sACN Library Usage Instructions

April 2020

Installation

The library can be installed by putting the Lib folder within an accessible path and then adding sacn as dependency within the cargo.toml of your project.

[dependencies]
sacn = { path = "../Lib"}

The library can then be accessed within your project using an extern crate import

extern crate sacn;

Usage Documentation

The documentation demonstrating usage of the library is included in the docs/sacn folder. This documentation can be re-generated using the cargo doc command. This command includes the first argument to not document external dependencies and the second argument to include documentation of non-public items. If you are just utilising the library rather than adding features then the second argument can be omitted to reduce the documentation size. The last argument opens the documentation automatically after generation.

cargo doc --no-deps --document-private-items --open

Running the demo programs

The demo programs which are used to demonstrate the library in use as-well-as for the tests which require them can be built and run using the commands. These commands should be run from inside the sACN folder.

cargo run --bin demo_src <interface_IP> <source_name>
cargo run --bin demo_rcv <interface_IP>

Where:
<interface_IP> is the IP of the interface that the demo program should use.
<source_name> is the name of the source within the sACN packets.

To see the usage instructions for the demo programs once running type 'h' to display the help.

Testing the library - Compile Errors / Warnings

To check that the library compiles as expected without errors or warnings (none should be expected) run:

cargo check

Testing the library - Unit Tests

The library unit tests are created using the standard rust unit testing system and can therefore be run directly from cargo.

cargo test

Testing the library - Single Machine Integration Tests

The single machine integration tests are also run from cargo but require some machine configuration. The machine should be setup to use multiple IP addresses within the same subnet with the defaults required for the test being *"192.168.0.6", "192.168.0.7"* and *"192.168.0.8"* for IPv4
and "2a02:c7f:d20a:c600:a502:2dae:7716:601b", "2a02:c7f:d20a:c600:a502:2dae:7716:601c" and "2a02:c7f:d20a:c600:a502:2dae:7716:601d" for IPv6. Theses defaults can be changed and different addreses used by modifying the 'TEST_NETWORK_INTERFACE_IPV4' constant at the top of the ipv4_tests.rs file to change the IPv4 addresses and modifying the 'TEST_NETWORK_INTERFACE_IPV6' constant at the top of the ipv6_tests.rs file.
Once configured the tests can be run using the command below. Note that some of these tests require the network to support IP multicast. This command will only run the tests relevant to the current OS (IPv6 multicast receive tests skipped on windows).

cargo test -- --ignored --test-threads=1

Testing the library - Code Coverage

To view the code coverage of the library the grcov tool from Mozilla is used. Once setup (as detailed within the grcov documentation) the following commands can be run to install the nightly toolchain (required for grcov), build the library and run the tests (run from the parent top-level project directory). This will generate a coverage folder in target/debug/ which contains a webpage (index.html) with the details of the code coverage.

export CARGO_INCREMENTAL=0

export RUSTFLAGS="-Zprofile -Ccodegen-units=1 -Copt-level=0 -Clink-dead-code -Coverflow-checks=off -Zno-landing-pads"
rustup run nightly cargo build
rustup run nightly cargo test -- --ignored --test-threads=1
rustup run nightly cargo test
grcov ./target/debug/ -s . -t html --llvm --branch --ignore-not-existing -o ./target/debug/coverage/

Testing the library - Multi-Machine Integration Tests

The setup of these tests is detailed in the report. These tests require a specific machine configuration with a shared file system as detailed within the report. Once set up there are two sets of tests. The single-rcv-src folder allows running tests that work between just 2 machines, a distinct sender and receiver. To run these modify the 'test_run'.sh so that 'REMOTE_PC' and 'REMOTE_PC_2' contain the addresses of the 2 machines to use for the test. The tests can then be run using:

sh ./test.sh

To run the tests which show the protocol working between multiple machines enter the 'multiple-rcv-src' folder, the 'REMOTE_PC' array within the 'test_run_multiple.sh' file can then be modified to include the test machines to use. The tests are then run using sh ./test.sh. Note that as discussed in the report these tests do not automatically check if the test succeeded and require manual intervention to check the produced output is correct as there are multiple possible correct outputs. The tests can be run using:

sh ./test.sh

Testing the library - Interoperability Testing

The library was tested for interoperability with three external programs, Avolites Titan v11.4, sACNView version 2.1.0 and Vectorworks Vision Plus 2019 version 24.0.6.521266. This testing is detailed within the report and in the 'CS4099 - Interoperability Testing.pdf' file. The Interoperability Testing folder also includes videos showing the various tests and the results with details allowing recreation if desired.

Testing the library - Fuzz Testing - Linux Only

The library parsing function is tested using fuzz testing. This utilises a rust version of American Fuzzy Lop with run instructions described here https://rust-fuzz.github.io/book/afl/setup.html. This AFL only supports linux. For this project afl can be installed by running(from the Fuzzing/sacn-parse-fuzz-target folder):

cargo install afl

The fuzzer can then be run using the following commands (from the Fuzzing/ folder), note that sometimes depending on the machine more configuration is needed by afl and this will be prompted for.

cargo afl build
cargo afl fuzz -i fuzz_in/ -o out target/debug/sacn-fuzz-target

The fuzzer will then run forever until stopped.