# [DINST]

~~Dumb~~ Digitrax Instruction Set

Christopher Bero
Team 4A
CPE453 - Spring 2015

This document outlines train_rec, a program capable of handling communication to a digitrax HO model track via a network.

# train_rec

aka Train What?

Train Receiver (interchangeably: train_rec) is a multi-threaded message handler written in C++ with the Qt framework. At its core is a set of classes which abstract between C++ level variables and the raw hex which comprises Digitrax's (rather disgusting) serial language. These classes are fairly portable, and can be salvaged for future projects which wish to take the track communication in a different direction. At the top level train_rec has four threads (existing as classes) which operate the GUI, USB connection, SQL connection, and UDP socket. This division greatly benefits the program's performance but does also add some slight programming overhead. Tangential classes serve a very small purpose and are used as a stop-gap to keep the system's architecture clean; they may be removed by future updates to the software. Because of their low-impact on the overall project, these last classes are not documented here.

Low level classes:
- locobyte.cpp
- locopacket.cpp

Top level classes:
- locoserial.cpp
- locosql.cpp
- locoudp.cpp

Tangential classes:
- locoblock.cpp
- locotrain.cpp

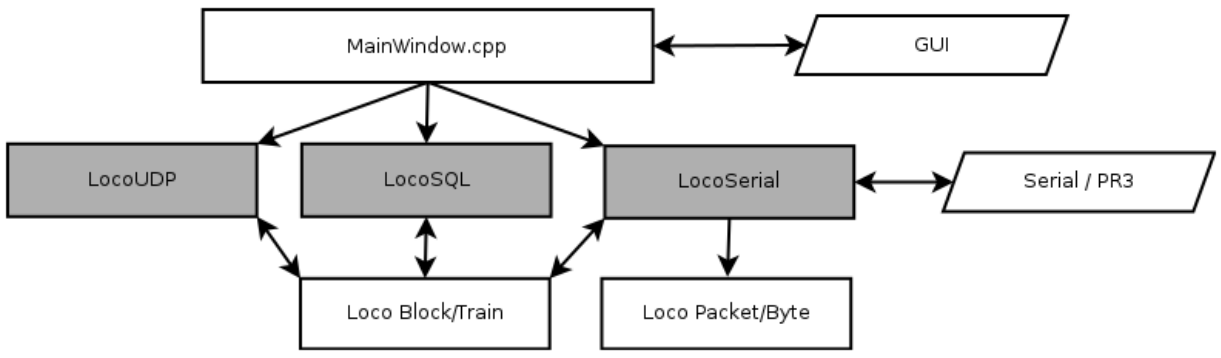


*Illustration 1: High Level Design*

# REQUESTS

aka OMG WHAT DO?

Requests are methods of interacting with Loconet via SQL tables. Each type of request is designed to make your life as a team in CPE453 easier. To that effect, if you wish for additional features, don't hesitate to ask (of course, whether the feature is feasible will have to be determined).

Keep in mind that slots are a software-only construct. Each train has a (non-unique) ID, which can only be controlled via a "slot number" that is associated with the train. The SQL in train_rec wields slot numbers with reckless abandon and occasionally calls them trains to keep the notion easier (but can be confusing!).

**Train/Slot Requests:**

Set the direction and speed of a train/slot. Slot number is the ID. Speed is a scalar percent. Direction is a boolean value, with true corresponding with reverse movement.

**Switch Requests:**

Set the position of a switch. The number physically labeled on the switch is used as the ID.

**Macro Requests:**

Most everything else is a macro. Macros are utilities to make interacting with Loconet easier on a higher level. Check the demo program, train_rec_sql, for macro implementations.

# MACROS

aka Trains for Normal People

Macros listed here are to provide an overview, as sql implementation may vary. Sample implementation software is available either on the EB Linux Lab computers, or from https://github.com/ctag/cpe453/tree/master/train_rec_sql.

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.

# Detection Section Addressing

aka Wizardry

**TL;DR:**
Likely the most elusive part of this system; luckily what we've gleaned will keep you from having to work with this conversion. You can totally just skip this page.

**Introduction:**
There are three numbers associated with a detection section. *Raw hex* is created by parsing address bits. *LS numbers* are an abstract that appears to be JMRI specific? Lastly are *BDL-DS numbers*, where each BDL168 has a number, and each DS on the BDL has a spot 1-16. We get raw hex from Loconet, and want to reach BDL-DS numbers to use in SQL.

**An example:**
Consider the packet [B2 6C 58 79].

Using the Digitrax documentation, we find the associated hex address for the detection section to be [46C] with aux bit of [0]. Awesome, after this point the documentation is utterly useless!

Next, we convert the hex directly into a decimal number: 1132.

Now the aux bit comes into play; if the aux bit is set we add 1 to the decimal number, then multiply by 2. Otherwise (if aux is 0) we multiply by 2, then add 1. Thus we get (1132*2)+1 = 2265.

This resulting number, 2265, is the LS number; written LS2265. Now we want to get to the BDL-DS number. To get the board, do LS# modulo 16. To get the DS, do LS# divided by 16 and round up. In our example 2265%16 = 11 and ceil(2265/16) = 142. The last digit of the BDL number corresponds to the board below the track. Thus we achieve 2-11 as the detection section of interest! This algorithm has been tested a does work for all detection sections on the inner-urban track.

# Implementation

aka The Finished Puzzle

This is the easy part! (At least I really, sincerely hope).

You may download train_rec and supporting software from github.com/ctag/cpe453.

**OR**

As any EB Linux Lab user, run the following commands to use a pre-compiled executable of train_rec or train_rec_sql.

```
/home/student/csb0019/train_rec
```

train_rec is the main program here, it does all the magic for the track, and must be connected to one of the USB-buffers. You can run it from one of the Linux computers at the front of the train lab.

```
/home/student/csb0019/train_rec_sql
```

train_rec_sql is a sample program which allows you to manually control trains, switches, and view blocks(detection sections). Use it to get the SQL queries which work with train_rec.

As an easter-egg (I guess?) train_rec also supports direct UDP packet commands, should you so wish to implement it. Default port is 7755, send a datagram with just the QString hex for a packet minus the checksum. Packet will be checksummed and fired off to serial without much parsing.