train_rec
FQT
Team 4A

Installation

Requirements:
- A computer running Debian 7.8.
- All Qt packages available for your system.
- Access to a mysql server with table structure agreed upon in class. Can be generated by the sql dump files on this team's github project. https://github.com/ctag/cpe453/tree/master/sql_loader
- Basic understanding of code compilation, Qt, and general ability to research questions and troubleshoot.
- Mysql-workbench

SQL:
The above sql files are a standard for mySQL and their implementation to populate a server is separate from our project.

Procedure:
- Collect the program, train_rec, from hosting at github. https://github.com/ctag/cpe453/tree/master/train_rec
- Open project file with qt-creator.
- Compile and run.

Testing

With train_rec running, and the usb buffer connected to the computer:

Action: Click the refresh button on 'Packets'.
Response: Drop down list of usb ports is updated.

Action: Select appropriate usb device from drop down list on 'Packets'.
Response: The device is selected. Congratulations.

Action: Click 'connect' on 'Packets'.
Response: The program will attempt to connect to the usb device and report status on 'Console'.

Action: Fill fields on 'SQL'.
Response: Self evident fields are filled.

Action: Click 'connect' on 'SQL'.
Response: The program will attempt to connect to the declared mySQL server. Status is logged to 'SQL' tab.

Action: For each of the listed macros, fill the req_macro table on the mySQL server. The word IN_CAPS at the top of each field is the macro, which goes in the 'macro' field of the table, arguments are specified as numbers corresponding to utility.
Response: The program will delete the row and send a packet as modeled to loconet.

Example Listing:

```
INSTRUCTION, arg1, arg2
Packet Format in [byte]s.

    Description of macro, how to wield it, why it matters. If no arguments are given, then leave their fields as 'null' in sql.
```

Here INSTRUCTION would be a string entered in the “macro” field of the table, with arg1 and arg2 being placed in their respective columns as well.

```
SLOT_SCAN, slot#
[BB] [SLOT] [00] [CHK]

    Queries track for the status of the given slot#. Valid slots are 1-119. Returns with the state of a 'train' in sql, which is really just a slot. Don't query the status of slots that you aren't interested in; for example, what if MASTER returns several slots with the same locomotive address? Which one do you update to control the real train?! If you only query slots that you know have trains, this won't happen.
```

```
SLOT_DISPATCH, slot#
[BA] [SLOT] [00] [CHK]

    Supposedly puts the given slot back into “COMMON” mode, but has
```

not worked consistently for me. Use when slot has mode “IN_USE”. If the associated train does not update, then the command failed with a LACK packet. To be safe, follow with a SLOT_SCAN.

SLOT_CLEAR, slot#
[B5] [SLOT] [03] [CHK]

Puts the given slot into “FREE” mode. Use when slot has mode “COMMON” or “IN_USE”. Associated train state does not automatically update, must run SLOT_SCAN after this command to check the train state.

SLOT_REQ, train#
[BF] [00] [ADR] [CHK]

Requests a slot for a given train address. Returns with the state of a 'train' in sql, which is really just a slot. This is the initial command to go from train address to slot number that's used in the other macros. After running this macro, check the slot/train table for a new row corresponding to the train address.

TRACK_ON
[83] [CHK]

Turns power to the track on. Usually results in all detection sections sending a bootup packet which lists them as “occupied”. train_rec can be configured to ignore extraneous detection sections and not let them spam the status table when the track powers on.

TRACK_OFF
[82] [CHK]

Turns power to the track off.

TRACK_RESET
[82] [CHK] then [83] [CHK]

Turns power off, waits a few (ten) seconds, turns power back on.

TRAIN_REQ, train#, speed%
[SEVERAL PACKETS]

Though no team has requested this feature, it has been added at the behest of the product owner. Has not been tested; don't use it. Train# is the address of a train or trains; speed% is an integer -100 to 100. The system will spam traffic to get a slot for the train, set it to active, and give it a throttle command; because of this, you may have to call the macro twice if the train is not already in an IN_USE slot.

Suggested workflow: Take a train address, call SLOT_REQ to get a slot, then spam this macro twice. After that, you should only need to call this once to update speed, though it will still inundate the track with traffic and make life miserable for everyone else.

Action: With track powered on (see above macro), run a train across two detection sections which are listed in the 'track_ds' table of the mySQL server.
Response: The table will update to show a '1' in the status field of the occupied detection section.

Action: Click 'Disconnect' on 'Packets'.
Response: The program will close the usb connection.

Action: Click 'Disconnect' on 'SQL'.
Response: The program will close the SQL connection.

That's it, all requirements from a loconet bridge. Switch status is not returned because it is not available. Extra features in the train_rec suite, including train_rec_sql, are provided as-is.