. This allows sensors with different requirements to run sequentially on the same node.

The file must always be present, even if it is empty. During installation the file is moved to `/var/aao/sensors/` and the list of files in this directory indicates to the system which sensors are installed.

TODO: i2c

5.3 Install

The file `install` in the directory is run once, when the sensor package is installed. This allows for the installation of packages and scripts necessary for the functioning of the package.

The file is executed by `bash` and the use of `sudo` is allowed.

5.4 The installation process with `si`

The sensor install script, `si`, from `sensor-control` is used to install sensor packages. For developer reference the installation process is described below.

1. `si` clones the `Wildlife-Systems/sensor-<sensor-name>` repository from GitHub.
2. The `install` script is executed.
3. `sensor-<sensor-name` is marked as executable and move to /usr/bin/.
4. `<sensor-name>` is moved to /var/aao/sensors/.
5. The cloned repository is removed from the local filesystem.

5.5 Submitting packages to WildlifeSystems

Submitting packages (where licensing allows) to WildlifeSystems allows the ecosystem to be developed and sustained collaboratively by the user community.

Packages can be sent to the administrators as a compressed file, or a request can be sent to fork an existing GitHub repository. Contact details can be found at https://wildlife.systems/contact.html.

Chapter 6

Sound Devices

6.1 How sound devices work

Chapter 7

Sound devices in WildlifeSystems

Chapter 8

Imaging Devices

8.1 How imaging devices work

Chapter 9

Imaging devices in WildlifeSystems

Chapter 10

Power Management

Power management on the Raspberry Pi is useful when deployments are made that are powered by batteries and/or renewable sources such as solar power that are intermittent.

In addition, there are small environmental benefits on consuming less power on systems which have continual grid power.

The `pi-pwr` script can be used to turn off unused functionality, either always or just when it is not required.

10.1 Installation of power management tools

The power management tools are installed as part of the node installation process, however they can be easily installed independently on any Raspberry Pi system.

```
wget -O - https://github.com/wildlife-systems/pi-pwr/raw/master/install | sudo bash
```

10.2 Turning funtionality on and off

10.3 Considerations

Disabling all network functionality will prevent the node from communicating until either the functionality is turned back on or the Raspberry Pi is restarted.

If disabling all connectivity is desired periodically then the functionality to turn these systems back on must be scripted.

10.3.1 A note on `sudo`

Raspberry Pi OS (and previously Raspbian) allows the default user to run sudo without a password. This is not true for other Linux distributions, such as Ubuntu. This could lead to a password prompt when using `pi-pwr`. As nodes are designed to run autonomously, the installation process for `ws-node` will configure `pi-pwr` to not require a `sudo` password.

Chapter 11

Indicators and heartbeats

The script `ws-indicate` is used to indicate the device's status using the LED(s) on board the Raspberry Pi. The script `ws-heartbeat` can be used to transmit the device's status to a user-defined script that could provide logging, or submit the status to an online dashboard.

11.1 Installation of `ws-indcate` and `ws-heartbeat`

These tools are installed as part of the node installation process.

11.2 Indicators

Internally `ws-inidicate` makes repeated calls to `pi-pwr` to control the LED(s). There are three indicator routines, heartbeat (quick flash of LEDs in order to show device is functioning), countdown (flashes power LED), and record (power LED on, action LED off).

```
sudo ws-indicate
sudo ws-indicate countdown 5   # counts down from 5
sudo ws-indicate record action # record light is on while action is executed
```

11.2.1 A note on `sudo`

Raspberry Pi OS (and previously Raspbian) allows the default user to run sudo without a password. This is not true for other Linux distributions, such as Ubuntu. This could lead to a password prompt when using `ws-indicate`. As

nodes are designed to run autonomously, the installation process for `ws-node` will configure `ws-indicate` to not require a `sudo` password.

11.3 Heartbeat

The script `ws-heartbeat` is used to send a heartbeat signal to the devices.wildlife.systems server to indicate that the node is alive and connected, as well as to provide some information to assist problem diagnosis. The data sent comes from the onboard sensor readings (`sr onboard`) and the server stores the node ID, timestamp, CPU and GPU temperatures, and the amount of memory and storage used. The node must be registered with WildlifeSystems for this data to be stored.

The script may be run for debugging purposes at any time as follows.

```
ws-heartbeat
```

The script will exit silently on success.

Chapter 12

Developer Guidelines

12.1 Documentation

There are three main kinds of documentation in the WildlifeSystems project.

1. **Code comments**: Primarily for future you (or someone similar) to get to grips with your code. *Why does this code do this?*.
2. **Package documentation**: Describes what your package does, how it interacts with the larger system. *What will this package do for me?*
3. **Overall project documentation**: This manual. *How do I use the entire system for my research?*

Chapter 13

Server Tools

These tools are only needed for running and maintaining the wildlife.systems webserver.

13.1 Installation

```
git clone git@github.com:Wildlife-Systems/wildlife.systems-tools.git
cd wildlife.systems-tools
./install
cp .ws-db.php ~/

#Edit ~/.ws-db.php to connect to the wildlife-systems database
```

13.2 Adding nodes and receieving a token

```
ws-node-add <node_id>
```

Adding a node by it's ID (serial number of the Raspberry Pi) will add the device to the wildlife.systems database, generate a token, and display the token (a UUID) in the console. This token is used to authenticate the device when it sends data to wildlife.systems.

13.3 Removing a node

```
ws-node-remove <node_id>
```

Appendix A

Return codes

The various scripts that form the WildlifeSystems ecosystem use a standard set of return codes.

A.1 00-09 Script functionality

Code	Label	Description
0	OK	Terminated normally. No error.
1	Already running	The script determined it was already running and terminated.
2	No arguments	The script requires arguments but none given. Will give help text.

A.2 10-19 Parameter problems

Code	Label	Description
10	Invalid argument	One or more of the arguments to the script was invalid.
11	Incorrect filename pattern	A standard filename pattern was expected. Allowed values are 'timestamp'.

A.3 20-29 Sensor problems

Code	Label	Description
20	Unknown device	This device is not supported, or software is not installed.
21	Unknown sensor	This sensor is not known on this device.

A.4 30-39 Sound device problems

Code	Label	Description
30	Unsupported device	The specified device is unsupported.
31	Unsupported feature	This feature is not present on the sound device.
32	Unknown feature	The feature requested is unknown.

A.5 40-49 Image device problems

Code	Label	Description
40	Unsupported device	The specified device is unsupported.
41	Unsupported feature	This feature is not present on the sound device.
42	Unknown feature	The feature requested is unknown.

A.6 50-59 Power management problems

Code	Label	Description
50	Disallowed value (should be on or off)	The specified device is unsupported.
51	Disallowed value (should be on, off or default)	This feature is not present on the sound devi

A.7 60-69 Special meanings

Code	Label	Description
60	Identify as a sensor device script	Returned by sensor device script in response to first param