[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.

Index

aka Again, and this time in English!

To aid in getting up to speed with this Dumb Instruction Set, a collection of redundant and otherwise useless terms are specified on this page. These pertain to both train_rec's code base and Digitrax's own documentation.

Loconet

Loconet is the HO model track and supporting digitrax hardware. I prefix most of the classes in train_rec with loco- but have eschewed using loconet as a class name to keep from conflicting with this usage of the word.

Serial, USB, Loconet Buffer, PR3

These terms correspond to either the physical PR3 device which allows a computer to link to the track or to the serial connection to the PR3 via train_rec's locoSerial class.

Slot

Can be either a Qt slot, or a virtual slot used by Loconet to track trains. Virtual loconet slots are a construct which manage trains and special functions. Trains do not have to have unique addresses, but the pairing of a slot to a train is unique, and once paired trains are controlled by using the number of the slot they are in.

opcode

Digitrax fancies themselves real computer scientists and call the first byte in each packet an opcode. This byte determines the purpose of the packet. A database of opcodes is available via the locoPacket class.

Checksum

Every loconet packet is terminated with a checksum byte which can be used as parity to verify the packet was correctly received. This process is taken care of in the locoPacket class.

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 a higher level, train_rec has four threads (existing as classes) which operate the GUI (mainwindow), USB connection (locoserial), SQL connection (locosql), and UDP socket (locoudp). 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. To aid in future development, low and high level classes come equipped with a small suite of debugging statements which can be individually toggled on and off for each class.

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

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

Tangential classes:
- locoblock.cpp
- locotrain.cpp

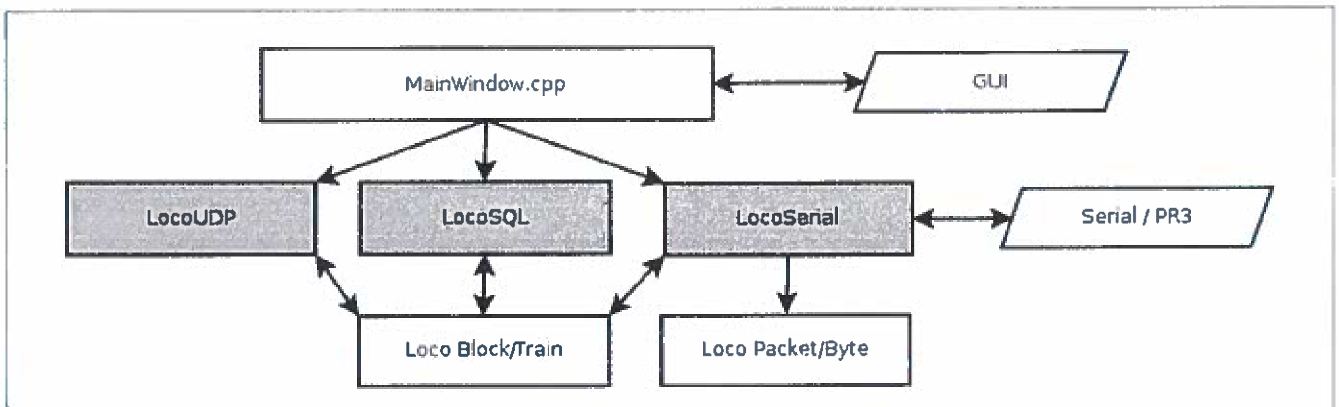


Illustration 1: High Level Design

Class - LocoByte

aka Descent into sanity

LocoByte does a lot of heavy lifting for you, it acts as a container for individual bytes inside of a loconet packet. As such, the class can self-determine if its instance is a packet's opcode. Most of the structuring for the class is built around portability and input/output; You can enter the bytes as hex, decimal, binary, or directly as qBitArrays/qByteArrays.

Methods

== - You can compare locobytes with the custom == operator.
= - You can assign a qBitArray to the locobyte with =.
bitsFromHex() - A Qstring with hex, such as "BA" can be loaded into the locobyte.
set_fromHex() - Public wrapper around bitsFromHex().
get_binary() - Returns a Qstring of binary.
get_hex() - Returns a Qstring of the hex.
get_oneBit() - Used to poll individual bits of the byte.
set_oneBit() - used to modify individual bits of the byte.
get_isOP() - Uses Digitrax's specification to determine if a byte is an opcode.
get_packetLength() - Uses Digitrax's specification to determine the supposed length of the rest of the packet in bytes. Only to be used on opcode or byte directly after opcode.
get_hasFollowMsg() - Uses Digitrax's specification to determine if the packet should receive a reply from the track.
do_debugBits() - Don't use this, it just spills the class contents to std_out.
do_genComplement() - Used in generating the checksum, will set the internal byte to the complement of whatever it used to be. **Doesn't return a value.**
set_fromBinary() - Sets the class contents from a Qstring of the byte's binary representation.
set_fromByteArray() - Method adapted from solution found online (referenced in the class's code). Used to set the byte's qBitArray from a qByteArray.
get_qByteArray() - Returns the calculated qByteArray.
get_qBitArray() - Returns the core qBitArray, faster than get_qByteArray().
get_decimal() - Returns an integer of the byte's decimal value.
SetDebug() - Unimplmented.

Class - LocoPacket

aka Descent into [BD0244F7]

LocoPacket is… another c++ class? Yeah, I mean its pretty self evident. Use this class to build loconet packets from locobyte instances. LocoPacket is integrated with a SQLite database that pairs opcodes with descriptions, this list determines which opcodes are legal to be used with the track.

Methods

== - You can check equality with this overloaded operator.
clear() - Purge the packet contents.
do_openDB() - Attempts to open the SQLite database of opcodes.
do_closeDB() - Closes the connection to the SQLite database.
set_allFromHex() - Sets the packet contents from a Qstring.
get_packet() - Returns the Qstring of the packet's hex.
get_finalSize() - Returns the Digitrax specification of how long the packet should be based on the opcode. Does not reflect the current number of bytes in the packet.
get_size() - Returns the current number of bytes in the packet.
get_Opcode() - Returns Qstring of the opcode byte.
get_locobyte() - Transparently escalates a locobyte out of the packet.
do_xor() - Used in generating the packet's checksum, returns a qstring of two locobytes xor.
do_genChecksum() - Generates and appends a checksum for the packet.
do_appendByte() - Adds a byte from a qstring to the end of the packet.
do_appendByteArray() - Adds a qByteArray to the end of the packet.
do_appendLocoByte() - Adds a locobyte to the end of the packet.
validChk() - Returns a boolean status for the packet's checksum.
hasOP() - Returns a boolean status for the packet's opcode.
get_isEmpty() - Transitional method to determine if a packet is empty.
validOP() - Checks the packet's opcode against the SQLite database.
get_DBopcodes() - Returns all of the opcodes in the SQLite database.
get_DBnames() - Returns all of the corresponding opcode names in the SQLite database.
is_followOnMsg() - Boolean value of whether the packet should get a response from the track.
get_QBitArray() - Returns the packet's QbitArray.
get_QbyteArray() - Returns the packet's QbyteArray.

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.

TODO

aka Pull requests welcome.

- Debugging could use an overhaul. I feel the structure is fine, but the statements could be much clearer, and there should by run-time buttons for enabling debugging statements.
Estimated difficulty: super easy.

- Let's face it, the GUI sucks. A full overhaul may not be necessary, but at least connect the signals/slots from each thread to relevant output on the screen.
Estimated difficulty: how well do you really know Qt? Good luck.

- Tangential classes should be removed and replaced with a smarter signal/slot design between the threads.
Estimated difficulty: a good day's work.

- SQL structure is currently somewhat hard-coded to ECE's pavelow server, this should be fixed and the table structure should be better refined and defined.
Estimated difficulty: 40 hours.

- LocoSQL currently handles the conversion between speed and packet hex, this should be pushed back to LocoSerial.
Estimated difficulty: You're going to learn a lot about this code structure… 40 hours.

- UDP is really non-existant. A bolstered UDP class capable of handling macros could make the program much more robust for future projects.
Estimated difficulty: 40 hours.

Team 4A

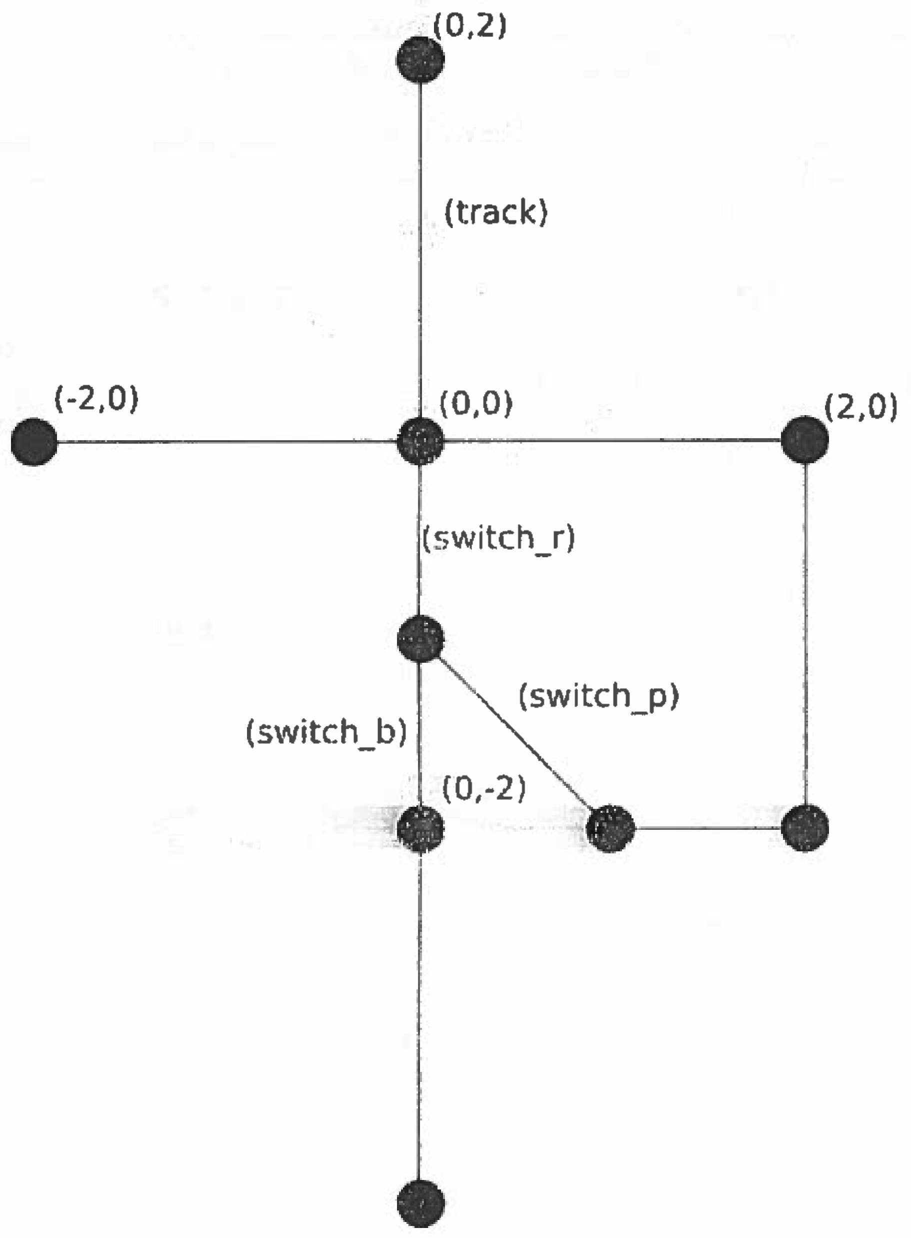
SQL Track Representation

In the interest of providing a table of the track which is both entirely described and easy to implement, teams 4 A/B have decided to design a zero-mass graph in the form of a single-linked list. The current testing table is described below.

Name:	**id**	**from**	**x**	**y**	**track_type**	**ds**
Type:	Integer, unique key.	Previous section's id. Integer.	First quadrant x value, inches. Float.	First quadrant y value, inches. Float.	Enum: track, switch_r, switch_p, switch_b	ID of detection section, reference from separate table. Integer.

This format allows for a reasonably small table which can describe the entire track by assuming an uncomplex graph type where all edges have associated vertices. Each piece of track or branch of a switch defines a destination x,y tuple and has an origin associated with the coordinates of the 'from' section behind it to create one edge bounded by two vertices.

The table currently holds a sample of imaginary track from the following template. Table contents have not been updated yet to reflect the 1'st quadrant coordinate decision, in which all values are to be associated with space in the first quadrant of a Cartesian plane to avoid handling negative values in computation.


(0,2)
(track)
(-2,0)
(0,0)
(2,0)
(switch_r)
(switch_p)
(switch_b)
(0,-2)

Train Receiver Usage

Follow along on any ECE Linux computer.

- Plug in the track's USB cable.
- From a terminal, run `/home/student/csb0019/train_rec`

- On the “Packets” tab under “Serial Connection” click 'refresh'
- Choose the USB device from the drop down list. It is usually labelled /dev/ttyACM* or /dev/ttyUSB*
- Click 'Connect'

- On the “SQL” tab, enter your desired server's credentials and click 'Connect'

- That should be it, for normal operation the program will now route SQL commands to the track.

- Before closing the program, click 'Disconnect' for both the serial device and SQL connection. The program can then be halted normally.

LocoNet□ Personal Use Edition 1.0 SPECIFICATION:

Digitrax Inc., Panama City, FL 32404
October 16, 1997

Introduction:

This is the definition of the protocol used by Digitrax products that communicate on the **Long distance** version the LocoNet□ network. **This LocoNet Personal Use Edition 1.0 information is provided solely for non-commercial private use by Digitrax customers.** No rights are conveyed for the commercial use of this information, and Digitrax Inc. is not able to provide technical support for private use. Digitrax conveys no warranty for this information and incurs no obligations for its use or incorrect usage. Possession of this information signifies acceptance of these conditions of usage. LocoNet is a registered trademark of Digitrax Inc.

Summary and design philosophy:

LocoNet□ is a "PEER to PEER" distributed network system on which all devices can monitor the network data flow. The network is event driven by different devices in time, and is not polled by a centralized controller in normal operation. LocoNet is a powerful power decentralized and "scalable" distributed system. This is similar to the way the worldwide telephone system works- i.e. there is no worldwide "master control program" and all telephone central offices follow a strict set of "rules" for worldwide access. The Internet operates in a similar "distributed" manner.

The access technology is Carrier Sense Multiple access with Carrier Detect, CSMA/CD. This type of network technology is the overwhelming choice for connecting computer LAN's around the world. The first implemented 10M bps version of Ethernet has been upgraded to 100M bps and now has even 1,000M bps extensions. The CSMA/CD is the basic physical or media access layer that allows multiple devices to interoperate and exchange data efficiently.

LocoNet□ uses CSMA/CD techniques to arbitrate and control network access. On top of this physical layer, LocoNet□ specifies a higher level of message protocol that gives efficient management of data structures for operations in the model railroad environment. A sensible bit rate was chosen as a good compromise between ease of wiring, with no need for terminations or complex distribution rules, and fast latency or rapid response time. We avoided a simple "specsmanship" of specifying faster bit rates to distinguish us from other system manufacturers' capabilities, and instead chose a fundamentally new system architecture we call LocoNet□.

Since LocoNet□ is a distributed system where each device can determine the urgency of its access, it is not easy to compare in a one-to-one manner with older technology such as polled bus systems. We can compare the explosive growth of commercial computer LAN's, and even the Internet itself, to understand that all new "state-of-the-art" data processing systems are based on network type topology. The old idea of a central "mainframe" and "Main Control Program" was state of the art for the 60's and 70's and has yielded to new technologies and ideas.

Comparing "raw" bit rates is not meaningful in this context since the strength of a true network system is in allowing multiple queued access without requiring polling or data flow controlling overheads. The real strengths of a LAN become apparent when a large number of devices need fast data access, or low latency, to transfer requests or state change information. With a LocoNet□ implementation, a modest bit rate is sufficient on a large railroad layout to give realistic operation. Note that a typical LocoNet□ implementation allows us to achieve about 98% of network traffic capacity with less than 1% collisions.

Another decisive advantage of an optimally designed LAN is the ability to overlay or add new capabilities and thus "scale" the system in features as well as size. Not having to modify a "master control program" each time we wish to reconfigure or expand system features is a very real advantage.

An example of this is the way the Digitrax "Big Boy" can be expanded with DT100's and get a network "fast clock", even though the DT200 master did not itself support this. The soon to be released Digitrax Tetherless throttles (both Radio and IR) can take advantage of the network environment to add a number of different types of features among many interfaces on the same LocoNet□ network. LocoNet□ is configured to allow all types of data traffic to flow on a single wiring scheme. This means you do not have to run separate Throttle, Booster, Feedback and PC wiring for the system. Since LocoNet □ is based on LAN technology, a large LocoNet□ layout can use Bridging , Routing and Fault isolation techniques as used on commercial networks to expand to a large physical extent.

To allow the addition of multiple independently operating PC's around a large physical layout using simple access hardware, the LocoNet□ was "slowed down" and simplified somewhat over a "raw hardware" driven system capability. This trade off was made to ensure ease of attachment of PC's, and we at Digitrax feel that once the power of "computer assisted modeling" was appreciated, a single PC would rapidly prove inadequate, as task complexity and demands increased. Also, we feel it is important that PC tasks can be independent, modular and spread out among many different PC's around the layout. In particular being able to use slower old "AT" 286's and 386's running DOS and Windows 3.1 etc., was important, since requiring a dedicated new "Wintel PC" costing over US$2,000 seems like a very expensive proposition for what is essentially a hobby!

<u>Technical Specification:</u>

The normal LocoNet□ state is IDLE, with no data traffic unless a device has information to send. With no traffic flow, the network is RFI quiet.

<u>Physical:</u>

The full implementation of LocoNet□ uses a 6 pin USOC RJ12 style TELCO connector. The network is designed to operate "daisy chained" on unterminated 26 AWG 3 pair cable or flat 6 conductor type 120 ohm impedance ribbon cable. It can be cabled in numerous variations. It is designed to be tolerant of cabling environment. It is permissible for the individual wires to loop back on themselves, noting that the 2 Rail_sync lines are of opposite phase and cannot be connected. The connections are balanced to minimize RFI. The connections may be branched in any combination to yield a "Star" or "Bus" or any combination thereof. Only a single LocoNet□ current termination is needed and is typically supplied by the system "Master".

Pinouts for the RJ11/6 connector are:

1-RAIL_SYNC-	white
2-SIGNAL GROUND	
3-LOCONET-	
4-LOCONET+	
5-SIGNAL GROUND	
6-RAIL_SYNC+	blue

Using typical UL6010 6 conductor 26AWG Telephone flat ribbon cable, a network may typically have a total parallel cable length of up to 2,000', with no point-to-point length exceeding 1000'. This is using the "standard" Long Distance LocoNet□ termination of a 15 milliamp positive *current source* on pins 3 and 4 of the connectors.[The capacitance of 2,000' of this cable is approximately 68,000pF, and the loop resistance of a pair of these stranded 26AWG conductors is 80ohms per 1000'].

The maximum amount of parallel cable is limited by the requirement that the minimum RISE TIME in the region of +2 Volts to +7 Volts, is 0.35 Volts per microsecond. The minimum FALL TIME in the same region should be greater than 0.75 Volts per microsecond

Electrical:

LocoNet is a "Wired-Or" multiple access linear network using CSMA/CD techniques. Repeaters, network buffers and isolators can be implemented. For the current SINGLE ENDED implementation, the 2 LocoNet signals ,+ and -, are paralleled and the RJ11 cable connections become polarity insensitive, wire resistance is lowered and connection reliability is enhanced.

SINGLE ENDED voltage levels and characteristics are :

a) High = 1 = "MARK" : LOCONET+/- voltage above +4.0 Volts with respect to ground conductors.

b) Low =0 = "SPACE" : LOCONET+/- voltage below +4.0 Volts with respect to grounds.

c) The data should be received with 1.0 volt of HYSTERESIS centered on +4.0 volts.

d) Maximum LOCONET+/- high voltage is +24V and nominal is +12V

e) Minimum receiver input impedance is 47 Kilohms, measured from pins 3&4 to pins 2&5(gnd)

f) The Transmitter is OPEN COLLECTOR to SIGNAL GROUND and should be able to sink 50 milliamps in the "ON" state at no more than 1.6V, and withstand 35 Volts in the OFF state.

g) One single device shall provide the "Wired-Or" pull-up for the LOCONET+/- signals. Typical termination is performed by the packet generating "MASTER" and is a 15milliamp current source from +12V.

h) Loconet devices may draw up to 15 mA from the RAIL_SYNC+ /- lines whenever the voltage is greater than 7V. The unloaded voltage is between 12V and 26V max. It is general practice to provide a LOCAL current limited copy of the closest track voltages, to pins 1&6 of Throttl e jacks around the layout. In this case the master "backbone" copy of RAILSYNC +/- is not on the Throttle jack.

i) The RAIL_SYNC+/- are a low power copy of the DCC data to be transmitted to the rails. The signals may be received by a differential receiver and boosted to drive the rails.

j) A device with a separate power supply isolated from LOCONET, may connect to the LOCONET+/-pins 3&4 and SIGNAL GROUND pins 2&5 with a just 2 wires.

k)To use a 1/4" Stereo 3 pin Plug , the SIGNAL GROUND shou ld be connected to the Sleeve, the LOCONET +/- connected to the Tip, and the Sleeve may be connected as a power source. The power supplied to the Sleeve MUST be a CURRENT SOURCE (from +12V to +26V) and be limited to 20 milliamps maximum , because the Plug shorts the Tip and Ring when initially inserted.

NETWORK Timing:

LocoNet data is sent in normal ASYNCHRONOUS format using 1 START bit, 8 DATA bits and 1 STOP bit. The 8 bit data is transmitted LSB first. The bit times are 60.0 uSecs or 16.66 KBaud +/- 1.5%. A PC serial "COM" device can use the convenient rate of 16.457 KBaud. This corresponds to a Divisor of 07 for the standard NS8250 UART chip or equivalent used by most compatibles. Bytes may be transmitted "back-to-back", with a Start bit immediately following the Stop bit of the previous character.

Normal network "IDLE" is the "MARK" voltage state. Data is sent HALF DUPLEX and transmitters process the TRANSMIT ECHO to monitor network collisions.

CARRIER DETECT (CD) for fundamental network access timing may utilize simple RC time constant "one-shots". CD becomes active immediately on any detection of network in the SPACE state. It then times out for 20 bit times or 1.2 milliseconds as the **CD BACKOFF** time and goes inactive. CD jitter of up to 180uS is acceptable and helps ensure even statistical network access with minimal COLLISIONS.

All transmitters are responsible for detecting TRANSMIT COLLISIONS on a 1 bit or whole echo-byte basis. If a TRANSMIT collision is detected the TRANSMITTER will force a line BREAK of 15 BIT times with a Low or "SPACE" on LocoNet, and decrement the Transmit Attempt count. (The device can attempt the next acess at the same Priority, or change it by some small amount, depending on an internal Phase reference, if the delay from Network free to Siezure is greater than 2uS).

All receivers will process the BREAK as bad data framing and reset Message parsers The network is then free to re-arbitrate access. Any message that has format or framing errors , data errors or is a fragment caused by noise glitches and does not completely follow the MESSAGE FORMAT will be ignored by ALL receivers, and a new OPCODE will be scanned for re-synchronization.

NETWORK Access:

To SEIZE access to the LocoNet a device shall wait for the **CD BACKOFF** time to elapse from the last space level seen on LOCONET+/-. The "MASTER" device may at this time seize the network immediately upon seeing CD has "released". All other devices add **additional time delays** before being allowed to attempt NETWORK SEIZE. Throttles and other devices will ***always*** wait a minimum of another 6 bit times or 360uS **MASTER** delay before being allowed to attempt a network seize or access.

On the first attempt to access the network to transmit new input information, a device will add a further **PRIORITY** delay of up to 20 bit times. If network access is not gained after the priority delay, due to seizure/usage by another device, the PRIORITY delay is decremented by 1 bit time for the next access attempt, which may occur after the current message or fragment ends. In this way all devices may be queued in priority, and **none may seize the network in priority over the MASTER**, which often returns acknowledgments and other information based on a previous request message.

A device shall make at least 25 Transmit Attempts before deciding Message Transmit failure.The Transmit Attempts must include attempting Network access for at least 15 milliseconds per access attempt.

A BUSY opcode is included to allow the master to keep the network active whilst it is performing a task that requires a response, and entails a significant processing delay, i.e. it can ensure no new requests are started until it has responded to the last message. In addition to the BUSY opcode, the master may simply add 15 bit BREAK sequences to the network to delay any new messages starting until it has completed and responded.

Individual device types may have their access tailored by setting different maximum and minimum PRIORITY delays. In particular, SENSOR type devices may have initial Priority of 6 or less, so they can broadcast messages to the network in a timely manner.

To provide the greatest protection against network bandwidth being wasted due to repeated collisions a device should *assert the SPACE of the start bit of the message OPCODE within 2 microseconds of determining that its access delays have elapsed and the network is still free.* This has the effect of improving the COLLISION aperture uncertainty for a transmit collision. If the transmitting device detects a transmit collision either by bad TRANSMIT ECHO or a TRANSMITTED 1 bit being forced to 0 on LOCONET, it will initiate the 15 bit BREAK sequence to flag all devices that data is bad.

PC Access:
A simple "COM" port on a PC may access the *network* by a more direct method. The protocol has been encoded so that a PC may watch the LocoNet message dialog and infer that the *network* is free because the last message decoded does not imply a **follow-on response**, so that the *network* is immediately free for a new message dialog. In this situation, the PC may immediately seize the *network* before the CD BACKOFF time has elapsed. This allows the PC to pre-empt all other devices and completely control the LocoNet to the level desired. Note that the message <81><7e> is a "time burner" NOP code sent by a Master to restart the CD Backoff timers, and hence keep the *network* busy in a hardware sense. This <81> opcode should thus be simply stripped and ignored.

Several PC's may share access to LocoNet by subdividing the 20 bit CD BACKOFF delay into priority windows for access. They are responsible for detecting transmit COLLISIONS by checking their TRANSMIT ECHO data and watching a CARRIER DETECT to see if a PC transmit "window" is active already, before attempting to transmit.

If the LOCONET+/- signal remains at a fixed SPACE (low) level for more than 100 milliseconds, a DEVICE will assume a DISCONNECT state is in effect. From this DISCONNECT state or initial start-up state a device will wait a 250millisecond **STARTUP** backoff before attempting to access the network. A device will not need to reset its internal state upon DISCONNECT and re-connection ,but if it is maintaining a SLOT in the refresh stack it will be required to check the SLOT status matches its internal state *before re-using any SLOT.* If a device diconnects from LocoNet and so does not access or reference a slot within the system PURGE time, the master will force the unaccessed SLOT to "COMMON" status so other system devices can use the SLOT.
The typical purge time of a DT200 operating as a Master is about 200 seconds. A good "ping" or Slot update activity is about every 100 seconds, i.e. if a user makes no change to a throttle/slot within 100 seconds, the throttle/device should automatically send another speed update at the current speed to reset the Purge timeout for that Slot.

MESSAGE Format:
All LocoNet communications are via multi-byte messages. The "MASTER" is defined as the device that is maintaining the refresh stack for DCC packet generation and is actively generating the DCC track data. Refresh of information is typically only performed for MOBILE decoders. Stationary type decoders are not refreshed and individual IMMEDIATE commands are sent out to the track as requested.

The MASTER is only privileged in respect to performing the task of maintaining the locomotive REFRESH stack and generating DCC packets. In this way other network transactions may occur that the MASTER does not need to be involved with or understand , as long as they follow the MESSAGE PROTOCOL and timing requirements. i.e. Other devices may have a dialog on the network without disturbing or involving the "MASTER".

Devices on LocoNet monitor the MESSAGES, check for format and data integrity and parse good messages to decode if action is required in the context. Devices such as Throttles, Input Sensors , Computer interfaces and Control panels may generate LocoNet messages without needing prompting or polling by a central controller.

Devices frequently will be added and removed from an operating LocoNet . The devices and protocol are tolerant of electrical and data transients. The format chosen gives a good degree of data integrity, guaranteed quick network-state synchronization, high data throughput , good distribution of access to many competing devices and low event latency. Also , the devices may be operated without need for unique ID or other requirements that can make network administration awkward.

The data bytes on LocoNet are defined as 8 bit data with the most significant bit (transmitted last in the 8 bit octet) as an OPCODE flag bit. If the MS bit , D7, is 1 the 7 least significant bits are interpreted as a network OPCODE . The opcode byte may only occur once in a valid message and is the FIRST byte of a message. All the remaining bytes in the message must have a most significant bit of 0 , including the last CHECKSUM byte. The CHECKSUM is the 1's COMPLEMENT of the byte wise Exclusive Or of all the bytes in the message. except the CHECKSUM itself. To validate data accuracy, all the bytes in a correctly formatted message are Exclusive Or'ed. If this resulting byte value is "FF" hexadecimal, the message data is accepted as good.

The OPCODES may be examined to determine message length and if subsequent response message is required. Data bits D6 and D5 encode the message length. **D3=1 implies Follow-on message/reply:**

	D7	D6	D5	D4 --	D3	D2	D1	D0	
(Opcode Flag)									
	1	0	0	F	D	C	B	A	Message is 2 bytes, including Checksum
	1	0	1	F	D	C	B	A	Message is 4 bytes, inc. checksum
	1	1	0	F	D	C	B	A	Message is 6 bytes, inc checksum
	1	1	1	F	D	C	B	A	Message in N bytes, where next byte in message is a 7 bit BYTE COUNT.

The A,B,C,D,F are bits available to encode 32 OPCODES per message length.

<u>REFRESH SLOTS:</u>

The model of the MASTER refresh stack is an array of up to 120 read/write refresh SLOTS. The slot address is a principal component and is generally the second byte or 1st argument of a message to the master. The refresh SLOT contains up to 10 data bytes relating to a Locomotive and also controls a task in the Track DCC refresh stack. Most mobile decoder or Locomotive operations process the SLOT associated with the Locomotive to be controlled. The SLOT number is a similar shorthand ID# to a "file handle" used to mark and process files in a DOS PC environment. Slot addresses 120-127 ARE reserved for System and Master control.

Slot #124 ($7C) is allocated for read/write access to the DCS100 programming track, and the format of the 10 data bytes is not the same as a "normal" slot. See later.

<u>Standard Address Selection:</u>

To request a MOBILE or LOCOMOTIVE decoder task in the refresh stack, a Throttle device requests a LOCOMOTIVE address for use,(opcode <BF>,<loco adr hi>,<loco adr lo>, <chk>). The Master (or PC in a Limited Master environment) responds with a SLOT DATA READ for the SLOT ,(opcode <E7>,,) ,that **contains** this Locomotive address and all of its state information. If the address is currently not in any SLOT, the master will load this NEW locomotive address into a new SLOT ,[speed=0, FWD, Lite/Functions OFF and 128 step mode]and return this as a SLOT DATA READ. If no inactive slots are free to load the NEW locomotive address, the response will be the Long Acknowledgment ,(opcode <B4>,) , with a "fail" code, 0.

Note that regular "SHORT" 7 bit NMRA addresses are denoted by <loco-adr hi>=0. The Analog , Zero stretched, loco is selected when both <loco adr hi>=<loco adr lo>=0. <Loco adr lo> is always a 7 bit value. If <loco adr hi> is non-zero then the Master will generate NMRA type 14 bit or "LONG" address packets using all 14 bits from <loco adr hi> and <loco adr lo> with Loco adr Hi being the MOST significant address bits. Note that a DT200 Master does NOT process 14 bit adr requests and will consider the <loco adr hi> to always zero. You can check the <TRK> return bits to see if the Master is a DT200.

The throttle must then examine the SLOT READ DATA bytes to work out how to process the Master response. If the STATUS1 byte shows the SLOT to be COMMON, IDLE or NEW the throttle may change the SLOT to IN_USE by performing a <u>NULL MOVE</u> instruction ,(opcode

<BA>,<slotX>,<slotX>,<chk>) on this SLOT. **This activation mechanism is used to guarantee proper SLOT usage interlocking in a multi-user asynchronous environment.**

If the SLOT return information shows the Locomotive requested is IN_USE or UP-CONSISTED (i.e. the SL_CONUP, bit 6 of STATUS1 =1) the user should NOT use the SLOT. Any UP_CONSISTED locos must be UNLINKED before usage! Always process the result from the LINK and UNLINK commands, since the Master reserves the right to change the reply slot number and can reject the linking tasks under several circumstances. Verify the reply slot # and the Link UP/DN bits in STAT1 are as you expected.

The throttle will then be able to update Speed./Direction and Function information . Whenever SLOT information is changed in an active slot , the SLOT is flagged to be updated as the next DCC packet sent to the track. If the SLOT is part of linked CONSIST SLOTS the whole CONSIST chain is updated consecutively.

If a throttle is disconnected from the LocoNet□, upon reconnection (if the throttle retains the SLOT state from before disconnection) it will request the full status of the SLOT it was previously using. If the reported STATUS and Speed/Function data etc., from the master exactly matches the remembered SLOT state the throttle will continue using the SLOT. If the SLOT data does not match, the throttle will assume the SLOT was purged free by the system and will go through the setup "log on" procedure again.

With this procedure the throttle does not need to have a unique "ID number". SLOT addresses DO NOT imply they contain any particular LOCOMOTIVE address. The system can be mapped such that the SLOT address matches the LOCOMOTIVE address within, if the user directly Reads and Writes to SLOTs without using the Master to allocate Locomotive addresses

DISPATCHING:
Active Locomotives (including Consist TOP) SLOTS may be released for assignment to BT2 throttles in the "DISPATCH" mode. In this case a BT2 operating in its normal mode will request a DISPATCH SLOT that has been prepared by a supervisor type device. This is included for Club type operations where simpler throttles with limited capabilities are given to Engineers (Operators) by the Hostler or Dispatcher.

To DISPATCH PUT a slot , perform a **SLOT MOVE to Slot 0**. In this case the Destination Slot 0 is not copied to, but the source SLOT number is marked by the system as the DISPATCH slot. This is only a "one deep stack".

To DISPATCH GET, perform a **SLOT MOVE from Slot 0** (no destination needed). If there is a DISPATCH marked slot in the system, a SLOT DATA READ (<E7>,,,) with the SLOT information will be the response. If there is NO DISPATCH slot, the response will be a LONG ACK (opc <B4>,,) with the Fail code,00.

FUTURE EXPANSION CODES: (still in definition stage)
Immediate codes may be sent to the Master by a device. These are converted to DCC packets and sent as the next packet to the rails. They are not entered into any refresh stack. These are available in a system based on the DCS100/"Chief".

Opcodes for access to an auxiliary Service mode Programming Track are included. These requests are not entered in the main DCC packet stream .

Note that several confusing expansions and opcode sequences have been stripped from this LocoNet□ version. An experimenter who implements this protocol correctly should have no problems running on a LocoNet□ that has other expanded features. Again, we recommend resisting the temptation to "optimise" or take shortcuts with this protocol since it will lead to guaranteed future problems with your hardware and software.

LocoNet OPCODE SUMMARY: All Copyrights and rights reserved, Digitrax 1997
NOTE any OPcodes shown here in *itallics* are not finalised and are informational only. Do not use. All other OPCODES and states are reserved for future expansion.

```
;LocoNet  Personal Use version definitions   1.0
; DRAFT DEFINITIONS October 16, 1997 SUBJECT TO REVISION
```

;2 Byte MESSAGE opcodes

```
; FORMAT = <OPC>,<CKSUM>
;                                                          FOLLOW ON  RESPONSE
;                                                               MSG?   TYPE
OPC_IDLE            0x85   ,FORCE IDLE state, B'cast emerg. STOP  NO
OPC_GPON            0x83   ;GLOBAL power ON reque st              NO
OPC_GPOFF           0x82   ;GLOBAL power OFF req                  NO

OPC_BUSY            0x81   ;MASTER busy code, NUL                 NO
```

;4 byte MESSAGE OPCODES

```
; FORMAT = <OPC>,<ARG1>,<ARG2>,<CKSUM>
;
OPC_LOCO_ADR       0xBF   ;REQ  loco ADR                          YES   <E7>SLOT READ
                    ; <0xBF>,<0>,<ADR>,<CHK> REQ loco ADR
                   ;DATA return <E7>, is SLOT#,DATA that ADR was found in
                   ;IF ADR not found, MASTER puts ADR in FREE slot
                   ;and sends DATA/STATUS return <E7>......
                   ;IF no FREE slot,Fail  LACK,0 is returned   [<B4>,<3F>,<0>,<CHK>]

OPC_SW_ACK        0xBD   ;REQ SWITCH WITH acknowledge function (not DT200)   YES   LACK
                   ; <0xBD>,<SW1>,<SW2>,<CHK> REQ SWITCH function
     <SW1> =<0,A6,A5,A4- A3,A2,A1,A0>, 7 ls adr bits. A1,A0 select 1 of 4 input pairs in a DS54
     <SW2> =<0,0,DIR,ON- A10,A9,A8,A7>   Control bits and 4 MS adr bits.
                  ,DIR=1 for Closed,/GREEN, =0 for Thrown/RED
                  ,ON=1 for Output ON, =0 FOR output OFF
                   ;response is      <0xB4> <3D><00> if DCS100  FIFO is full,command rejected
                                     <0xB4><3D><7F> if DCS100 accepted

OPC_SW_STATE      0xBC   ;REQ state of SWITCH                           YES    LACK
                   ; <0xBC>,<SW1>,<SW2>,<CHK> REQ state of SWITCH

OPC_RQ_SL_DATA     0xBB   ;Request SLOT DATA/status block          YES <E7>SLOT READ
                   ; <0xBB>,<SLOT>,<0>,<CHK>  Request SLOT DATA/status block

OPC_MOVE_SLOTS     0xBA   ;MOVE slot SRC to DEST                   YES  <E7>SLOT READ
                   ; <0xBA>,<SRC>,<DEST>,<CHK> Move SRC to DEST if SRC       or LACK  etc
                  ; is NOT   IN_USE,  clr SRC
          ;SPECIAL CASES
                  ;If SRC=0 ( DISPATCH GET) , DEST=dont care, Return SLOT READ DATA of
                  ;DISPATCH Slot
                   ;IF SRC=DEST (NULL move) then SRC=DEST is set to IN_USE , if legal move
                   ;If DEST=0, is DISPATCH Put, mark SLOT as DISPATCH
                     ;RETURN slot status <0xE7> of DESTINATION slot DEST if move legal
                   ;RETURN Fail LACK code if illegal move <B4>,<3A>,<0>,<chk>
                  , illegal to move to/from slots 120/127
```

```
OPC_LINK_SLOTS     0xB9  ;LINK slot ARG1 to slot ARG2          YES   <E7>SLOT READ
                  ; <0xB9>,<SL1>,<SL2>,<CHK>  SLAVE slot SL1 to slot SL2
                  ;Master LINKER sets the SL_CONUP/DN flags appropriately
                 ,Reply is return of SLOT Status <0xE7>. Inspect to see result of Link
                 ,invalid Link will return Long Ack Fail <B4>,<39>,<0>,<CHK>

OPC_UNLINK_SLOTS   0xB8  ;UNLINK slot ARG1 from slot ARG2      YES   <E7>SLOT READ
                  ; <0xB8>,<SL1>,<SL2>,<CHK>  UNLINK slot SL1 from SL2
                  ;UNLINKER executes unlink STRATEGY and returns new SLOT#
                  ; DATA/STATUS of unlinked LOCO . Inspect data to evaluate UNLINK

CODES 0xB8 to 0xBF have responses

OPC_CONSIST_FUNC   0xB6  ;SET FUNC bits in a CONSIST uplink element  NO
                  ; <0xB6>,<SLOT>,<DIRF>,<CHK> UP consist FUNC bits
                  ;NOTE this SLOT adr is considered in UPLINKED slot space

OPC_SLOT_STAT1     0xB5  ;WRITE slot stat1                     NO
                  ; <0xB5>,<SLOT>,<STAT1>,<CHK> WRITE stat1

OPC_LONG_ACK       0xB4  ;Long acknowledge                     NO
                  ; <0xB4>,<LOPC>,<ACK1>,<CHK> Long acknowledge
                  ;<LOPC> is COPY of OPCODE responding to (msb=0).
                  ;LOPC=0 (unused OPC) is also VALID fail code
                  ;<ACK1> is appropriate response code for the OPCode

OPC_INPUT_REP      0xB2       ; General SENSOR Input codes      NO
                 ; <0xB2>, <IN1>, <IN2>, <CHK>

      <IN1>  =<0,A6,A5,A4- A3,A2,A1,A0>, 7 ls adr bits. A1,A0 select 1 of 4 inputs pairs in a DS54
      <IN2>  =<0,X,I,L- A10,A9,A8,A7>        Report/status bits and 4 MS adr bits.
             "I"=0 for DS54 "aux" inputs and 1 for "switch" inputs mapped to 4K SENSOR space.
             (This is effectively a least significant adr bit when using DS54 input configuration)
             "L"=0 for input SENSOR now 0V (LO) , 1 for Input sensor >=+6V (HI)
             "X"=1, control bit , 0 is RESERVED for future!

OPC_SW_REP         0xB1  ;Turnout  SENSOR state REPORT         NO
                  ; <0xB1>,<SN1>,<SN2>,<CHK> SENSOR state REPORT

      <SN1>  =<0,A6,A5,A4- A3,A2,A1,A0>, 7 ls adr bits. A1,A0 select 1 of 4 input pairs in a DS54
      <SN2>  =<0,1,I,L- A10,A9,A8,A7>        Report/status bits and 4 MS adr bits.
                              this <B1> opcode encodes input levels for turnout feedback
             "I" =0 for "aux" inputs (normally not feedback), 1 for "switch" input used for turnout
                   feedback for DS54 ouput/turnout # encoded by A0-A10
             "L" = 0 for this input 0V (LO), 1= this input > +6V (HI)
      alternately;
      <SN2>  =<0,0,C,T- A10,A9,A8,A7>        Report/status bits and 4 MS adr bits.
                              this <B1> opcode encodes current OUTPUT levels
             "C"= 0 if "Closed" ouput line is OFF, 1="closed" output line is ON (sink current)
             "T"=0 if "Thrown" output line is OFF, 1="thrown" output line is ON (sink I)

OPC_SW_REQ         0xB0  ;REQ SWITCH function                  NO
```

```
                  ; <0xB0>,<SW1>,<SW2>,<CHK> REQ SWITCH function
        <SW1> =<0,A6,A5,A4- A3,A2,A1,A0>, 7 ls adr bits. A1,A0 select 1 of 4 input pairs in a DS54
        <SW2> =<0,0,DIR,ON- A10,A9,A8,A7>   Control bits and 4 MS adr bits.
              ,DIR=1 for Closed,/GREEN, =0 for Thrown/RED
              ,ON=1 for Output ON, =0 FOR output OFF
     Note-,Immediate response of <0xB4><30><00> if command failed, otherwise no response

;"A" CLASS codes

CODES 0xA8 to 0xAF have responses

OPC_LOCO_SND      0xA2   ;SET SLOT sound functions              NO
OPC_LOCO_DIRF     0xA1   ;SET SLOT dir,F0-4 state               NO
OPC_LOCO_SPD      0xA0   ;SET SLOT speed                        NO
                          e.g. <A0><SLOT#><SPD><CHK>
```

; 6 Byte MESSAGE OPCODES

; FORMAT = <OPC>,<ARG1>,<ARG2>,<ARG3>,<ARG4>,<CKSUM>

<reserved>

; VARIABLE Byte MESSAGE OPCODES

; FORMAT = <OPC>,<COUNT>,<ARG2>,<ARG3>,.....<ARG(COUNT-3)>,<CKSUM>

```
OPC_WR_SL_DATA    0xEF   ;WRITE SLOT DATA, 10 bytes             YES    LACK
                  ; <0xEF>,<0E>,<SLOT#>,<STAT>,<ADR>,<SPD>,<DIRF>,<TRK>
                  ;<SS2>,<ADR2>,<SND>,<ID1>,<ID2>,<CHK>
                  ; SLOT DATA WRITE, 10 bytes data /14 byte MSG

OPC_SL_RD_DATA    0xE7   ;SLOT DATA return, 10 bytes            NO
                  ; <0xE7>,<0E>,<SLOT#>,<STAT>,<ADR>,<SPD>,<DIRF>,<TRK>
                  ;<SS2>,<ADR2>,<SND>,<ID1>,<ID2>,<CHK>
                  ; SLOT DATA READ, 10 bytes data /14 byte MSG

;NOTE; If STAT2.2=0 EX1/EX2 encodes an ID#,[if STAT2.2=1 the STAT.3=0 means EX1/EX2 are
ALIAS]
;ID1/ID2 are two 7 bit values encoding a 14 bit unique DEVICE usage ID
;ID1/ID2#'s      00/00         -means NO ID being used
;                01/00 to 7F/01 -ID shows PC usage.Lo nibble is TYP PC# (PC can use hi values)
;                00/02 to 7F/03 -SYSTEM reserved
;                00/04 to 7F/7E -NORMAL throttle RANGE
```

OPC_PEER_XFER   0xE5   ;move 8 bytes PEER to PEER, SRC->DST   NO resp

;<0xE5>,<10>,<SRC>,<DSTL><DSTH>,<PXCT1>,<D1>,<D2>,<D3>,<D4>,

;   <PXCT2>,<D5>,<D6>,<D7>,<D8>,<CHK>

;SRC/DST are 7 bit args. DSTL/H=0 is BROADCAST msg

;   SRC=0 is MASTER

;   SRC=0x70-0x7E are reserved

;SRC=7F is THROTTLE msg xfer, <DSTL><DSTH> encode ID#, <0><0> is THROT B'CAST

;<PXCT1>=<0,XC2,XC1,XC0 - D4.7,D3.7,D2.7,D1.7>

;XC0-XC2=ADR type CODE-0=7 bit Peer TO Peer adrs

```
;                     1=><D1>is SRC HI,<D2>is DST HI
;<PXCT2>=<0,XC5,XC4,XC3 - D8.7,D7.7,D6.7,D5.7>
;XC3-XC5=data type CODE- 0=ANSI TEXT string,balance RESERVED

OPC_IMM_PACKET    0xED   ;SEND n-byte packet immediate      LACK
          ;<0xED>,<0B>,<7F>,<REPS>,<DHI>,<IM1>,<IM2>,<IM3>,<IM4>,<IM5>,<CHK>
          ;<DHI>=<0,0,1,IM5.7-IM4.7,IM3.7,IM2.7,IM1.7>
          ;in <REPS> D4,5,6=#IM bytes,D3=0(reserved); D2,1,0=repeat CNT
               ;Not limited MASTER then LACK=<B4>,<7D>,<7F>,<chk> if CMD ok
               ;IF limited MASTER then  Lim  Masters respond with <B4>,<7E>,<lim adr>,<chk>
               ;If internal buffer BUSY/full respond with <B4>,<7D>,<0>,<chk>
(NOT IMPLEMENTED IN DT200)
```

Notes:

The SLOT DATA bytes are, in order of TRANSMISSION for <E7> READ or <EF> WRITE
NOTE SLOT 0 <E7> read will return MASTER config information bytes .

0) SLOT NUMBER: ;0-7FH, 0 is special SLOT, 070H-07FH DIGITRAX reserved:

1) SLOT STATUS1:

```
D7-SL_SPURGE   ;1=SLOT purge en,ALSO adrSEL (INTERNAL use only)
               ;                  (not seen on NET!)
               ;CONDN/CONUP: bit encoding-Control double linked Consist List
               ;11=LOGICAL MID CONSIST , Linked up AND down
D6-SL_CONUP    ;10=LOGICAL CONSIST TOP, Only linked downwards
               ;01=LOGICAL CONSIST SUB-MEMBER, Only linked upwards
               ;00=FREE locomotive, no CONSIST indirection/linking
               ;ALLOWS "CONSISTS of CONSISTS".  Uplinked means that Slot SPD
              ;number is now SLOT adr of SPD/DIR and STATUS of consist. i.e. is
              ;an Indirect pointer. This Slot has same BUSY/ACTIVE bits as TOP of
              ; Consist. TOP is loco with SPD/DIR for whole consist. (top of list).

               ;BUSY/ACTIVE: bit encoding for SLOT activity
D5-SL_BUSY     ;11=IN_USE loco adr in SLOT   -REFRESHED
D4-SL_ACTIVE   ;10=IDLE  loco adr in SLOT    -NOT refreshed
               ;01=COMMON loco adr IN SLOT    -refreshed
               ;00=FREE SLOT, no valid DATA    -not refreshed

D3-SL_CONDN   ;shows other SLOT Consist linked INTO this slot,see SL_CONUP

D2-SL_SPDEX ;        ;3 BITS for Decoder TYPE encoding for this SLOT
D1-SL_SPD14          ;011=send 128 speed mode packets
D0-SL_SPD28          ;010=14 step MODE
                     ;001=28 step. Generate Trinary packets for this Mobile ADR
                     ;000=28 step/ 3 BYTE PKT regular mode
                     ;111=128 Step decoder, Allow Advanced DCC consisting
                     ;100=28 Step decoder ,Allow Advanced DCC consisting
```

2) SLOT LOCO ADR: ;LOCO adr Low 7 bits (byte sent as ARG2 in ADR req opcode <BF>)

3) SLOT SPEED: ;0x00=SPEED 0 ,STOP inertially;0x01=SPEED 0 EMERGENCY stop
;0x02->0x7F increasing SPEED,0x7F=MAX speed

(byte also sent as ARG2 in SPD opcode <A0>)

4)SLOT DIRF byte: (byte also sent as ARG2 in DIRF opcode <A1>)

D7-0	;always 0
D6-SL_XCNT	; reserved , set 0
D5-SL_DIR	;1=loco direction FORWARD
D4-SL_F0	;1=Directional lighting ON
D3-SL_F4	;1=F4 ON
D2-SL_F3	;1=F3 ON
D1-SL_F2	;1=F2 ON
D0-SL_F1	;1=F1 ON

5) TRK byte: (GLOBAL system /track status)

D7-D4 Reserved

D3	GTRK_PROG_BUSY	1=Programming TRACK in this Master is BUSY.
D2	GTRK_MLOK1 ,	1=**This Master IMPLEMENTS LocoNet 1.1 capability**
	,	0=Master is DT200
D1	GTRK_IDLE :	0=TRACK is PAUSED, B'cast EMERG STOP.
D0	GTRK_POWER :	1=DCC packets are ON in MASTER, Global POWER up

6) SLOT STATUS2:
D3- 1=expansion IN ID1/2, 0=ENCODED alias
D2- 1=Expansion ID1/2 is NOT ID usage
D0- 1=this slot has SUPPRESSED ADV consist-

7) SLOT LOCO ADR HIGH: Locomotive address high 7 bits. If this is 0 then Low address is normal 7 bit NMRA SHORT address. If this is not zero then the most significant 6 bits of this address are used in the first LONG address byte (matching CV17). The second DCC LONG address byte matches CV18 and includes the Adr Low 7 bit value with the LS bit of ADR high in the MS postion of this track adr byte.
Note a DT200 MASTER will always interpret this as 0.

8) SLOT SOUND: Slot sound/ Accesory Function mode II packets. F5-F8
(byte also sent as ARG2 in SND opcode)
D7-D4 reserved
D3-SL_SND4/F8
D2-SL_SND3/F7
D1-SL_SND2/F6
D0-SL_SND1/F5 ;1= SLOT Sound 1 function 1active (accessory 2)

9) EXPANSION RESERVED ID1: 7 bit ls ID code written by THROTTLE/PC when STAT2.4=1

10) EXPANSION RESERVED ID2: 7 bit ms ID code written by THROTTLE/PC when STAT2.4=1

Stationary Broadcast Command:
Note that a 3 byte DCC track packet configured as:
<sync> ,<1011-1111>,<1000-D c b a > <ecb> is a DCC Broadcast Address to Stationary decoders.

Broadcast LocoNet Switch adr is then <SW2>=<0,0,a,D-1,1,1,1>, <SW1>=<0,1,1,1-1,0,c,b>

Stationary Interrogate Command:

The DCC packet <sync>,<1011-1111>,<1100-D c b a> <ecb>is an Interrogation for all DS54's. This causes a 2 LocoNet <B1> messages encoding both Output state and Input state, for each sensor adr a/b/c encodes.
Interrogate LocoNet Switch adr is <SW2>=<0,0,a,1-0,1,1,1>, <SW1>= <0,1,1,1-1,0,c,b>
This is generated by DCS100 at power ON, and scans all 8 inputs of all DS54's.

Programmer track:

The programmer track is accessed as Special slot #124 ($7C, 0x7C). It is a full asynchronous shared system resource.
To start Programmer task, write to slot 124. There will be an immediate LACK acknowledge that indicates what programming will be allowed. If a valid programming task is started, then at the final (asynchronous) programming completion, a Slot read <E7> from slot 124 will be sent. This is the final task status reply.

Programmer Task Start:

<0xEF>,<0E>,<7C>,<PCMD>,<0>,<HOPSA>,<LOPSA>,<TRK>;<CVH>,<CVL>,<DATA7>
,<0>,<0>,<CHK>

This OPC leads to immediate LACK codes:

<B4>,<7F>,<7F>,<chk>	Function NOT implemented, no reply.
<B4>,<7F>,<0>,<chk>	Programmer BUSY , task aborted, no reply.
<B4>,<7F>,<1>,<chk>	Task accepted , <E7> reply at completion.
<B4>.<7F>,<0x40>,<chk>	Task accepted blind NO <E7> reply at completion.

Note that the <7F> code will occur in Operations Mode Read requests if the System is not configured for and has no Advanced Acknowlegement detection installed.. Operations Mode requests can be made and executed whilst a current Service Mode programming task is keeping the Programming track BUSY. If a Programming request is rejected, delay and resend the complete request later. Some readback operations can keep the Programming track busy for up to a minute. Multiple devices, throttles/PC's etc, can share and sequentially use the Programming track as long as they correctly interpret the response messages . Any Slot RD from the master will also contain the Programmer Busy status in bit 3 of the <TRK> byte.

A <PCMD> value of <00> will abort current SERVICE mode programming task and will echo with an <E6> RD the command string that was aborted.

<PCMD> Programmer Command: Defined as

D7	-0
D6	-Write/Read , 1= Write, 0=Read
D5	-Byte Mode , 1= Byte operation, 0=Bit operation (if possible)
D4	-TY1 Programming Type select bit
D3	-TY0 Prog type select bit
D2	-Ops Mode, 1=Ops Mode on Mainlines, 0=Service Mode on Programming Track
D1	-0 reserved
D0	-0-reserved

Type codes:

Byte Mode	Ops Mode	TY1	TY0	Meaning
1	0	0	0	Paged mode byte Read/Write on Service Track
1	0	0	0	Paged mode byte Read/Write on Service Track
1	0	0	1	Direct mode byteRead/Write on Service Track
0	0	0	1	Direct mode bit Read/Write on Service Track
x	0	1	0	Physical Register byte Read/Write on Service Track
x	0	1	1	Service Track- reserved function
1	1	0	0	Ops mode Byte program, no feedback
1	1	0	1	Ops mode Byte program, feedback
0	1	0	0	Ops mode Bit program, no feedback
0	1	0	1	Ops mode Bit program, feedback

<HOPSA>Operations Mode Programming- 7 High address bits of Loco to program, 0 if Service Mode
<LOPSA>Operations Mode Programming- 7 Low address bits of Loco to program, 0 if Service Mode
<TRK> Normal Global Track status for this Master, Bit 3 also is 1 WHEN Service Mode track is BUSY.

<CVH> High 3 BITS of CV#, and ms bit of DATA.7 <0,0,CV9,CV8 - 0,0, D7,CV7>

<CVL> Low 7 bits of 10 bit CV address. <0,CV6,CV5,CV4-CV3,CV2,CV1,CV0>
<DATA7>Low 7 BITS OF data to WR or RD COMPARE <0,D6,D5,D4 - D3,D2,D1,D0>
ms bit is at CVH bit 1 position.

__**Programmer Task Final Reply:**__ (if saw LACK <B4>,<7F>,<1>,<chk> code reply at task start)

<0xE7>,<0E>,<7C>,<PCMD>,<PSTAT>,<HOPSA>,<LOPSA>,<TRK>;<CVH>,<CVL>,<DATA7>
,<0>,<0>,<CHK>

<PSTAT> Programmer Status error flags. Reply codes resulting from completed task in PCMD

D7-D4	-reserved
D3	-1=User Aborted this command
D2	-1= Failed to detect READ Compare acknowledge response from decoder
D1	-1= No Write acknowledge response from decoder
D0	-1= Service Mode programming track empty- No decoder detected

This <E7> response is issued whenever a Programming task is completed. It echos most of the request information and returns the PSTAT status code to indicate how the task completed. If a READ was requested <DATA7> and <CVH> contain the returned data, if the PSTAT indicates a successful readback (typically =0). Note that if a Paged Read fails to detect a successful Page write acknowledge when first setting the Page register, the read will be aborted, showing no Write acknowledge flag D1=1

__**FAST Clock:**__ The system FAST clock and parameters are implemented in Slot#123 <7B>.

Use <EF> to write new clock information, Slot read of 0x7B,<BB><7B>.., will return current System clock information, and other throttles will update to this SYNC. Note that all attached display devices keep a current clock calculation based on this SYNC read value, i.e. devices MUST not continuously poll the clock SLOT to generate time, but use this merely to restore SYNC and follow current RATE etc. This clock slot is typically "pinged" or read SYNC'd every 70 to 100 seconds , by a single user, so all attached devices can synchronise any phase drifts. Upon seeing a SYNC read, all devices should reset their local sub-minute phase counter and invalidate the SYNC update ping generator.

<u>Clock Slot Format</u>:
<0xEF>,<0E>,<7B>,<CLK_RATE>,<FRAC_MINSL>,<FRAC_MINSH>,<256-MINS_60>,<TRK>
;<256-HRS_24>,<DAYS>,<CLK_CNTRL>,<ID1>,<ID2>,<CHK>

<CLK_RATE>, 0=Freeze clock, 1=normal 1:1 rate, 10=10:1 etc, max VALUE is 7F/128 to 1

<FRAC_MINSL>, FRAC mins hi/lo are a sub-minute counter , depending on the CLOCK generator
<FRAC_MINSH>, Not for ext. usage. This counter is reset when valid <E6><7B> SYNC msg seen

<256-MINS_60>, This is FAST clock MINUTES subtracted from 256. Modulo 0-59
<256-HRS_24>,This is FAST clock HOURS subtracted from 256. Modulo 0-23
<DAYS>, number of 24 Hr clock rolls, positive count
<CLK_CNTRL> Clock Control Byte
D6- 1=This is valid Clock information, 0=ignore this <E6><7B>, SYNC reply
<ID1>,<ID2> This is device ID last setting the clock. <00><00> shows no set has happened
<7F><7x> are reserved for PC access

[END]