Digital Matter Telematics
Device Integration
V2.6
7 March 2021

www.digitalmatter.com

1 DEVICES

This document relates to all cellular devices – full list in section 5.2.2

Note the G50 (deprecated) uses a different protocol, available from DM.
The LPWAN (LoRa, Sigfox, etc) and Iridium devices use a different protocol, available from DM.

2 ARCHITECTURE AND CONCEPTS

The Digital Matter Telematics (DMT) devices connect to the server via TCP/IP data connection. Communications with the server is done using TCP **Messages** as per this specification.

The devices record data in the flash memory on the device in the form of 'data **records**' and these records are transmitted up to the server as part of the TCP communications. There are a number of different data fields that can be recorded depending on the device, event and data associated with that event.

We refer to **TCP MESSAGES** for the communication protocol between the device and the server. Device data is stored in **RECORDS** which are uploaded to the server inside **TCP MESSAGES** – with a particular note that multiple records can be sent within a single TCP message.

On first connecting to the server the device sends a HELLO message which identifies the device and contains metadata about the device including Serial Number, SIM ICCID, IMEI and other information. The server replies with a HELLO Response.
Then the device typically sends a number of TCP messages which contain 1 or more data records. A number of these messages can be sent by the device before it will ask the server to confirm successful reception with a COMMIT REQUEST TCP message. The server should ensure that the data records have been saved and respond to the device with a COMMIT RESPONSE, after which the device can delete the records on the device.

Caution- Developers not used to writing TCP socket communications often fall into the trap of expecting the TCP stack to somehow deliver the TCP messages individually in the receive buffer. This is not the case! The TCP data is delivered as a stream and the developer needs to parse out the messages from the stream. Our protocol contains a fixed length header which contains the message type and payload length and this makes it easier to parse out the messages from the stream. Please discuss with Digital Matter if you would like more information on this.

3 DATATYPES

All data fields are LITTLE ENDIAN, meaning that the least significant BYTE of the data field is stored first in the flash memory or TCP data stream.

4 DATA RECORDS

All data records begin with an 11 byte header:

Record Length	UINT16
Sequence Number	UINT32
DateTime in seconds since 1/1/2013	UINT32
Log Reason	BYTE

The header is followed by a variable number of “Data Fields”.

4.1 DATA FIELDS

Refer to the “DMT Data Fields” document for the latest specification of the data fields.

4.2 Unknown Data Fields

The flexible data record allows new field types to be added. Any unknown data fields should be ignored by the data parser. This can be done without disruption to the data stream as the length of the unknown data field is contained in its Key length.

4.3 Field Lengths as Versioning

A number of the data fields are fixed length fields as they contain a fixed size structure. The developer should not assume the field length from the field ID as the field length for fixed size structures could be used in future for versioning, provided that the existing layout is maintained and any new data is added to the end of the structure.

4.4 DateTime

In order to fit the date and time into a 4 byte UINT32 it is represented as a count of the number of seconds from the base date and time of “1 January 2013 00:00:00”.
This can be easily converted in most systems with “DateAdd” and “DateDiff” functions.

4.5 Log Reasons

Log reasons are used to determine why the logging algorithm decided that a record should be recorded.

Please refer to the document “DMT Log Reasons” for an up-to-date description of the log reason values and meanings.

Additional log reasons can be added for new devices / events.

5 DMT TCP/IP Messages

The DMT devices connect to the server using a TCP/IP data connection and communication between the device and the server is done using TCP **messages** in the protocol structure defined here.
It is important to note that the TCP data is a stream and that multiple TCP messages may be appended in the stream to the server.

All data fields are Little Endian – LS Byte first.

5.1 TCP Message Format

All messages have the following format:

Offset	Length	Data Type	Description
0	2	BYTE[ ]	Sync characters = 0x02, 0x55
2	1	BYTE	Message type
3	2	UINT16	Payload length
5	n	Payload specific	Payload

5.2 Messages

5.2.1 Hello (0x00)

Every connection must begin with a Hello message which identifies the device.

Offset	Length	Data Type	Description
0	2	BYTE[ ]	Sync characters = 0x02, 0x55
2	1	BYTE	Message type
3	2	UINT16	Payload length
5	4	UINT32	Device Serial Number
9	16	ASCII String	Modem IMEI
25	21	ASCII String	SIM ICCID
46	1	BYTE	Product ID
47	1	BYTE	Hardware Revision
48	1	BYTE	Firmware Major
49	1	BYTE	Firmware Minor
50	4	UINT32	Flags – reserved

Example:

02 55 sync chars
00 message type
31 00 payload length = 49 bytes
AC 86 01 00 serial number = 100012
33 35 31 37 33 32 30 35 30 38 37 35 32 30 35 00 IMEI
38 39 35 32 34 36 30 30 30 30 30 30 30 30 35 34 30 39 30 33 00 SIM ICCID
11 product ID = 0x11 = G52 SOLAR
01 hardware revision
02 Firmware major
0E Firmware minor
00 00 00 00 Flags

5.2.2 Device Product IDs

The main device Product IDs:

Product ID (Hex)	Device
0x4B	Bolt
0x44	Dart2
0x4E	Eagle
0x4A	Falcon
0x43	G62 Cellular
0x4F	G120
0x4D	Oyster2 Cellular
0x3E	Remora2
0x49	Yabby GPS
0x48	Yabby WiFi Cellular

Other, no longer in production devices.

Product ID (Hex)	Device
0x22	Dart
0x1E	Flexi1
0x11	G52 Solar
0x17	G60
0x1C	G100
0x21	Remora
0x3A	Oyster Cellular (v1)
0x34	Sting

5.2.3 Hello Response (0x01)

The server must send this in response to the HELLO message.

Offset	Length	Data Type	Description
0	2	BYTE[]	Sync characters = 0x02, 0x55
2	1	BYTE	Message type
3	2	UINT16	Payload length
5	4	UINT32	UTC date/time from the server in seconds since 1/1/2013
9	4	UINT32	Flags (all reserved bits set to zero) **Bit 0** = "device rejected". Set this bit in the response to notify the device that it is not valid in the target system. Wait for the device to close the connection or ensure that you allow sufficient time for the response to be sent to the device before closing the socket connection. This will allow the device to back off rather than retrying to connect continuously. **Bit 1 =** "OEM config only session". Set this bit to schedule a config only session with OEM after the current session ends. **Bit 2** = "geo-fence update available". Set this bit in the response to notify the device that it should attempt to download geo-fence data. Not all devices support this. If it is not supported by the device this flag is simply ignored.

Example

02 55 sync chars

01 message type

08 00 payload length = 8 bytes

67 7C 37 02 server time in secs since 1 Jan 2013

00 00 00 00 flags

5.2.4 Data Record Upload (0x04)

TCP message sent to the server to upload data records.

This message can contain **multiple** records.

In order to parse the records the server will look at the record header and extract the record size.

Offset	Length	Data Type	Description
0	2	BYTE[]	Sync characters = 0x02, 0x55
2	1	BYTE	Message type
3	2	UINT16	Payload length = N
5	N	BYTE[]	Record data

Example – A sample G52 Solar Heartbeat message

02 55

04 (message type Data record upload)

3D 00 (payload length = 61)

(payload of records)

3D 00 (record1 length = 61, in this case it is the entire payload so only this record)

47 46 00 00 (seqn number)

96 D6 84 02 (RTC datetime)

0B (log reason == heartbeat)

00 15 (FID=0 GPS Data, Len=21 bytes)

02 D4 84 02 F0 43 F4 EC 2A 69 09 45 2B 00 1F 00 05 00 11 23 03

02 08 (FID=2 Digital Data, len = 8) (DI = 0, DO = 0, Status = 0x0A = b1010 = not in trip, Vbat OK, Vext not OK, Connected to GSM)

00 00 00 00 00 00 0A 00

06 0F (FID=6 Analogue Data, len = 15) (analogue number + INT16 pairs)

04 1D 00 01 FE 0F 02 1E 00 05 00 00 03 BF 08

(end of record)

(end of message)

5.2.5 Commit Request (0x05)

This message is sent to request the server to commit the uploaded data records to the database.

It provides for a mechanism of safe delivery of the data to the server and records will only be deleted off the device once the successful commit response is received.

Offset	Length	Data Type	Description
0	2	BYTE[ ]	Sync characters = 0x02, 0x55
2	1	BYTE	Message type
3	2	UINT16	Payload length = 0

Example

02 55 sync chars

05 message type (commit request)

00 00 payload length = 0

5.2.6 Commit Response (0x06)

When the server has received a commit request message and it subsequently committed all uploaded data records to its database, it must respond with this message to let the device know that the records had been committed. If the device does not receive a commit response then the last 'batch' of records will be re-sent to the server when the device next connects. The server can use the sequence number and/or the date time in the records to detect duplicate records.

Offset	Length	Data Type	Description
0	2	BYTE[ ]	Sync characters = 0x02, 0x55
2	1	BYTE	Message type
3	2	UINT16	Payload length = 1
5	1	BYTE	Response Flags Bit 0 = result (0 = Commit Error, 1 = Commit OK) All other bits must be zero

Example

02 55 sync chars
06 message type (commit response)
01 00 payload length = 1
01 1 = commit OK

5.2.7 Other TCP Messages

The device specifications and the Digital Matter OEM server have several other messages for a variety of purposes. These should be ignored by any 3rd party software, except for 2 messages that need a canned response.

Devices are designed to primarily work with the OEM Server. If the device is pointed directly to a 3rd party software platform, these 2 messages need responses for the device to continue with normal operation. Implementers of the TCP protocol should include these 2 canned responses:

5.2.7.1 Message Type 0x14: Canned response 1

Multiple messages with ID = 0x14 may be received. Respond to each with the canned response below.

Request:

Offset	Length	Data Type	Description
0	2	BYTE[ ]	Sync characters = 0x02, 0x55
2	1	BYTE	Message type = 0x14
3	2	UINT16	Payload length is variable = X
5	X	BYTE[X]	Payload with variable contents

Canned Response:

Offset	Length	Data Type	Description
0	2	BYTE[ ]	Sync characters = 0x02, 0x55
2	1	BYTE	Message type = 0x15
3	2	UINT16	Payload length = 0x0C or [0x0C, 0x00]
5	12	BYTE[12]	Body = [0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00]

The full response, including the sync chars, message type and length will look like this:

```
0x02, 0x55,       Sync Chars
0x15,             Message Type
0x0C, 0x00,       Length of 12
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00              Canned response
```

5.2.7.2 Message Type 0x22: Canned response 2

Multiple messages with ID = 0x22 may be received. Respond to each with the canned response below.

Request:

Offset	Length	Data Type	Description
0	2	BYTE[ ]	Sync characters = 0x02, 0x55
2	1	BYTE	Message type = 0x22
3	2	UINT16	Payload length = [0x00, 0x00]

Canned Response:

Offset	Length	Data Type	Description
0	2	BYTE[ ]	Sync characters = 0x02, 0x55
2	1	BYTE	Message type = 0x23
3	2	UINT16	Payload length = [0x00, 0x00]

The full response, including the sync chars, message type and length will look like this:

```
0x02, 0x55,       Sync Chars
0x23,             Message Type
0x00, 0x00        Length of zero
```

5.3 Connection Closure

The device will, under normal operating conditions, close the server TCP connection once all data has been uploaded and the device determines that it has no more actions to perform. The device may keep the server connection open during live tracking or depending on its operational mode (for example in ‘recovery mode’ the G50 will attempt to always maintain a connection with the server).