J1939 Bootloader

Created by the J1939 Experts

1.1 J1939 Bootloader Overview:

Our J1939 Bootloader provides a universal and compact solution for reprogramming ECUs quickly, efficiently and securely. Our bootloaders have been in use worldwide for over fifteen years, beginning with J1939 and CAN based system. The embedded firmware has been split into three parts:

1) Bootloader
2) Midpoint tables
3) Application

When the CPU comes out of reset, the bootloader decides if there is valid application software. If there isn't, the flash loader is started and attempts to receive an application over the J1939 network. If there was a valid application, the application is started. Additional, the application has the ability to start the bootloader. This is typically done if the application receives a start command over the J1939 network.

The midpoint tables allow for abstraction between the application and the bootloader. The tables contain the following information: start addresses for the application and bootloader, application checksum value, application is present flag, and bootloader version number. These tables reside at a fix memory location in flash and should not be moved. If tables are moved, future versions of application code may not be backwards compatible with previous versions of bootloaders. Also, due to the S12X family 16-bit architecture both tables must reside in non-banked memory.

Memory Layout

Category	Start	End
Application Area #1	0x78_0000	0x7F_3FFF
Midpoint Tbl – App	0x7F_4000	0x7F_400F
Application Area #2	0x7F_4010	0x7F_BFFF
Midpoint Tbl - BL	0x7F_C000	0x7F_C00F
Bootloader	0x7F_C010	0x7F_FFFF

1.2 flashJ1939:

The PC bootloader application, named flashJ1939.exe, works with Simma Software VNA-Value low-cost J1939 to USB adapter to send the new flash image across the J1939 network. flashJ1939 allows the user to specify the J1939 address of the target ECU and the s-record or hex file to be used.

Example:

```
flashJ1939 232 1 app.srec
```

1.3 Points of Interest for Bootloader:

1.3.1 The bootloader only supports linear global addressing. Therefore, any s-record sent across the J1939 network must not use banked addressing.

1.3.2 The application checksum is currently generated via flashJ1939 when reading in the application file. It is then sent across the network and stored into flash memory so application validation can happen every power cycle.

1.4 Programming Message Flow Overview:

Step 1:

When an ECU is going to be programmed using the method described in this document, the first thing the PC needs to do is request the boot version of the target ECU system. This allows the PC or the user to decide if the new software, which is destined for the ECU, is compatible with the current boot version. If the entire flash is being updated, the boot version can be ignored since the new boot manager will overwrite the old boot manager.

Step 2:

The second message that is sent to the ECU will be either 'start programming entire flash' or 'start programming application flash.' The PC should then wait for a 'request for data' from the ECU. If the PC has not received the response, it may resend indefinitely, each time waiting for a 'request for data.'

Step 3: (J1939)

Next, the PC should send one address packet followed by at most twelve data packets. The address packet contains the number of following data bytes and also the starting address for the data. The sequence number in the address packet should match the sequence number in the first data packet, and then increment for each additional data packet. The sequence number ranges from 0 to 255, and overflows to 0. After sending the address and data packets, the PC should wait for another 'request for data.' This process continues until the PC has no more available data to supply the ECU. For the last transmission, it is anticipated that the PC probably will not have a full set of twelve packets to send. The ECU will respond be requesting data packets at the point where the PC halted transmission. If the requests are for unavailable data, the PC should ignore them.

Step 4:

The PC should now transmit a 'stop programming flash' message with the set and sequence number of the last available data packet. The PC waits for an 'acknowledge', and may resend until an 'acknowledge' is received.

Step 5:

This step is optional. Once the entire programming session has been completed, the ECU should reset itself. The PC waits for this to occur, then listens to the communications network for messages being sent out from the ECU. If no messages are being broadcast from the ECU, the PC may request data from the ECU. Once the PC has received messages from the ECU, it should display a ‘programming successful’ message to the user.

1.5 J1939 Packet Structure:

CAN 29-Bit ID			Data[0..7]	
PGN	DST	SRC	PPGN	Parameters[0..6]

The Proprietary A PGN (PDU1) is used for the J1939 bootloader, this packet has a PGN of 61184. There are three types of packets: Address (PPGN of 5), Data (PPGN of 3), and Command packets (PPGN of 4). Data is sent most significant byte first, and all data is unsigned except where noted.

<u>Address Packets:</u>
PPGN equals 5:
Parm[0]: sequence number of packet. Range is from 0 to 255.
Parm[1]: number of data bytes in following data packets.
Parm[2-5]: 32-bit starting address for following data

<u>Data Packets:</u>
PPGN equals 3:
Parm[0]: sequence number of packet. Range is from 0 to 255.
Parm[1-6]: data bytes

<u>Command Packets:</u>
PPGN equals 4:
Parm[0] equals 1: Start programming entire flash
DLC: 3 bytes
Notes: Parm[1] reserved for future use

Parm[0] equals 2: Start programming application flash
DLC: 3 bytes
Notes: Parm[1] reserved for future use

Parm[0] equals 3: Stop programming flash
DLC: 3 bytes
Parm[1]: sequence number of last data packet

Parm[0] equals 4: Request for command packet
DLC: 3 bytes
Parm[1]: PID for information (non-data) being requested

Parm[0] equals 5: Acknowledge

DLC: 7 bytes
Parm[1]: PID of packet being acknowledged

Parm[0] equals 6: Flash Erase Error
DLC: 3 bytes
Parm[1]: Flash error code (refer to Motorola’s an2720.pdf or flash.h for error codes)

Parm[0] equals 7: Flash Write Error
DLC: 3 bytes
Parm[1]: Flash error code (refer to Motorola’s an2720.pdf or flash.h for error codes)

Parm[0] equals 8: Boot manager version
DLC: 4 bytes
Parm[1-2]: Boot manager version in binary. Represents 0.0 to 255.255

Parm[0] equals 9: Request for data
DLC: 3 or 7 bytes
Note: On first requests, the recommended start address may be available.
Parm[1]: sequence number being requested
Parm[2-5]: recommended start address for the data being requested