ACS - Miscellaneous Ground utilities

This document gives an overview of various ground utility software collected in the `scripts` directory of `acs-env`. These utilities are placed during a standard installation (see `install.md`) into `~/.local/bin/`, which is added to `$PATH`. Scripts that are only used internally may be omitted from the list below.

Wireless configuration

`telem_config.sh`

Configures a locally-attached or on-aircraft serial telemetry radio. See `telemetry.md` for details.

`wifi_config.sh`

Configures a locally-attached 802.11 wireless interface to communicate on an aircraft network. See `wireless.md` for details.

ACS network messaging

`acs_net_dump.py`

This utility is a rough equivalent of `tcpdump` for ACS messages. The simplest and most common usage is:

```
acs_net_dump.py -d wlan0
```

where `wlan0` is an interface configured to receive traffic from an aircraft network (see `wifi_config.sh`). It outputs information from the message header in a readable format. For message types that have parsing code added in the utility, it will output the values from the message payload; for others, it will output the header only.

`acs_net_logger.py`

This wraps the `autonomy-payload` library `acs_logger.py` and provides an interface for recording and playing back ACS message traffic. To record traffic arriving at an interface (e.g., `eth0`):

```
acs_net_logger.py record eth0 ~/acs.log
```

Note that it will only record traffic sent to the subnet-broadcast address (e.g., `192.168.2.255`); traffic sent to unicast and universal broadcast addresses will not be received or recorded.
Likewise, playback of a file through an interface is done using:

```bash
acs_net_logger.py play wlan1 ~/acs.log
```
Messages sent by playback maintain their original headers and payloads, and playback is real-time (i.e., the timing between messages is maintained). As a consequence, the timestamp in the message header may not match the time at which the message is actually replayed.

### `acs_net_repeater.py`

This utility is actually not ACS messaging specific, but is rather a general UDP utility. It forwards UDP datagrams between broadcast devices and/or endpoint pairs.

For example, to forward datagrams between broadcast devices eth0 and wlan1:

```bash
acs_net_repeater.py -D eth0 -D wlan1
```
By default, port 5554 is forwarded. This can be altered with the -P option. Note that a single instance of the repeater will only forward for a single broadcast port, and that port will be the same for all broadcast addresses.

It can also be used to forward multiple ports across an IP endpoint pair. (Note: the underlying class can support multiple IP endpoint pairs, but this is not currently exposed by the CLI.) For example, to forward between ports 5555, 5556, and 5557:

```bash
acs_net_repeater.py -n 3
```
The default "base" port is 5555, and -n specifies the number of sequential ports to use. Option -b sets the base port. The default repeater-side IP is 127.0.1.1 and the default remote-side IP is 127.0.0.1; this is set up to support multiple SITL payload instances communicating across a local loopback interface. (Note: this method of connecting SITLs is deprecated in favor of virtual network bridges, but the utility is still useful for testing configurations.) Options -p and -r set the remote (payload) and repeater IP addresses, respectively.

These options can be mixed and matched in a variety of ways.

### `acs_net_stat.py`

Displays statistics about messages received from payloads (or other ground systems). Useful for quickly seeing what systems are online and the rates at which messages are being observed.

Example:

```bash
acs_net_stats.py -d wlan0
```
Use '--help' or '-h' to see all options; can control length of rolling average and offline timeout.
MAVLink messaging

mavdump.py

This utility provides a very simple way of checking for valid MAVLink protocol traffic over a serial connection. Each second, it outputs the count of MAVLink messages observed, the total number of bytes in those messages, and an estimate of the percent utilization of the serial channel, based on baud rate).

Example:

`mavdump.py /dev/ttyUSB0 57600`
Note that this utility is used by `telem_config.sh` to check for a connection to a remote radio before and after reconfiguration.

Mission monitoring

acs_mission_stat.py

This utility is a very simple catch-all for multi-aircraft information. It displays numbers of powered, ready, and airborne aircraft; aircraft that are currently landing; and aircraft with low battery levels.

Example:

`acs_mission_stat.py -d wlan0`
Note that the implementation is rough, so quirks should be expected.

Mission commanding

acs_heartbeat.py

Periodically broadcasts a heartbeat, consumed by the payload and used to indicate nominal link conditions to the autopilot.

Example:

`acs_heartbeat.py -d wlan0`

acs_net_mavlink_conn.py

Opens (or closes) a mavlink connection via the payload. Right now, connections should be UDP (TCP doesn't
seem to work for providing a server, and there are no additional serial channels onboard).

Example (from a groundstation at 192.168.2.123 on interface wlan0 running e.g. MAVProxy on UDP port 1234, open a UDP connection on aircraft 5 (192.168.2.5)):

```bash
acs_net_mavlink_conn.py --device wlan0 --target 5 --enable udp:192.168.2.123:1234
```

Use `--help` for all options. Use `--enable CHANNEL` to open a channel, and `--disable CHANNEL` to disable it.

## Mission generation

**update_templates.py**

Generates "template" files used for stack-and-slot mission configuration during aircraft launch. See [mission.md](#) for details.

## Aircraft updating

**setupPixhawkPlane.sh**

Utility to update and configure autopilots with the latest firmware. See [deploy.md](#) for details.

**updateOdroidPayload.exp**

Utility to update and configure payloads with the latest software. See [deploy.md](#) for details.

## Log retrieval

**fetchOdroidLogs.exp**

Automates the retrieval of payload ROS bags via `scp`. See [logs.md](#) for details.

## Simulation-In-The-Loop (SITL)

**init_n_sitls.bash**

Prepares files needed to run SITL instances on a local computer. See [sitl.md](#) for details.
multi-sitl-start.bash

Starts one or more Simulation-In-The-Loop (virtual) aircraft on the local computer. See `sitl.md` for details.

multi-sitl-cleanup.bash

Stops all running SITL instances on the local computer. See `sitl.md` for details.