WildlifeSystems - biodiversity technologies

Ed Baker

2026-01-13

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.

```
## Warning in xy.coords(x, y, xlabel, ylabel, log): NAs introduced by coercion
```

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.

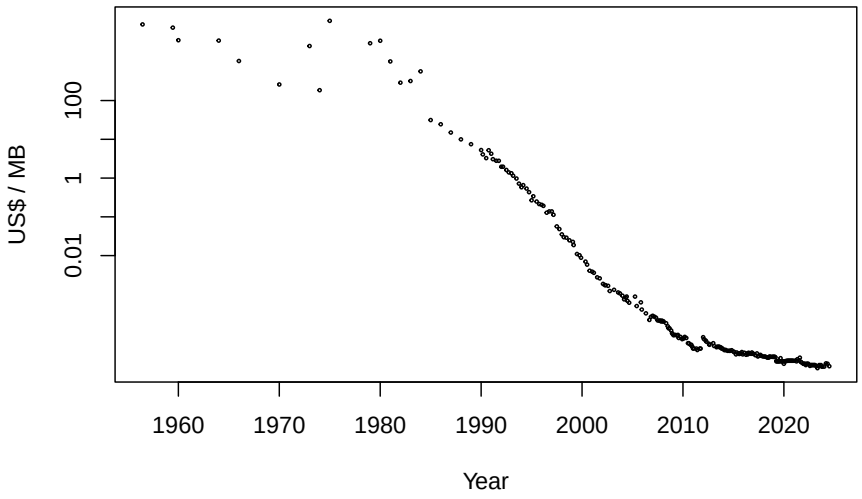

Figure 1.1: The decline in storage costs per megabyte in consumer hard drives.

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 2026
Ethernet	1973	53
BASH	1989	37
Linux	1991	35

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.

The WildlifeSystems ecosystem is designed to be able to handle this heterogeneity, and to allow for the easy integration of new technologies as they become available.

1.2.4 Modularise

The WildlifeSystems ecosystem is designed to be modular, with each component able to be used independently or in combination with other components. This allows users to build custom solutions that meet their specific needs, and to easily add or remove components as required.

Defined interfaces between components allow for easy integration of new technologies, and for components to be replaced or upgraded without affecting the rest of the system. This also lets contributors to focus on what they are best at.

1.2.5 Open Source

1.3 Structure of Wildlife Systems

1.3.1 Packages

Chapter 2

Nodes

The fundamental unit of the WildlifeSystems platform is the node. A node is a physical device that can be used to collect data from sensors, control devices, and communicate with other nodes. Nodes are (typically) Raspberry Pi computers, packaged with an array of environmental sensors, microphone(s), and/or camera(s).

The WildlifeSystems software is used to configure a node to collect data from the sensors and what to do with the data collected. Typically this will be to send the data for external storage and analysis, but it is also possible to store the data locally on the node. Data can also be processed locally on the node, for example to detect a specific event (e.g. a sound or image) and then send a notification to the user. The WildlifeSystems software is designed to be modular, so that new sensors and devices can be added easily, and the data processing can be configured to suit the user's needs.

2.1 Standard node

There is no such thing as a 'standard node', apart from the core software interfaces. In some deployments, however, there will be a great deal of standardisation. The Urban Research Station deployment at the Natural History Museum, London, for example, has a number of nodes that are identically configured. Although even here, there are differences in the sensors used, and the data processing that is done. The nodes are all configured to send data to the same server (the Data Ecosystem) but the exact configuration of each node is driven by current research needs. In this way a network of WildlifeSystems nodes can be seen as a platform for facilitating research, rather than as a single-purpose device.

To make it easier to deploy and work with individual nodes, it is recommended

that some commonly used sensors are deployed in the stadard fashion described below.

2.1.1 Self-monitoring

Electricity and water should not mix. While suitable enclosures go a long way to preventing an unfortunate meeting, many nodes are deployed outside in the elements. The addition of a cheap DHT11 sensor internally to monitor the humidity and temperature of the enclosure may help identify a potentially damaging situation before it becomes problematic. [TODO: See PRTG integration].

This sensor is recommended to be attached to GPIO pin 4.

2.1.2 i2s microphones

TODO

2.1.3 1-Wire bus

GPIO 17

2.2 Enclosures

2.2.1 Fixing inside enclosures

2.2.2 Attaching sensors

TODO: to sockets, not directly

TODO: breakout boards

TODO: custom boards

2.2.3 Power solutions

TODO: PoE

Chapter 3

How nodes work

After instllation configuration files are placed in `/etc/ws` that define the operation of the node.

On startup, WildelifeSystems nodes run three systemd services that are responsible for processing audio, images, and environmental sensors. These sensors read the appropriate configuration files, and start collecting and recording data.

3.1 Audio

The service `ws-run-audio` is configured by `/etc/ws/audio.conf` and is responsible for running the audio processing pipeline. The blocks in the pipeline are `record`, `sleep`, `upload-delete`, and `clearup`.

3.1.1 `init`

3.1.2 `run`

3.1.2.1 `record`

3.1.2.2 `sleep`

3.1.2.3 `upload-delete`

3.1.2.4 `clearup`

In situations where network connection is temporarily unavailable it is possible that audio files will collect on the advice. These are not automatically processed

during `upload-delete` and their processing must be handled manually, or by creating a processing scipt in `clearup`.

Chapter 4

Sensor Networks

Chapter 5

Environmental Sensors

5.1 How sensors work

5.1.1 Temperature

5.1.1.1 Thermistors

A thermistor is a type of resistor whose resistance changes significantly with temperature. The word is a portmanteau of thermal and resistor.

5.1.2 Humidity

5.1.2.1 Capacitive Humidity Sensors

A capacitative humidity sensor consists of a small capacitor with a hygroscopic dielectric material between two electrodes. The dielectric material, usually a plastic or polymer, has a low dielectric constant when dry. In the presence of moisture, the dielectric constant increases due to water vapor, which has a much higher dielectric constant. As moisture is absorbed, the sensor's capacitance increases. The amount of moisture absorbed depends on the surrounding temperature and water vapour pressure, which also affects the hygroscopic dielectric material used in the sensor.

5.1.3 Air Pressure

5.1.4 Gases

5.1.4.1 Heated Gas Resistance

Heated gas sensors are used to detect gases in the air. The sensor consists of a metal oxide semiconductor that is heated to a high temperature. Volatile organic compounds (VOCs) in the air change the resistance of the sensor.

These sensors are typically used indoors and have a short lifespan (perhaps two years). As such they are rarely used in environmental sensing applications, but are included on some common sensors such as the BME680.

Chapter 6

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.

WildlifeSystems takes an abstracted approach to sensors. Interface modules are provided for each sensor, which can be used to read the sensor data into a standard format.

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.

6.1 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.

6.2 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.

```
sudo apt install sensor-control
```

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.

6.3 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
```

6.4 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.

6.5 Installing new sensors

New sensors can be installed using the apt package manager once the Wildlife Systems APT repository has been added to the system. Instructions can be found on the webpage Configuring APT for Wildlife Systems..

```
sudo apt install sensor-<sensor-name>
```

Chapter 7

Supported sensors

7.1 List of supported sensors

7.1.1 BME680

4-in-1 sensor that provides humidity, temperature, pressure, and gas readings.

7.1.2 DS18B20

The DS18B20 is a digital thermometer that provides Celsius measurements over a 1-Wire interface. It is a low-cost, low-power device that can be used for a variety of environmental monitoring applications. Waterproof versions are available, making it suitable for use in water and soils.

```
sudo apt install sensor-ds18b20

sudo sr ds18b20
```

7.1.3 DHT11

The DHT11 is a low-cost temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and sends the data by digital signals on the data pin.

The low-cost nature of the sensor makes it useful desp ite several shortcomings, e.g. for monitoring the temperature and humidity inside electronics enclosures where accuracy is not critical but the sensor can provide critical early warning of overheating or condensation.

```
sudo apt install sensor-dht11

sudo sr dht11
```

7.1.4 LV-MaxSonar-EZ

A series of ultrasonic distance sensors that can measure distances from 2 cm to 4 m with high accuracy. They are commonly used in robotics and automation applications, but can also be used for environmental monitoring.

```
sudo apt install sensor-hrlv

sudo sr hrlv
```

Chapter 8

Implementing new sensors

New sensors should be implemented as new Debian packages, and submitted via GitHub. 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.

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.

8.1 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.

8.1.1 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 "]"
```

8.1.2 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),"]")
```

8.2 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 9

Sound Devices

9.1 How sound devices work

Chapter 10

Sound devices in WildlifeSystems

10.1 Installing sound device support

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

```
`wget -O - https://github.com/wildlife-systems/sound-device-control/raw/master/install | sudo bas
```

The installation will install the `alsa-utils` package on Ubuntu/Debian systems, and add the current user to the `audio` group.

10.2 Supported sound devices

The WildlifeSystems platform supports several popular sound capture and playback devices.

10.2.1 Raspberry Pi onboard audio

10.2.2 AudioInjector Zero

10.2.3 AudioInjector Octo

10.2.4 AudioInjector Ultra

10.2.5 I2C microphone (e.g. Google Voice Hat)

Limited to 48kHz sample rate, no record volume control.

10.2.6 Respeaker 6 mic array

10.2.7 Audio+ DAC

10.2.8 Wolfson Audio Card

10.3 Installing a sound device

To install support for a sound device, run the following command.

```
sdc-inst <sound-device-name>
```

To get a list of names of supported devices, run the following command.

```
sdc-inst list
```

Chapter 11

Imaging Devices

11.1 How imaging devices work

Chapter 12

Imaging devices in WildlifeSystems

Chapter 13

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.

13.1 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
```

13.2 Turning funtionality on and off

13.3 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.

13.3.1 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 14

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.

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

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

14.2 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
```

14.3 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 15

Integration with monitoring tools

15.1 PRTG

As the devices are Raspberry Pi based, we can use the PRTG monitoring tool to monitor standard aspects of their performance including CPU, memory, disk space, and network traffic.

15.1.1 Monitoring custom sensors

Some sensors attached to WildlifeSystems nodes may be used to monitor aspects of the devices themselves, rather than the environment. Using custom PRTG scripts it is possible to include these measurements in PRTG reports and alerts.

The following example reports the humidity of the device enclosure measured by a DHT11 sensor. It makes use of the WildlifeSystems `sr` command to read the sensor in a standard JSON format, and converts the reading to a format expected by PRTG.

```
#!/bin/bash

HV=$(sudo sr dht11 | jq 'map(select(.sensor == "dht11_humidity")) | .[]["value"]')

if (( $HV < 75 )); then
  /bin/echo "0:$HV:Acceptable humidity"
else
  /bin/echo "4:$HV:Too humid"
fi
```

Chapter 16

Developer Guidelines

16.1 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 17

Server Tools

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

17.1 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
```

17.2 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.

17.3 Removing a node

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

Appendix A

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 scripts such as `sr` and `ws-indicate`. As nodes are designed to run autonomously, the installation process for `ws-node` will configure these scripts to not require a `sudo` password.

In some installations (generally where the devices are managed by organisations) the `sudo` group will not be present. To allow running without a password the file `/etc/sudoers.d/ws` can be renamed (to avoid changes being undone if `ws-node` is re-installed) and the `%sudo` group modified to an appropriate group present on the system.

Appendix B

Return codes

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

B.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.
3	Config file error	The config file is missing or malformed.

B.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'.

B.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.

B.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.

B.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.

B.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

B.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