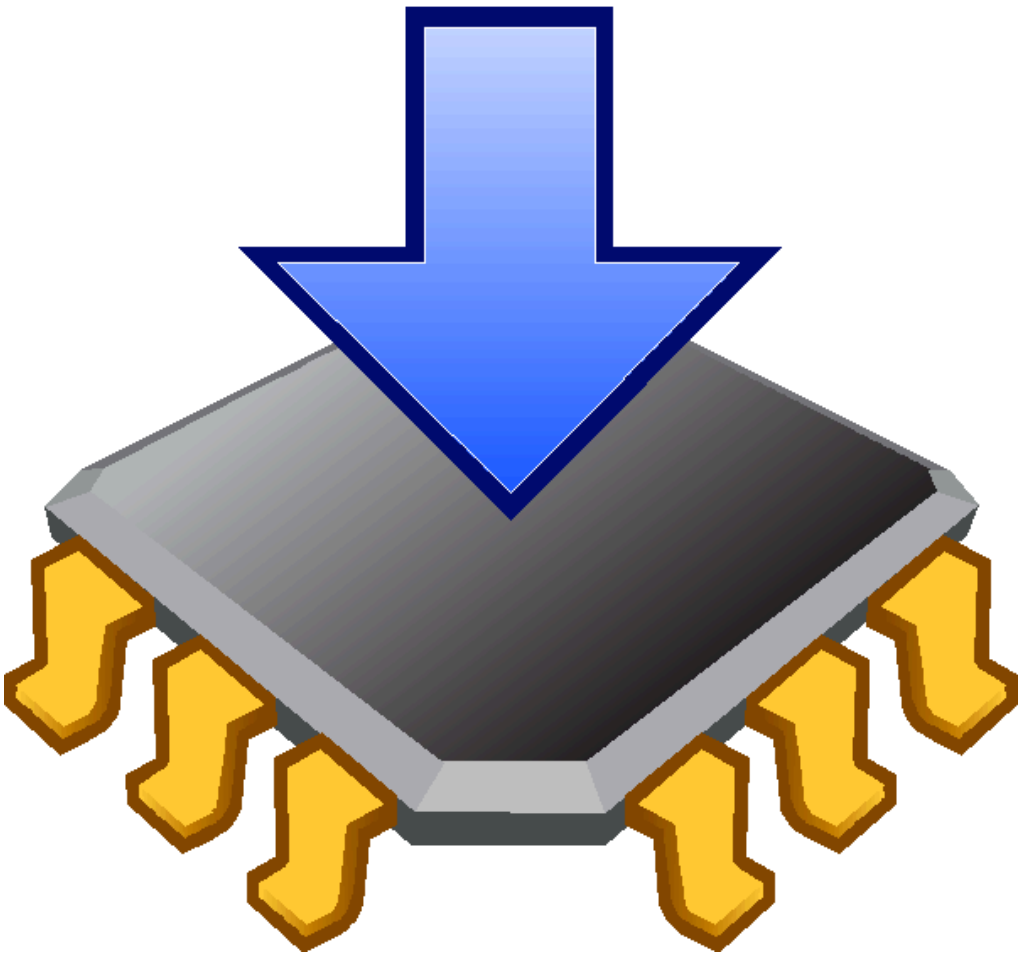
STN Bootloader

Firmware Update Specification for Devices with STN Bootloader



OBD SOLUTIONS®

Table of Contents

TO OUR VALUED CUSTOMERS

It is our intention to provide our valued customers with the best documentation possible to ensure successful use of your OBD Solutions products. To this end, we will continue to improve our publications to better suit your needs. Our publications will be refined and enhanced as new volumes and updates are introduced.

Latest Documentation

To obtain the most up-to-date version of this document, please visit our website at http://www.obdsol.com.

You can determine the version by examining its literature number found on the bottom outside corner of any page. The last character of the literature number is the version number, (e.g., **STNBLA** is version A of document **STNBL**).

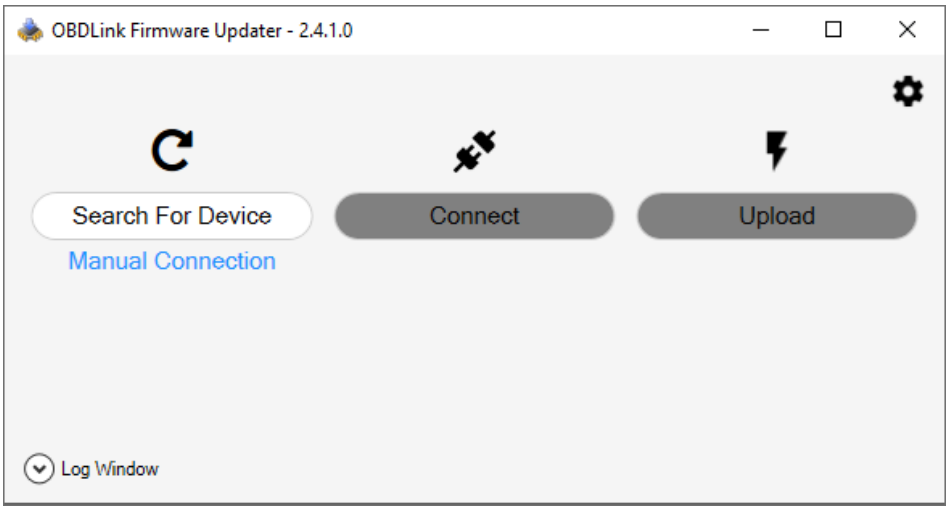
1.0 Introduction

STN Bootloader is a resident program that is factory programmed into devices designed by OBD Solutions. Its job is to write application firmware to the device's flash memory, allowing the device to be easily updated in the field. The updates can be used to remotely fix problems and add new functionality without the need for costly product recalls.

OBD Solutions provides a free utility called **STN Firmware Updater** (Figure 1) which can be used to upload new firmware from any computer that supports the Microsoft .NET framework. This utility is bundled with every official firmware release.

Unfortunately, in certain circumstances (e.g., in embedded or Linux environments), using the STN Firmware Updater may not be feasible. This document is intended to be used as a reference to allow programmers to implement the necessary algorithms for interfacing with STN Bootloader (versions 2.x, 3.x, and 4.x). Sample code is available upon request.

Figure 1 - Stn Firmware Updater



2.0 Features

STN Bootloader has a number of properties that make it ideal for carrying out remote updates:

1. **Tamper-proof.** The bootloader resides in a designated 'boot block', a small section of protected program memory, which is separate from application program memory. The firmware images are protected from tampering using strong military-grade encryption.
2. **Brick-proof.** The bootloader checks the firmware that is being uploaded, to make sure it is compatible with the device. If the upload is interrupted, it can be safely restarted.
3. **Superior noise immunity.** Updates can be done even over unreliable and noisy connections. The bootloader features a robust error recovery mechanism, and the integrity of each data packet is ensured by a 16-bit CRC.
4. **Automatic baud rate detection** allows the firmware updater application to negotiate the optimal communication baud rate.

3.0 Basic Operation

Devices that feature the STN bootloader operate in three distinct modes:

1. Startup mode
2. Bootloader mode
3. Normal mode

On startup, or after a hard reset, the device is in **startup mode**, where it waits for a predefined amount of time (by default, 200 milliseconds) for the bootloader session to be initiated. If during this time the device receives a *Connect* command from the host, it enters **bootloader mode**, and remains there until a reset. Note that the device will not respond to any bootloader commands until it receives a valid *Connect* command.

If the bootloader session is not initiated within the startup mode window, and valid firmware is present, the device will enter **normal mode** and run the application firmware.

If the firmware is not valid (e.g., a firmware upload was interrupted by a power loss or a reset) the device will remain in **startup mode** indefinitely, until it receives the *Connect* command.

To enter **bootloader mode**, the host application must first reset the device by cycling its power, pulsing the RESET pin, or issuing a reset command ('ATZ' for OBDLink devices). After a short pause to give the device a chance to restart (approximately 50 ms), send the *Connect* command to the bootloader. If the device replies with a "Connect ACK", consider the bootloader session started and proceed with the firmware upload. See Section 7.0, "Updating Firmware" for more information.

4.0 Communication Protocol

The STN bootloader employs a basic communication protocol that is robust, simple to use, and easy to implement.

Packet level flow control is built into the protocol. Thus, for every received command, there is a response (an ACK or a NACK).

All multibyte values are big-endian and are transmitted most significant byte first.

4.1 UART Settings

The bootloader communicates with the host via UART. The following communication settings are used:

- 8 data bits
- No parity
- 1 stop bit
- Automatic baud rate detection

The bootloader supports a wide range of baud rates. All standard RS-232 baud rates are supported in addition to a few higher baud rates. The following formula can be used to calculate all supported baud rates (expressed in bits per second):

$$Baud\ Rate\ = \frac{10{,}000{,}000}{divisor}$$

where *divisor* is an integer between 1 and 65,536.

Baud rate is automatically detected during the reception of the first <STX> character of each command packet. The bootloader will reply on the same baud rate.

4.2 Packet Format

All data that is transmitted to or from the device follows this basic packet format:

```
<STX><STX>[<DATA1>…<DATAn>]<CRCH><CRCL><ETX>
```

where each <…> represents a byte and […] represents the data field. The start of a packet is indicated by two 'Start of TeXt' control characters (<STX>) and is terminated by a single 'End of TeXt' control character (<ETX>). The last two bytes before the <ETX> are the 16-bit CCITT CRC.

4.3 Control Characters

Three control characters have special meaning. Two of them, <STX> and <ETX>, were introduced in the previous section. The third control character is the <DLE> ('Data Link Escape'), described later in this section. Table 1 provides a summary of the three control characters.

Table 1 - Control Characters

Control	Value	Description
<STX>	0x55	Start of TeXt
<ETX>	0x04	End of TeXt
<DLE>	0x05	Data Link Escape

The <DLE> is used to identify a value that could be interpreted in the data field or CRC as a control character. Within the data field or CRC, the bootloader will always accept the byte following a <DLE> as data, and will always send a <DLE> before bytes 0x55, 0x04, and 0x05 that are part of data or CRC and should not be interpreted by the receiver as control characters.

For example, if a byte of value 0x55 is transmitted as part of the data field, rather than as the <STX> control character, the <DLE> character is inserted before the 0x55 byte. In other words, the following response packet (hex):

55	55	49	04	4C	55	8A	05	9B	92	04
STX	**STX**									**ETX**

will be transmitted as

55	55	49	05	04	4C	05	55	8A	05	05	9B	92	04
STX	**STX**		**DLE**			**DLE**			**DLE**				**ETX**

The process of using <DLE> to escape data bytes that may be misinterpreted as control characters, is called "**byte stuffing**".

Note: *Control characters are not considered data and are not included in the CRC calculation.*

4.4 CRC

The error detection during communication is accomplished using a standard 16-bit CCITT CRC (XModem) algorithm. Table 2 details the CRC parameters. Appendix B: CRC Sample Code lists sample code for the CRC calculation.

Table 2 - CRC Parameters

Width	16 bits
Polynomial	0x1021
Initial Value	0x0000
Final XOR Value	0x0000
Reflection	none
Check Value	0x31C3

CRC is calculated on the data field prior to byte stuffing. It is transmitted after the data field, and before the <ETX> control character.

After a response is received, it must first be unstuffed. Then the CRC is calculated over the entire data packet (including the CRC bytes). If the reception is successful, the CRC calculation result will equal zero.

4.5 Commands

The data field of each packet transmitted to the bootloader should contain one command and (optionally) its associated data. Commands must be transmitted to the bootloader in the following format:

```
<command><data length>[<data>]
```

Table 3 details the command format. Commands supported by the bootloader are detailed in Section 5.0 “Bootloader Commands”.

Table 3 - Command Format

Field	Length (bytes)	Description
<command>	1	Bootloader command
<data length>	2	Length of command payload
[<data>]	variable	Optional command payload

Bytes comprising a command do not have to be sent back-to-back. However, there cannot be more than 200 milliseconds between any two bytes. If this timeout occurs, the bootloader will abort receiving the command, and will revert to looking for the start of the next command packet.

4.6 Responses

The data field of each packet received from the bootloader contains one response and its associated data. The responses are transmitted by the bootloader in the following format:

```
<ACK/NACK><command><data length>[<data>]
```

Table 4 details the response format.

Table 4 - Response Format

Field	Length	Description
<ACK> <NACK>	2 bits	Success/error indication: 01 = ACK 10 = NACK
<command>	6 bits	Command which this response is acknowledging
<data length>	1 byte	Length of payload response
[<data>]	variable	Optional response payload

4.6.1 ACK Responses

Upon successful execution of each command, the bootloader responds with an ACK response in the following format:

```
<0x40|command><data length>[<data>]
```

See Section 5.0 “Bootloader Commands” for details on responses to each bootloader command.

4.6.2 NACK Responses

When the bootloader encounters an error while processing a command, it will respond with a NACK response. NACK responses have a fixed length of three bytes (one data byte) and are in the following format:

```
<0x80|command><0x01><error code>
```

Table 5 lists possible error codes.

Table 5 - Error Codes

Error Type	Error Code (hex)	Description
Communication	01	CRC error
	02	Received packet too long
Command Validation	10	Unknown command
	11	Invalid command format
Firmware Upload	30	Sequence error
	50	Authentication error
	80	Programming error
	90	Verification error
	A4	Firmware version error

4.6.2.1 Communication Errors

NACK responses indicating communication errors have their command field set to 0x00, because the integrity of the packet has been compromised or could not be verified, and the byte received in place of the command field may not correspond to the command actually sent by the host. These errors may occur for any bootloader command. The following two errors indicate communication problems:

CRC

The bootloader responds with this error when the received packet fails the CRC check. To recover from this error, resend the packet.

Received Packet Too Long

Whenever the size of the bootloader's internal buffer is exceeded, it will stop receiving the command and wait for the <ETX> character; after it receives the <ETX>, the bootloader will respond with the Packet Too Long error.

When properly formatted, none of the commands with fixed data length will generate this error. However, the *SendChunk* command accepts a variable data length parameter, and may generate this error if the data length exceeds the maximum chunk size supported by the bootloader. See *StartUpload* command description for more information.

4.6.2.2 Command Validation Errors

After the bootloader receives a packet and verifies the CRC, it proceeds to validate the command. The following two errors may be generated for any command received by the bootloader:

Unknown Command

<*command*> field of a packet received by the bootloader does not match any of the supported commands.

Invalid Command Format

This error is generated by an incorrect formatting of the <*data*> field. This can happen if the data length is incorrect, or the data values are outside of the supported range.

4.6.2.3 Firmware Upload Errors

Firmware upload errors can occur at any point during the firmware upload sequence. The two commands that can generate these errors are *StartUpload* and *SendChunk*.

Sequence

This error is returned when a firmware upload sequence error is detected. The conditions that can cause this error are:

- *SendChunk* command received before *StartUpload* command
- Chunk number is out of sequence
- The number of bytes received by the bootloader exceeds the number specified by the <*image size*> field of the *StartUpload* command

Authentication

This error is returned if the host software attempts to upload a firmware image for the wrong device. It can also be caused by a corrupted firmware image.

If an attempt is made to upload an invalid firmware image, most of the time this is detected within the first few packets and the old firmware will remain intact.

Programming

This error is returned if the bootloader encounters any errors while attempting to program the uploaded firmware. To attempt a recovery, reset the device and try uploading the firmware again.

Verification

This error is similar to the Programming error. However, it has a special meaning for firmware images of the Validation type. See Section 7.0, "Updating Firmware" for more information.

Firmware version

This error is returned if the bootloader determines the firmware image you are trying to upload is below the minimum firmware version for this device. See Section 5.2, "Device Information Commands" for more information.

5.0 Bootloader Commands

5.1 Control Commands

Control commands are used for establishing and terminating the bootloader session and to aid in error recovery. All control commands have their data length field set to 0x0000. Table 6 summarizes control commands supported by the current version of STN Bootloader.

Table 6 - Control Commands

Command	Name	Description
03	Connect	Start bootloader session
02	Reset	Reset device
01	ResendLast	Resend last response
05	SetCommParams	Set communication parameters

Connect

Start bootloader session. After successfully receiving this command, the bootloader turns off the startup timer, and the device will remain in bootloader mode until it is reset (via hardware reset or the *Reset* command). The bootloader will not respond to any other command with either 'ACK' or 'NACK' unless it receives the *Connect* command.

Reset

Terminate bootloader session, and reset the device. Send *Reset* after successfully uploading the firmware, to restart the device in normal mode.

ResendLast

Resend last response. This command can be used to recover from a communication error during reception of a bootloader response (e.g., UART framing error or a failed CRC check).

SetCommParams

Set communication session parameters. This command can be sent any time during a communication session to change its settings.

Table 7 details *SetCommParams* command data field format.

Table 7 - SetCommParams Command Format

Field	Length (bytes)	Description
<interbyte timeout>	2	Interbyte timeout in milliseconds. Once a command reception has started, if the STN Bootloader does not receive any data for this period of time, it cancels command reception and goes back to waiting for new commands. Default is 200 ms.

5.2 Device Information Commands

Table 8 summarizes device information commands supported by the current version of STN Bootloader. These commands can be used in bootloader mode to request information about the device. All device information commands must have their data length field set to 0x0000.

Table 8 - Device Information Commands

Command	Name	Description
06	GetVersion	Get bootloader version
07	GetDevID	Get device ID
08	GetHwRev	Get hardware revision
0A	GetSerialNumber	Get serial number
0B	GetDeviceName	Get device name (up to 32 char)
0D	GetDeviceNameEx	Get device name (up to 64 char)
0F	GetFwStatus	Get firmware status (valid/invalid)
2E	GetMinFwVer	Get minimum firmware version

GetVersion

Get bootloader version. Returns bootloader version in the following format (data length = 2):

```
<major><minor>
```

GetDevID

Get device ID. Returns device ID as a big-endian 16-bit integer (data length = 2). See Appendix C: Device IDs for the list of possible device IDs.

GetHwRev

Get device hardware revision. Returns device hardware revision in the following format (data length = 2):

```
<major><minor>
```

GetSerialNumber

Get device serial number. Returns serial number as 8 ASCII characters (data length = 8).

GetDeviceName

Get device name. Returns device name as an ASCII string. Data length is fixed at 32 bytes. For strings shorter than 32 bytes, the remaining data bytes will be padded with 0s.

GetDeviceNameEx

Get device name extended. Returns device name as an ASCII string. Data length is variable up to 64 bytes. Unlike GetDeviceName, the string returned by this command is not padded with 0s.

GetFwStatus

Get firmware status. Reports whether valid application firmware is present (data length = 1). If valid firmware is not present, returns 0; any other value indicates valid firmware.

Use this command to verify that the firmware was properly programmed after upload.

GetMinFwVer

Get minimum firmware version. Returns 4 8-bit hex integers in the format of major, minor, wip, build. The build field is currently unused and will be 0.

```
<major><minor><wip><build>

05060C00
```

will be decoded as

```
5.6.12
```

Returns data length of 0 if no minimum firmware version is set.

5.3 Firmware Upload Commands

Table 9 summarizes firmware upload commands supported by the current version of STN Bootloader, which are used to upload each firmware image as specified by the firmware file.

Table 9 - Firmware Upload Commands

Command	Name	Description
30	StartUpload	Start firmware image upload
31	SendChunk	Send the next firmware image chunk

StartUpload

Start firmware image upload. Every firmware image upload must be preceded by a *StartUpload* command. The *StartUpload* command packet has the following format:

```
<0x30><0x0004>[<image size><mode>]
```

Table 10 details the data field format of the *StartUpload* command.

Table 10 - StartUpload Command Format

Field	Length (bytes)	Description
<image size>	3	Firmware image size
<mode>	1	Mode. Must be set to 0x01

The response to *StartUpload* command is a big-endian 16-bit integer specifying the maximum chunk size that can be accepted by the bootloader (data length = 2).

SendChunk

Send the next firmware image chunk. Firmware chunk length must be a multiple of 16 bytes, and cannot exceed the maximum chunk size returned by the *StartUpload* command. The chunk size can be varied to achieve the optimum balance between firmware upload speed, progress granularity, and the speed of error recovery.

The *SendChunk* command packet has the following format:

```
<0x31><chunk_len+2>[<chunk num><chunk>]
```

Table 11 details *SendChunk* command data field format.

Table 11 - SendChunk Command Format

Field	Length (bytes)	Description
<chunk num>	2	Chunk number
<chunk>	variable	Firmware image chunk data

Chunk number is a sequential number assigned to each chunk. It must be set to 0 for the first chunk and then incremented by one for each subsequent chunk.

The response to the *SendChunk* command contains the 16-bit chunk number (data length = 2).

6.0 Firmware File

Firmware images to be used with STN Bootloader are released as binary files with a .bin extension. The STN firmware file format is detailed in Appendix A: Firmware File Format.

The current file version is 5. If any other file version is detected, you should abort the upload, because the firmware may not be compatible with the file layout, described in this document.

A firmware file consists of the following sections:

1. File header
2. One or more firmware images

6.1 Firmware File Header

6.1.1 File Signature

Firmware file begins with the 8-byte file identification block: 6-byte file signature, which can be used to identify a valid STN Firmware File, and a 2-byte file version field.

6.1.2 Device IDs

Device IDs block defines all the devices that the firmware file is compatible with. Do not attempt uploading the firmware to a device that's not on this list.

Device IDs block consists of a 1-byte device ID count field, followed by an array of 2-byte device IDs.

6.1.3 Firmware Image Descriptors

Firmware image descriptors are present for files that contain multiple firmware images. This block is optional, and if the file contains a single image, the firmware image descriptor count field will be set to 0. In which case, the rest of the file will be the single firmware image. The single image size can be found by subtracting the FW Image Descriptors Count file location from the entire file size. For the single image, the image type will be *Normal* (see Table 14).

If the count field is non-zero, it is followed by the array of 12-byte firmware descriptors.

Firmware descriptors instruct the updater on which firmware images to upload and in which order. The firmware descriptor array is a linked list of firmware images. The upload always starts with the firmware at index 0. However, based on the firmware image type, specified in its descriptor, the next firmware image to upload is determined by either the Next FW Index or the Error FW Index fields. See Table 14 – Firmware Image Types for specific rules for each firmware image type. The last firmware image will have its Next and Error indexes set to 0xFF.

6.2 Firmware Images

Firmware file contains one or more firmware images. Each firmware image is a block of bytes that needs to be uploaded to the device, using *StartUpload* and *SendChunk* commands. The image upload sequence is specified by the Firmware Image Descriptors (see Section 6.1.3).

Note: *DO NOT simply upload all firmware images. Be sure to follow the sequence, specified by the firmware image descriptors.*

7.0 Updating Firmware

Basic firmware update procedure is as follows:

1. Load the firmware file header into memory (see Appendix A: Firmware File Format).
2. Start the bootloader session (*Connect*).
    a. If the connection is successful, proceed to step 3.
    b. If the connection fails, use one of the following three methods to reset the device and enter the bootloader mode:
        i. Power cycle the device
        ii. Toggle the RESET pin
        iii. Issue an application firmware 'device reset' command. For OBDLink devices, send '??\r' to clear out the input buffer, then issue the 'ATZ' command. The proper flow of commands, if OBDLink is in **normal mode** and can be reset via UART:

            `Connect -> '??\r' -> 'ATZ\r' -> Delay 50 ms -> Connect`
3. Verify the bootloader version (*GetVersion*). Proceed only if the major version is 2, 3 or 4.
4. Get Device ID (*GetDevID*) and verify that it matches one of the Device IDs, specified in the firmware header. Otherwise, abort.
5. Upload firmware images contained in the firmware file, starting with image 0. For files containing multiple firmware images, follow the image descriptor linked list, according to the image type and the next image indexes. (see Table 14):
    a. Send the *StartUpload* command.
    b. Parse the response of the *StartUpload* command to determine the maximum chunk size (see Section 5). For typical applications, chunk size of 1024 bytes results in optimum balance of speed, error recovery, and progress granularity.
    c. Send the firmware image using the *SendChunk* command until all bytes of the firmware image have been transmitted.
6. With all the firmware images having been sent successfully, send the *GetFWStatus* command. A value of 0x00 for FW Status indicates firmware was not properly programmed, and requires a restart of the download process. Any other returned value indicates a successful transfer.
7. If the transfer was successful, issue a bootloader *Reset* command. The device will reboot and enter **normal mode**.

Appendix A: Firmware File Format

Firmware images to be used with STN Bootloader are released as binary files with a .bin extension. The STN firmware file format is detailed in this section. All multi-byte values are big-endian.

Table 12 - Firmware File Structure

	Field	Size (bytes)	Description
File Header	Signature	6	File signature (ASCII **STNFWv**)
	Version	2	File version (ASCII **05**)
	Device ID Count	1	Size of device IDs array
	Device IDs Array	[Dev ID Count] * 2	Device IDs of compatible devices that can be programmed with this firmware
	FW Image Descriptors Count	1	Size of firmware descriptors array. *If the count is zero, the array is empty, and the remainder of the file is a single firmware image.*
	FW Image Descriptors Array	[FW Image Desc Count] * 12	Optional firmware descriptors array (see Table 13)
	FW Images	Variable	One or more firmware images, specified by firmware image descriptors

Table 13 - Firmware Image Descriptor Format

Field	Length (bytes)	Description
Image Type	1	Firmware image type (see Table 14)
Reserved	1	
Next FW Index	1	Next firmware image to upload. 0xFF means there are no more images to upload.
Error FW Index	1	Firmware to upload on error. Used only for images of 'Validation' type.
Image Offset	4	Absolute file offset to beginning of firmware image
Image Size	4	Size of firmware image in bytes

Table 14 - Firmware Image Types

FW Image Type	Value (hex)	Description
Normal	00	Go to **[Next FW Index]** on success, handle all errors normally (ignore **[Error FW Index]**)
Normal, Tolerate Errors	01	Go to **[Next FW Index]** on success; if connection is lost, reconnect, then continue with the **[Next FW Index]**; if device comm error occurs, send *ResetDevice* command, reconnect, then continue with the **[Next FW Index]**; handle all other errors normally (ignore **[Error FW Index]**)
Validation	10	Go to **[Next FW Index]** on success, go to **[Error FW Index]** on Verification error, handle all other errors normally

Appendix B: CRC Sample Code

Table-based Algorithm

```
unsigned short crcTable[256] =
{
    0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50A5, 0x60C6, 0x70E7,
    0x8108, 0x9129, 0xA14A, 0xB16B, 0xC18C, 0xD1AD, 0xE1CE, 0xF1EF,
    0x1231, 0x0210, 0x3273, 0x2252, 0x52B5, 0x4294, 0x72F7, 0x62D6,
    0x9339, 0x8318, 0xB37B, 0xA35A, 0xD3BD, 0xC39C, 0xF3FF, 0xE3DE,
    0x2462, 0x3443, 0x0420, 0x1401, 0x64E6, 0x74C7, 0x44A4, 0x5485,
    0xA56A, 0xB54B, 0x8528, 0x9509, 0xE5EE, 0xF5CF, 0xC5AC, 0xD58D,
    0x3653, 0x2672, 0x1611, 0x0630, 0x76D7, 0x66F6, 0x5695, 0x46B4,
    0xB75B, 0xA77A, 0x9719, 0x8738, 0xF7DF, 0xE7FE, 0xD79D, 0xC7BC,
    0x48C4, 0x58E5, 0x6886, 0x78A7, 0x0840, 0x1861, 0x2802, 0x3823,
    0xC9CC, 0xD9ED, 0xE98E, 0xF9AF, 0x8948, 0x9969, 0xA90A, 0xB92B,
    0x5AF5, 0x4AD4, 0x7AB7, 0x6A96, 0x1A71, 0x0A50, 0x3A33, 0x2A12,
    0xDBFD, 0xCBDC, 0xFBBF, 0xEB9E, 0x9B79, 0x8B58, 0xBB3B, 0xAB1A,
    0x6CA6, 0x7C87, 0x4CE4, 0x5CC5, 0x2C22, 0x3C03, 0x0C60, 0x1C41,
    0xEDAE, 0xFD8F, 0xCDEC, 0xDDCD, 0xAD2A, 0xBD0B, 0x8D68, 0x9D49,
    0x7E97, 0x6EB6, 0x5ED5, 0x4EF4, 0x3E13, 0x2E32, 0x1E51, 0x0E70,
    0xFF9F, 0xEFBE, 0xDFDD, 0xCFFC, 0xBF1B, 0xAF3A, 0x9F59, 0x8F78,
    0x9188, 0x81A9, 0xB1CA, 0xA1EB, 0xD10C, 0xC12D, 0xF14E, 0xE16F,
    0x1080, 0x00A1, 0x30C2, 0x20E3, 0x5004, 0x4025, 0x7046, 0x6067,
    0x83B9, 0x9398, 0xA3FB, 0xB3DA, 0xC33D, 0xD31C, 0xE37F, 0xF35E,
    0x02B1, 0x1290, 0x22F3, 0x32D2, 0x4235, 0x5214, 0x6277, 0x7256,
    0xB5EA, 0xA5CB, 0x95A8, 0x8589, 0xF56E, 0xE54F, 0xD52C, 0xC50D,
    0x34E2, 0x24C3, 0x14A0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
    0xA7DB, 0xB7FA, 0x8799, 0x97B8, 0xE75F, 0xF77E, 0xC71D, 0xD73C,
    0x26D3, 0x36F2, 0x0691, 0x16B0, 0x6657, 0x7676, 0x4615, 0x5634,
    0xD94C, 0xC96D, 0xF90E, 0xE92F, 0x99C8, 0x89E9, 0xB98A, 0xA9AB,
    0x5844, 0x4865, 0x7806, 0x6827, 0x18C0, 0x08E1, 0x3882, 0x28A3,
    0xCB7D, 0xDB5C, 0xEB3F, 0xFB1E, 0x8BF9, 0x9BD8, 0xABBB, 0xBB9A,
    0x4A75, 0x5A54, 0x6A37, 0x7A16, 0x0AF1, 0x1AD0, 0x2AB3, 0x3A92,
    0xFD2E, 0xED0F, 0xDD6C, 0xCD4D, 0xBDAA, 0xAD8B, 0x9DE8, 0x8DC9,
    0x7C26, 0x6C07, 0x5C64, 0x4C45, 0x3CA2, 0x2C83, 0x1CE0, 0x0CC1,
    0xEF1F, 0xFF3E, 0xCF5D, 0xDF7C, 0xAF9B, 0xBFBA, 0x8FD9, 0x9FF8,
    0x6E17, 0x7E36, 0x4E55, 0x5E74, 0x2E93, 0x3EB2, 0x0ED1, 0x1EF0
};

unsigned short UpdateCcittCrc(unsigned short crc, unsigned char data)
{
    return (crc << 8) ^ crcTable[(crc >> 8) ^ data];
}

unsigned short CalculateCcittCrc(unsigned char* data, int len)
{
    unsigned short crc = 0;

    while (len-- > 0)
        crc = UpdateCcittCrc(crc, *data++);

    return crc;
}
```

Efficient Bytewise Algorithm

```
unsigned short UpdateCcittCrc(unsigned short crc, unsigned char data)
{
    unsigned short x;

    x = ((crc >> 8) ^ data) & 0xFF;
    x ^= x >> 4;
    crc = (crc << 8) ^ (x << 12) ^ (x << 5) ^ x;

    return crc;
}

unsigned short CalculateCcittCrc(unsigned char* data, int len)
{
    unsigned short crc = 0;

    while (len-- > 0)
        crc = UpdateCcittCrc(crc, *data++);

    return crc;
}
```

Appendix C: Device IDs

Table 15 lists device IDs for OBD Solutions devices.

Table 15 - Device IDs

Device ID (hex)	Device Name
1000	OBDLink CI
1100	OBDLink
1101	OBDLink S
1110	STN1110
1120	microOBD 200
1130	OBDLink SX
1150	OBDLink MX Rev 1.x
1151	OBDLink MX Rev 2.x
1152	OBDLink MX Wi-Fi
1155	OBDLink LX
1170	STN1170
1200	STSP200 - ECUsim CAN PIM
1300	STSP300 - ECUsim 5100 PIM
1310	STS2000 - ECUsim 2000
1320	STSP400 - ECUsim 5100 PIM
2100	STN2100
2120	STN2120
2230	OBDLink EX
2231	OBDLink EX
2232	OBDLink EX
2255	OBDLink MX+
2256	OBDLink MX+
2310	OBDLink CX
2320	Carbyte

Appendix D: Revision History

Revision D (September 10, 2025)

This revision includes typographical and formatting changes throughout the document text. Other changes are listed below.

- Updated Section 1.0, “Introduction”
  - Updated STN Firmware Updater image
  - Added reference to STN Bootloader version 4.x
- Updated Section 3.0, “Basic Operation”
  - Changed “STN11xx/2xxx family devices” to “OBDLink devices”
- Updated Section 4.6.2, “NACK Responses”
  - Changed error headings that didn’t match their table description
  - Corrected Validation error to Verification error
- Updated Section 5.2, “Device Information Commands”
  - Added GetDeviceNameEx command
  - Added details to table description of GetDeviceName command
- Updated Section 7.0, “Updating Firmware”
  - Updated application firmware reset procedure
  - Changed GetFwStatus usage to match command description
- Updated Appendix A: Firmware File Format
  - Corrected Validation error to Verification error
- Updated Appendix C: Device IDs
  - Added missing devices

Revision C (June 1, 2021)

This revision includes minor typographical and formatting changes throughout the document text. Other changes are listed below.

- Updated Section 1.0, “Introduction”
  - Updated STN Firmware Updater image
- Updated Section 3.0, “Basic Operation”
  - Updated STN device family number
- Updated Section 4.6.2, “NACK Responses”
  - Added firmware version error
- Updated Section 5.1, “Control Commands”
  - Added SetCommParams command
- Updated Section 5.2, “Device Information Commands”
  - Added GetMinFwVer command
- Updated Appendix C: Device IDs
  - Added missing devices
- Updated Appendix D: Revision History
  - Removed tables and changed order

Revision B (February 2, 2018)

This revision includes minor typographical and formatting changes throughout the document text. Other changes are listed below.

- Updated Section 1.0, “Introduction”
  - Added sample code availability
- Updated Section 5.3, “Firmware Upload Commands”
  - Corrected StartUpload command example
- Added Section 6.0, “Firmware File”
- Changed Section name “Firmware Update Procedure” to “Updating Firmware”
- Updated Section 7.0, “Updating Firmware”
  - Thoroughly updated the description of the update procedure.
- Appendix C: Device IDs
  - Added missing devices

Revision A (January 25, 2011)

Initial release of this document.

Appendix E: Contact Information

OBD Solutions LLC
11048 N 23rd Ave Ste 101
Phoenix, AZ 85029
United States

Phone: +1 623.582.2366
Email: sales@obdsol.com
Web: www.obdsol.com