OpenDMTP Protocol Definition Reference Manual

Manual Revision HIstory			
Rev	**Date**	**Changed**	**Author**
0.0.1	2006/01/22	Initial Release	MDF
0.0.2	2006/05/07	Update new property types	MDF
0.1.0	2006/05/31	Increased UniqueIDs (0xE011) payload to 20 bytes. Added definition for Server File Upload packet. Added Geofence admin command property (0xF542). Relocated Alarmed Geofence properties. Added GPS latitude/longitude encoding algorithms	MDF
0.1.1	2006/07/10	Correction in Appendix C: "Standard GPS packet (high resolution)" packet type incorrectly stated as E030. The packet type should be E031.	MDF
0.2.0	2007/01/25	Added several new status codes. Added several new properties. Added an optional "maximum number of events" field to the server packets $E000 and $E001.. Moved property PROP_CFG_GPS_MODEL from 0xEF22 to 0xEF2A. Moved property PROP_COMM_APN_SETTINGS from 0xF3AA to 0xF3AC. Moved property PROP_GEOC_ACTIVE_ID from 0xF571 to 0xF567. In order to accomodate a higher top-end limit of 4.29M km, (the previous limit was 429K km), the unit of measurement used to accumulate internal odometer values has changed from 0.1 meter to 1 meter units. The following properties have been changed accordingly: PROP_GPS_ACCURACY, PROP_GPS_DISTANCE_DELTA, PROP_ODOMETER_#_VALUE, PROP_ODOMETER_#_LIMIT. In order to provide easier porting to other platforms, the unit of measurement for elapsed timer limits and values has been changed from milliseconds to seconds. The following properties have been changed accordingly: PROP_ELAPSED_#_LIMIT, PROP_ELAPSED_#_VALUE.. Added a paragraph about data types that can be placed in custom event packets.	MDF
0.2.1	2007/03/18	Specified minimum and maximum suggested size for the PROP_STATE_UNIQUE_ID property. Added property PROP_STATE_DEVICE_BT.	MDF
0.2.2	2007/04/15	Added status code STATUS_POWER_FAILURE, STATUS_STATE_ENTER, STATUS_STATE_EXIT.	MDF
0.2.3	2008/01/10	Added status codes STATUS_WAYMARK_0/1/2 (was STATUS_WAYMARK). Added status code STATUS_EXCESS_BRAKING and PROP_MAX_BRAKE_G_FORCE property. Added property support for serial port #4. Added properties PROP_DEV_AUTO_RESET, PROP_STATE_OSVERSION, PROP_STATE_IS_IN_MOTION, PROP_COMM_UPLOAD_PORT,. Added out-of-band 'Get'/'Put' request packet types to server/client file transfer packet.	MDF
0.2.4	2008/04/04	Added property PROP_COMM_FAILURE_DELAY.	MDF

OpenDMTP

Contents:

Appendices:

1) Introduction

This manual outlines the packets, status codes, and defined properties, needed to implement the OpenDMTP protocol. This manual currently does not contain the definitions for the reserved server error codes, client error codes, or custom event field types (needed for custom event packet negotiation). For details concerning these areas, the reference implementation header files must be examined and can be found in the OpenDMTP C Client package.

The following header files should be considered part of this document:

base/events.h	- Custom event packet field types
base/serrors.h	- Reserved server error codes
base/cerrors.h	- Reserved client error codes
base/cdiags.h	- Reserved client diagnostic codes
base/statcode.h	- Reserved status codes
base/packet.h	- Client/Server packet definitions
base/cmderrs.h	- Reserved client command errors
base/props.h	- Reserved client property deifnitions.

These header files are available in the OpenDMTP 'C' reference implementation package.

2) Packets/Encoding

DMTP supports both binary and ASCII encoded packets. Depending on the transport media type in use, one form of encoding may be more preferable than another. For instance, binary encoded packets may be best for a straight socket based connection, while ASCII encoded packets may be better for communication over a serial port, Bluetooth, or wireless modem.

2.1) Binary encoded packets

Binary encoded packets will have the following general format:

Byte:Length	***Description***
0:1	Packet header (0xE0 for standard binary encoded DMTP packets)
1:1	Packet type
2:1	Payload length (0 to 255 bytes)
3:X	X bytes of payload

For example, a hex representation of an Account-ID packet would look like this:

`0xE012084F70656E444D5450`

Where:

- 'E0' is the packet header
- '12' is the packet type (Account ID)
- '08' is the payload length (8 bytes)
- '4F70656E444D5450' is the payload itself ("OpenDMTP")

2.2) ASCII encoded packets

ASCII encoded packets can be encoded in 2 forms, Hex or Base64. While both encodings will work fine when transmitting over an ASCII media the increased size of the packet must be considered. Using a Base64 encoded payload will add about 33% to its size, while encoding in Hex will double the size of the payload (100% increase in size).

ASCII encoded packets will have the following general format:

Characters	***Description***
"$"	Start of ASCII encoded packet [Hex 0x24]
"E0"	Packet header (hex "E0" for standard DMTP)
"XX"	Packet type specified in ASCII hex characters
":" or "="	Payload encoding type (":" for Hex, "=" for Base64)
"XX..XX"	Followed by the Hex or Base64 encoded payload
"\r"	Terminated with a single carriage return character.

For example, a Base64 encoded Account-ID packet would look like this:

`"$E012=T3BlbkRNVFAA\r"`

Where:

- "$" is the start of ASCII packet
- "E0" is the packet header
- "12" is the packet type (Account ID)
- "=" is the Base64 encoding indicator
- "T3BlbkRNVFAA" is the Base64 encoded payload ("OpenDMTP")
- "\r" is the terminating carriage-return.

ASCII encoded packets can optionally include a trailing checksum that will be validated by the server. The trailing checksum can be included before the terminating carriage return and has the following form:

    "*XX"

Where:

- "*" is an indication of the beginning of the checksum
- "XX" is the hex representation of the checksum.

For example, the above Account-ID packet, with the included checksum, would look like this:

    "$E012=T3BlbkRNVFAA*07"

Where:

- "$E012=T3BlbkRNVFAA" is the Base64 encoded packet
- "*" is the indication of the beginning of the checksum
- "07" is the checksum

The checksum is calculated as the XOR sum of all bytes in the string following the prefixing character '$' (exclusive), and up-to but not including the '*' character. Here is an example 'C' function that will return the checksum value. This function can be used for creating or testing a checksum value.

```
#define CHECKSUM_SEPARATOR '*'
unsigned char calcChecksum(const unsigned char *d) {
    // 'd' points to the first character after the prefixing '$' character
    unsigned char cksum = *d++;
    for (; *d && (*d != CHECKSUM_SEPARATOR); d++) {
        cksum ^= *d;
    }
    return cksum;
}
```

If a checksum is specified and is found to be invalid, the server will return an error.

2.3) Server packet encoding

When possible, the packet encoding used by the server should mirror that of the client during a communication session. That is, if the first packet that the client sends to the server is binary encoded, then the server should respond with all binary encoded packets. Or, if the first packet sent by the client is ASCII encoded, then the server should send all ASCII encoded packets.

3) Transport Media

The "Transport Media" type is the method used for transmitting data from the client to a remote server, or some form of axillary storage. The reference implementation includes transport media support for connections over sockets, connections over a GPRS modem, and simple storage of events in a local file for later retrieval. However, many other types of transport media are also possible, such as communication over a standard serial port, BlueTooth, or other wireless media.

3.1) Simplex/Duplex communication

Depending on the requirements of the transport media, OpenDMTP supports both bidirectional (duplex) and one-way (simplex) communication. Duplex connections allow the client to communicate directly with the server and receive commands and reconfiguration parameters from the server when necessary. When sending data via a simplex communication, the client is unable to receive commands or reconfiguration parameters from the server, or even an acknowledgment that the transmission was received. Socket communications is an example that can support both duplex (TCP) and simplex (UDP) communication types, and the client is free to choose which method to use when sending different kinds of messages. Some data transports involving satellite communications might be one example of a client which can only support simplex communications, and in this case can never receive commands or reconfiguration parameters from the server.

If a client supports both Duplex and Simplex type communication, it must be aware of the advantages and disadvantages of both forms of communication. When sending events via Duplex communication, the client receives an acknowledgment from the server that the data was actually received, however there is more overhead in a duplex type connection which may end up costing more in a wireless environment (e.g. GPRS). When sending events via Simplex communication, the overhead is much less, thus the cost will be much less, however the delivery of the message is not guaranteed and the client does not receive any form of acknowledgment from the server that the events were actually delivered (note: openning an in-bound UDP/simplex service is typically way too much overhead for a mobile client to handle, however, this would be a limitation of the specific platform and environment, and not a restriction in the protocol).

3.2) Duplex conversation initiator

Communication is usually initiated by the client, however, the transport media type usually determines who starts the conversation (ie. sends the first packet). For instance, in a periodically connected transport media, such as socket based communication, the client should first identify itself to the server and transmit whatever information (events, etc) that it has ready to send. In a sporadically connected transport media (where a connection is typically maintained until physically broken), such as a serial port or wireless BlueTooth communication, it may be the server which is expected to initiate the conversation.

4) Client Properties

The client 'properties' provides a means for configuring nearly all aspects of the clients behavior. This includes how it is to connect to the server, and how it is to analyze its various GPS rules (motion, distance, etc).

The client may specify that given property is read-only or write-only. This means that the remote server may only read from a specific property, or write to a specific property.An example of a read-only property might be the device serial number. The client may not want the server to be able to change this value (it may be based on client hardware register). An example of a write-only property would be something like a client based command, such as causing the client to unlock a car door (if the client is so equiped). Other properties may be both readable and writable, such as a property specifying the number of second between periodic in-motion messages, etc.

The DMTP protocol reserves the property id 0x0000 and all property ids in the range 0xD000 to 0xFFFF. Custom implementations of the protocol are free to use any other available property id.

See Appendix 'A' below for a complete list of standard client properties.

5) Client Generated Events

Events will be generated in the client based on some set of rules, then queued for transmission to the server. DMTP provides for a wide range of 'status-codes' that can mark an event with a reason why that event was generated. For instance, a motion rule which is triggered when the speed of the vehicle exceeds some set limit could cause an 'Excess Speed' event to be generated.

See Appendix 'B' below for a complete list of the standard status codes.

6) Custom Event Packet Negotiation

Events are generated on the client and are queued for transmission to the server. The types of data that the client deems pertinent and wishes to include in an event can be quite varied. These include data types such as time, latitude/longitude, speed, heading, distance travelled, etc. In a low-bandwidth environment it is imperative that the client transmit only that data which is deemed important. To avoid having to create multiple packet types that include individual data types, or include every possible combinations of data types that might be needed, DMTP supports a feature called "custom event packet negotiation". This means that a client can construct a single packet with only that imformation if feels is necessary, and send it to the server. If the server does not understand the packet format, it can ask the client for a "custom event template". After receiving this 'template' from the client, the server knows the format of the custom packet and can continue parsing the client data.

The available custom event fields support numeric (integers), ASCII string, and binary types. All numeric integer data must be presented in network-byte-order, or big-endian, format and may be 1 to 4 bytes in length. String fields may contain ASCII data that is less than, or equal to the maximum length of the field. If the ASCII data is less than the length of the field, then it must be terminated with a single null byte (ie. hex 0x00). The remaining packet fields would then continue immediately following this null byte. Binary data must fill the entire fixed length of the field, and may be between 1 and 255 bytes in lengh. The combined length of all defined data fields must not exceed 255 bytes.

It it possible that a server may not have implemented the custom event packet negotiation feature and thus is unable to solicit the appropriate 'template' from the client. In this case the server will return an error to the client indicating that it does not support this feature and the client will have to act accordingly (either fall back to a standard known packet format, or simply terminate the connection).

See header file “base/events.h” for a detailed list of possible custom event field types.

7) Client Connection Configuration

Some Internet based DMTP service providers may impose some connection and communication limits on the client, based on the provided level of service. These limits may include how many duplex or simplex connections are allowed per hour, or how many events may be sent per hour.

The DMTP protocol provides for configurable properties that can assist the client in adhering to these restrictions.

See the "Communication Properties" section in Appendix 'A' below for a list of configurable connection properties.

8) Client to Server Packets

Here is a summary of the standard client to server packet types (the prefixing hex 'E0' represents the client packet header and is standard on all DMTP client to server packets):

E000 - End of block/transmission, "no more to say"
E001 - End of block/transmission, "I have more to say"
E011 - Unique identifier
E012 - Account identifier
E013 - Device identifier
E030 - Standard-resolution GPS packet definition
E031 - High-resolution GPS packet definition
E05X - Pre-defined DMT service provider packet definitions [50-5F]
E07X - Client defined custom data packet definitions [70..7F]
E0B0 - Return property value
E0CF - Custom packet definition 'template'
E0D0 - Diagnostic codes
E0E0 - Error codes (see header file “base/cerrors.h” for a list of possible client error codes)

See Appendix 'C' below for a complete description and definition of the individual client packet types.

9) Server to Client Packets

Here is a summary of the standard server to client packet types (the prefixing hex 'E0' represents the server packet header and is standard on all DMTP server to client packets):

E000 - End of block, "Only speak when spoken to. Speak now"
E001 - End of block, "You may speak freely"
E0A0 - Acknowledge received events
E0B0 - Get property value
E0B1 - Set property value
E0C0 - Server file upload
E0E0 - Error codes (see header file “base/serrors.h” for a list of possible server error codes)
E0FF - End transmission (connection will be closed)

See Appendix 'D' below for a complete description and definition of the individual server packet types.

Appendix 'A' - Standard Client Properties

Client property definitions can also be found in the header file “base/props.h”.

A.1) Device configuration properties

Device Auto-Reset Interval:	
Property Key	**0xEE01** PROP_DEV_AUTO_RESET ("dev.autoreset")
Description	Preferred device automatic "reset" interval (in seconds)
Attributes	UInt32, Optional
Value Byte:Len	***Value Field Description***
0:4	32 bit value representing the preferred reset interval in seconds

Notes:

– This value can be considered a 'hint' to the device regarding the preferred reset interval. The device may choose to delay a reset if a critical task is currently being performed.

A.2) Transport media port configuration properties

Transport media serial port name:	
Property Key	**0xEF11** PROP_CFG_XPORT_PORT ("cfg.xpo.port")
Description	Serial port name to which the transport media device is attached
Attributes	ASCIIZ, Read-Only, Optional
Value Byte:Len	***Value Field Description***
0:X	ASCIIZ String representing the serial port name

Notes:

- If used, this value represents the serial port to which the transport media device is attached. For instance, if the system is configured for GPRS data transport, then this value may represent the serial port to which the GPRS modem is attached.
- The read-only attribute of this property should be enforced by the client. to prevent the server from accidentally changing this value.

Transport media serial port speed (BPS):	
Property Key	**0xEF12** PROP_CFG_XPORT_BPS ("cfg.xpo.bps")
Description	Transport media device serial port speed, in bits-per-second.
Attributes	UInt32, Read-Only, Optional
Value Byte:Len	***Value Field Description***
0:4	32 bit value representing the serial port speed, in BPS

Notes:

- If used, this value represents the speed of the serial port to which the transport media device is attached.
- The read-only attribute of this property should be enforced by the client. to prevent the server from accidentally changing this value.

Transport media port debug	
Property Key	**0xEF1D** PROP_CFG_XPORT_DEBUG ("cfg.xpo.debug")
Description	For use when debugging the transport media device
Attributes	Boolean, Read-Only, Optional
Value Byte:Len	***Value Field Description***
0:1	Non-zero if debugging this transport media device

Notes:

- Used only when debugging this transport media device.
- The read-only attribute of this property should be enforced by the client. to prevent the server from accidentally changing this value.

A.3) GPS port configuration

GPS serial port name:	
Property Key	**0xEF21** PROP_CFG_GPS_PORT ("cfg.gps.port")
Description	Serial port name to which the GPS device is attached
Attributes	ASCIIZ, Read-Only, Optional
Value Byte:Len	***Value Field Description***
0:X	ASCIIZ String representing the serial port name

Notes:

- If used, this value represents the serial port to which the GPS device is attached.
- The read-only attribute of this property should be enforced by the client. to prevent the server from accidentally changing this value.

GPS serial port speed (BPS):	
Property Key	**0xEF22** PROP_CFG_GPS_BPS ("cfg.gps.bps")
Description	GPS device serial port speed, in bits-per-second.
Attributes	UInt32, Read-Only, Optional
Value Byte:Len	***Value Field Description***
0:4	32 bit value representing the serial port speed, in BPS

Notes:

- If used, this value represents the speed of the serial port to which the GPS device is attached.
- The read-only attribute of this property should be enforced by the client. to prevent the server from accidentally changing this value.

GPS model name/type:	
Property Key	**0xEF2A** PROP_CFG_GPS_MODEL ("cfg.gps.model")
Description	The model name/type of the attached GPS device
Attributes	ASCIIZ, Read-Only, Optional
Value Byte:Len	***Value Field Description***
0:X	ASCIIZ String representing the name/type of the attached GPS device.

Notes:

- This value may be used for custom GPS device initialization.
- The read-only attribute of this property should be enforced by the client. to prevent the server from accidentally changing this value.

GPS port debug	
Property Key	**0xEF2D** PROP_CFG_GPS_DEBUG ("cfg.gps.debug")
Description	For use when debugging the GPS device
Attributes	Boolean, Read-Only, Optional
Value Byte:Len	***Value Field Description***
0:1	Non-zero if debugging this GPS device

Notes:

- Used only when debugging this GPS device.

A.4) General Serial port configuration

Serial port 0 name	
Property Key	**0xEF31** PROP_CFG_SERIAL0_PORT ("cfg.sp0.port") **0xEF41** PROP_CFG_SERIAL1_PORT ("cfg.sp1.port") **0xEF51** PROP_CFG_SERIAL2_PORT ("cfg.sp2.port") **0xEF61** PROP_CFG_SERIAL3_PORT ("cfg.sp3.port") **0xEF71** PROP_CFG_SERIAL4_PORT ("cfg.sp4.port")
Description	General port name for serial ports #0,1,2,3,4
Attributes	ASCIIZ, Read-Only, Optional
Value Byte:Len	***Value Field Description***
0:X	ASCIIZ String representing this serial port name

Notes:

- The read-only attribute of this property should be enforced by the client. to prevent the server from accidentally changing this value.

Serial port 0 speed (BPS):	
Property Key	**0xEF32** PROP_CFG_SERIAL0_BPS ("cfg.sp0.bps") **0xEF42** PROP_CFG_SERIAL1_BPS ("cfg.sp1.bps") **0xEF52** PROP_CFG_SERIAL2_BPS ("cfg.sp2.bps") **0xEF62** PROP_CFG_SERIAL3_BPS ("cfg.sp3.bps") **0xEF72** PROP_CFG_SERIAL4_BPS ("cfg.sp4.bps")
Description	General serial device port speed, in bits-per-second.
Attributes	UInt32, Read-Only, Optional
Value Byte:Len	***Value Field Description***
0:4	32 bit value representing the serial port speed, in BPS

Notes:

- If used, this value represents the speed of general serial ports #0/1/2/3/4 respectively.
- The read-only attribute of this property should be enforced by the client. to prevent the server from accidentally changing this value.

Serial port 0 debug	
Property Key	**0xEF3D** PROP_CFG_SERIAL0_DEBUG ("cfg.sp0.debug") **0xEF4D** PROP_CFG_SERIAL1_DEBUG ("cfg.sp1.debug") **0xEF5D** PROP_CFG_SERIAL2_DEBUG ("cfg.sp2.debug") **0xEF6D** PROP_CFG_SERIAL3_DEBUG ("cfg.sp3.debug") **0xEF7D** PROP_CFG_SERIAL4_DEBUG ("cfg.sp4.debug")
Description	For use when debugging serial ports #0/1/2/3/4
Attributes	Boolean, Read-Only, Optional
Value Byte:Len	***Value Field Description***
0:1	Non-zero if debugging this serial port

Notes:

- Used only when debugging this serial port

A.5) Command properties

Save properties to auxillary storage	
Property Key	**0xF000** PROP_CMD_SAVE_PROPS ("cmd.saveprops")
Description	Force property 'save'
Attributes	Write-Only, Optional
Value Byte:Len	***Value Field Description***
n/a	No arguments

Notes:

- This indicates to the client that any changed properties should be saved to auxillary storage.

Authorize user/driver	
Property Key	**0xF002** PROP_CMD_AUTHORIZE ("cmd.auth")
Description	Set device authorization
Attributes	Write-Only, Optional
Value Byte:Len	***Value Field Description***
0:X	User name
X:X	Password (optional)

Notes:

- Per client requirements, this command authenticates the specified user and then sets the PROP_STATE_USER_ID property to the name of the user.

Generate status event	
Property Key	**0xF011** PROP_CMD_STATUS_EVENT ("cmd.status")
Description	Generate/Send status event
Attributes	Write-Only, Optional
Value Byte:Len	***Value Field Description***
0:2	Status code of event to generate
2:1	Index, as needed by client [optional]

Notes:

- The client may decide which status codes are supported in this command. However, at least STATUS_LOCATION should be supported to allow querying the device about it's current location.

<u>Set output</u>	
Property Key	**0xF031** PROP_CMD_SET_OUTPUT ("cmd.output")
Description	Set digital output
Attributes	Write-Only, Optional
Value Byte:Len	***Value Field Description***
0:1	Index of digital output
1:1	Output state (0x00 = off, 0x01 = on)
2:4	Duration in milliseconds to remain in specified state [optional]

Notes:

- Support for this command on the client is optional (as in the case where the client does not support digital outputs). If not supported by the client it should return an error indicating that this command is not supported.

<u>Set output</u>	
Property Key	**0xF0FF** PROP_CMD_RESET ("cmd.reset")
Description	Reset/Reboot client
Attributes	Write-Only, Optional
Value Byte:Len	***Value Field Description***
0:1	Reset type (0=cold reset, 1=warm reset, other values defined by client)
1:X	Client defined reset authorization [optional]

Notes:

- Support for this command on the client is optional.

A.6) State/Version properties

Protocol version	
Property Key	**0xF100** PROP_STATE_PROTOCOL ("sta.proto")
Description	Protocol version
Attributes	UInt8, Array, Read-Only
Value Byte:Len	***Value Field Description***
0:1	Major version ID
1:1	Minor version ID
2:1	Minor revision ID [optional]

Notes:

- This value represents the version of the DMTP protocol that this client has implemented.
- This value should change only as core features of the protocol itself changes.
- The read-only attribute of this property should be enforced by the client..

Firmware version	
Property Key	**0xF101** PROP_STATE_FIRMWARE ("sta.firm")
Description	Firmware version
Attributes	ASCIIZ, Read-Only
Value Byte:Len	***Value Field Description***
0:X	ASCIIZ String representing the firmware version id

Notes:

- This value is defined by the client and represents the version of the firmware.

OS version	
Property Key	**0xF104** PROP_STATE_OSVERSION ("sta.osvers")
Description	OS version
Attributes	ASCIIZ, Read-Only
Value Byte:Len	***Value Field Description***
0:X	ASCIIZ String representing the OS version id

Notes:

- This value is defined by the client and represents the version of the OS.

Copyright notice	
Property Key	**0xF107** PROP_STATE_COPYRIGHT ("sta.copyright")
Description	Copyright notice
Attributes	ASCIIZ, Read-Only
Value Byte:Len	***Value Field Description***
0:X	ASCIIZ Copyright notice

Note:

- This value is defined by the client.

Device serial number	
Property Key	**0xF110** PROP_STATE_SERIAL ("sta.serial")
Description	Client device serial number
Attributes	ASCIIZ, Read-Only
Value Byte:Len	***Value Field Description***
0:X	ASCIIZ String representing the device serial number

Notes:

- This value is defined by the client and may be the same as the Device-ID.

Device Unique ID	
Property Key	**0xF112** PROP_STATE_UNIQUE_ID ("sta.uniq")
Description	Unique ID
Attributes	Binary, Read-Only, Optional
Value Byte:Len	***Value Field Description***
0:X	Unique code provided by the DMT service provider

Notes:

- This unique id is provided by the DMT service provider and uniquely identifies the device. This value should be at least 4 bytes, but not more than 20 bytes.

Account ID	
Property Key	**0xF114** PROP_STATE_ACCOUNT_ID ("sta.account")
Description	Account ID
Attributes	ASCIIZ, Read-Only
Value Byte:Len	***Value Field Description***
0:X	Device Account ID

Notes:

- This account id is provided by the DMT service provider and uniquely identifies the owner of the account.
- The Account ID must include only the characters from the set A to Z, 0 to 9, underscore, and hyphen.
- The maximum length of an Account ID is 20 characters.

Device ID	
Property Key	**0xF115** PROP_STATE_DEVICE_ID ("sta.device")
Description	Device ID
Attributes	ASCIIZ, Read-Only
Value Byte:Len	***Value Field Description***
0:X	Device ID

Notes:

- This device id is provided by account owner and is registered to the DMT service provider. This device id uniquely identifies the device within the account id.
- The Device ID must include only the characters from the set A to Z, 0 to 9, underscore, and hyphen.
- The maximum length of an Device ID is 20 characters.

Device ID	
Property Key	**0xF116** PROP_STATE_DEVICE_BT ("sta.device.bt")
Description	Device ID used for secondary transport media name (ie. Serial/Bluetooth)
Attributes	ASCIIZ, Read-Only, Optional
Value Byte:Len	***Value Field Description***
0:X	Device ID

Notes:

- This device id is used when there is more than one transport media (such as GPRS and Bluetooth) and the device needs to identify itself with a separate ID over each transport..
- The Device ID must include only the characters from the set A to Z, 0 to 9, underscore, and hyphen.
- The maximum length of an Device ID is 20 characters.

User ID	
Property Key	**0xF117** PROP_STATE_USER_ID ("sta.user")
Description	User ID
Attributes	ASCIIZ, Read-Only
Value Byte:Len	***Value Field Description***
0:X	User ID

Notes:

- This user id is provided by account owner and is registered to the DMT service provider. This user id may be used as-needed by the device to send as field information in an event.
- The User ID must include only the characters from the set A to Z, 0 to 9, underscore, and hyphen.
- The maximum length of an User ID is 20 characters.

User ID time	
Property Key	**0xF118** PROP_STATE_USER_TIME ("sta.user.time")
Description	Time User ID was specified
Attributes	UInt32, Read-Only, Optional
Value Byte:Len	***Value Field Description***
0:4	Number of seconds since midnight Jan 1, 1970 GMT

Notes:

- This timestamp should be initialized automatically at the time the PROP_STATE_USER_ID value was set. This value is typically used to provide the length of time that a given user has been attached to this device.

Current device time	
Property Key	**0xF121** PROP_STATE_TIME ("sta.time")
Description	Current time of device
Attributes	UInt32, Read-Only, Optional
Value Byte:Len	***Value Field Description***
0:4	Number of seconds since midnight Jan 1, 1970 GMT

Notes:

- Implementation of this property is optional. However, if the client cannot, or does not wish to support this feature, it should at least return a 0-length value.

Current/Latest device GPS fix	
Property Key	**0xF123** PROP_STATE_GPS ("sta.gpsloc")
Description	Latest (current) GPS fix
Attributes	GPS, Read-Only, Optional
Value Byte:Len	***Value Field Description***
0:4	GPS fix time
4:3 4:4	Standard-resolution encoded latitude, or High-resolution encoded latitude
7:3 8:4	Standard-resolution encoded longitude, or High-resolution encoded longitude
10:4 12:4	[optional] (Standard-resolution) odometer meters, or [optional] (High-resolution) odometer meters

Notes:

- Depending on the degree of accuracy that the client wishes to provide, the client may return either a 6-byte (standard resolution), or 8-byte (high resolution), encoded Lat/Lon. The server will infer from the length of the payload how the Lat/Lon was encoded. (See Appendix E for latitude/longitude encoding details).
- If the client cannot, or does not wish to support this feature, it must at least return a 0-length value.

GPS device diagnostics	
Property Key	**0xF124** PROP_STATE_GPS_DIAGNOSTIC ("sta.gpsdiag")
Description	Latest (current) GPS diagnostic information
Attributes	Uint32, Array, Read-Only, Optional
Value Byte:Len	***Value Field Description***
0:4	Last GPS sample time (time of last sample attempt)
4:4	Last GPS valid fix time (time of last valid fix)
8:4	Number of valid GPS fixes since reboot
12:4	Number of invalid GPS fixes since reboot
16:4	Number of forced GPS restarts

Notes:

- This property is used by the client to provide diagnostic information regarding the current health of the GPS module.

Is Vehicle in Motion	
Property Key	**0xF127** PROP_STATE_IS_IN_MOTION ("sta.inmotion")
Description	Non-zero if vehicle is currently in motion.
Attributes	Boolean
Value Byte:Len	***Value Field Description***
0:1	Non-zero if client is currently in motion.

Queued events	
Property Key	**0xF131** PROP_STATE_QUEUED_EVENTS ("sta.evtqueue")
Description	Event counts (number of queued/unacknowledged events, and total events generated)
Attributes	UInt32, Read-Only, Optional
Value Byte:Len	***Value Field Description***
0:4	Number of queued, unacknowledged, events
4:4	Total number of events generated (either since last reboot, or since installed)

Client device diagnostics	
Property Key	**0xF141** PROP_STATE_DEV_DIAGNOSTIC ("sta.devdiag")
Description	Latest (current) Client device diagnostic information
Attributes	Uint32, Array, Read-Only, Optional
Value Byte:Len	***Value Field Description***
0:4	Current device reset count (number of times the device has been reset)
4:4	Current supply voltage (in millivolts)
8:4	reserved
12:4	reserved
16:4	reserved

Notes:

- This property is used by the client to provide diagnostic information regarding the current health of the device.

A.7) Data transmission/communication properties

Client speaks first	
Property Key	**0xF303** PROP_COMM_SPEAK_FIRST ("com.first")
Description	Indicates whether, or not, the client is to initiate conversation on connect.
Attributes	Boolean
Value Byte:Len	***Value Field Description***
0:1	Non-zero if client is expected to initiate conversation

Notes:

- This value may be read-only on the client.
- This value should be set to true if within the transport media and environment the client is expected to initiate the conversation upon initial connection to the server.
- This value is typically true for all periodic socket connections, and may be false for occasional (sporadic) connections.

Client initial brief message	
Property Key	**0xF305** PROP_COMM_FIRST_BRIEF ("com.brief")
Description	Client is to send ID only on first packet block sent to server.
Attributes	Boolean
Value Byte:Len	***Value Field Description***
0:1	Non-zero if client is to send only a brief message on initial connection.

Notes:

- If 'true', client must send only ID and EOB packets on **first** packet block. It must not send any other packets. Subsequent blocks may include other packets such as event, diagnostics, and errors.
- The purpose of this property is to allow the server to reconfigure the client prior to the client sending its unacknowledged events.

Time to wait between connection failures	
Property Key	**0xF309** PROP_COMM_FAILURE_DELAY ("com.faildelay")
Description	The amount of time to wait between connection failures.
Attributes	Uint32, Array
Value Byte:Len	***Value Field Description***
0:4	Minimum delay (in seconds) between failed connections
4:4	Maximum delay (in seconds) between failed connections

Notes:

- With some wireless service providers, the process of establishing a connection to the network (ie. PDP context) consumes bytes allocated to the purchased data plan. This occurs before the client device has attempted to connect to the server, and before any data has been transmitted. As a result, connection and data transmission failures must also be managed, otherwise the process connecting and re-connecting can consume all allotted bytes in the data plan.
- On the first connection failure (after a successful connection), the client must wait at least the minimum delay seconds before attempting another connection, but not more than the maximum delay seconds. Typically, the client may implement a "backoff" strategy where the amount of delay for subsequent failure delays is doubled until the maximum delay time is reached.
- This property may be ignored for networks that do not impose a data penalty for connection attempts (ie. Bluetooth, etc).

Maximum connections	
Property Key	**0xF311** PROP_COMM_MAX_CONNECTIONS ("com.maxconn")
Description	Maximum number of allowed connections per time period
Attributes	Uint8, Array
Value Byte:Len	***Value Field Description***
0:1	Maximum total connections per time period (Duplex + Simplex)
1:1	Maximum Duplex connections per time period
2:1	Number of minutes over which the above limits apply

Notes:

- These values should match those provided by the level of service granted by the DMT service provider.
- The number of total connections should always be >= the number of Duplex connections.
- The number of Duplex connection should be set to '0' if all messages are to transmitted via Simplex (eg. UDP).

Minimum transmit delay	
Property Key	**0xF312** PROP_COMM_MIN_XMIT_DELAY ("com.mindelay")
Description	Absolute minimum time delay (seconds) between transmit intervals
Attributes	Uint16
Value Byte:Len	***Value Field Description***
0:2	Minimum time in seconds between transmissions

Notes:

- The device must never transmit more often than the interval specified by this property (even for critical events).

Minimum non-critical transmit interval	
Property Key	**0xF313** PROP_COMM_MIN_XMIT_RATE ("com.minrate")
Description	Minimum data transmit interval (seconds)
Attributes	UInt32
Value Byte:Len	***Value Field Description***
0:4	Minimum time in seconds between transmissions of non-critical events.

Notes:

- For non-critical events, the device should never transmit more often than the interval specified by this property.

Maximum transmit interval	
Property Key	**0xF315** PROP_COMM_MAX_XMIT_RATE ("com.maxrate")
Description	Maximum data transmit interval
Attributes	UInt32
Value Byte:Len	***Value Field Description***
0:4	Maximum time in seconds between transmissions

Notes:

- If this amount of time passes without any data transmission, the client must initiate a non-data transmission to see if the server wishes to send the client any information or reconfiguration.
- This value should never be less than PROP_COMM_MIN_XMIT_RATE.

Maximum Duplex events	
Property Key	**0xF317** PROP_COMM_MAX_DUP_EVENTS ("com.maxduplex")
Description	Maximum events to send per block (on Duplex connections)
Attributes	UInt8
Value Byte:Len	***Value Field Description***
0:1	Maximum number of events to send per acknowledge block

Notes:

- This value should be at least 1, but should not be greater than 128. The server may refuse the data if greater than 128.

Maximum Simplex events	
Property Key	**0xF318** PROP_COMM_MAX_SIM_EVENTS ("com.maxsimplex")
Description	Maximum number of events to send per Simplex transmission
Attributes	UInt8
Value Byte:Len	***Value Field Description***
0:1	Maximum number of events to send per Simplex transmission

Notes:

- This value should be at least 1, but should not be greater than 16. Since Simplex transmissions may not guarantee delivery (UDP does not), making this value larger may result in a more significant data loss should a particular message be lost.

Communication settings	
Property Key	**0xF3A0** PROP_COMM_SETTINGS ("com.settings")
Description	Communication settings - as defined by device
Attributes	ASCIIZ, Optional
Value Byte:Len	***Value Field Description***
0:X	ASCIIZ String defining the communication settings needed by the Device.

Notes:

- The client may choose to make this read-only.
- The contents and format of the string is defined by the client.

Remote server host	
Property Key	**0xF3A1** PROP_COMM_HOST ("com.host")
Description	Communication settings host name
Attributes	ASCIIZ, Optional
Value Byte:Len	***Value Field Description***
0:X	ASCIIZ host name or ip address, identifier

Notes:

- The client may choose to make this read-only.

Remote server port	
Property Key	**0xF3A2** PROP_COMM_PORT ("com.port")
Description	Communication settings port
Attributes	UInt16, Optional
Value Byte:Len	***Value Field Description***
0:2	Remote host port number for Duplex/Simplex communications.

Notes:

- The client may choose to make this read-only.
- This value must be supplied by your DMT service provider (ie. the service to which you are connecting).

DNS server 1	
Property Key	**0xF3A3** PROP_COMM_DNS_1 ("com.dns1")
Description	Communication settings DNS 1
Attributes	ASCIIZ, Optional
Value Byte:Len	***Value Field Description***
0:X	ASCIIZ DNS ip address (primary)

Notes:

- The client may choose to make this read-only.
- This value must be supplied by your GSM/GPRS airtime service provider.

DNS server 2	
Property Key	**0xF3A4** PROP_COMM_DNS_2 ("com.dns2")
Description	Communication settings DNS 2
Attributes	ASCIIZ, Optional
Value Byte:Len	***Value Field Description***
0:X	ASCIIZ DNS ip address (primary)

Notes:

- The client may choose to make this read-only.
- This value must be supplied by your GSM/GPRS airtime service provider.

Communication Connection name	
Property Key	**0xF3A5** PROP_COMM_CONNECTION ("com.connection")
Description	Communication settings Connection name
Attributes	ASCIIZ, Optional
Value Byte:Len	***Value Field Description***
0:X	ASCIIZ Connection name as required by the client device.

Notes:

- The client may choose to make this read-only.
- This value is typically used by Windows CE devices to identify the connection name which can provide Internet connectivity.

Communication APN name	
Property Key	**0xF3A6** PROP_COMM_APN_NAME ("com.apnname")
Description	Communication settings APN name
Attributes	ASCIIZ, Optional
Value Byte:Len	***Value Field Description***
0:X	ASCIIZ APN name as required by the client device.

Notes:

- The client may choose to make this read-only.
- This value must be supplied by your GSM/GPRS airtime service provider.

Communication APN server	
Property Key	**0xF3A7** PROP_COMM_APN_SERVER ("com.apnserv")
Description	Communication settings APN server
Attributes	ASCIIZ, Optional
Value Byte:Len	***Value Field Description***
0:X	ASCIIZ APN server as required by the client device.

Notes:

- The client may choose to make this read-only.
- This value must be supplied by your GSM/GPRS airtime service provider.

Communication APN user	
Property Key	**0xF3A8** PROP_COMM_APN_USER ("com.apnuser")
Description	Communication settings APN user
Attributes	ASCIIZ, Optional
Value Byte:Len	***Value Field Description***
0:X	ASCIIZ APN user as required by the client device.

Notes:

- The client may choose to make this read-only.
- This value must be supplied by your GSM/GPRS airtime service provider.

Communication APN password	
Property Key	**0xF3A9** PROP_COMM_APN_PASSWORD ("com.apnpass")
Description	Communication settings APN password
Attributes	ASCIIZ, Optional
Value Byte:Len	***Value Field Description***
0:X	ASCIIZ APN password as required by the client device.

Notes:

- The client may choose to make this read-only.
- This value must be supplied by your GSM/GPRS airtime service provider.

Communication APN phone number	
Property Key	**0xF3AA** PROP_COMM_APN_PHONE ("com.apnphone")
Description	Communication settings APN phone number
Attributes	ASCIIZ, Optional
Value Byte:Len	***Value Field Description***
0:X	ASCIIZ APN phone as required by the client device.

Notes:

- The client may choose to make this read-only.
- This value is typically "*99***1#", but may be defined by your GSM/GPRS airtime service provider..

Communication APN settings	
Property Key	**0xF3AC** PROP_COMM_APN_SETTINGS ("com.apnsett")
Description	General communication settings
Attributes	ASCIIZ, Optional
Value Byte:Len	***Value Field Description***
0:X	ASCIIZ general communication settings

Notes:

- The client may choose to make this read-only.
- This property is provided to cover configuration items not covered in the other communication properties.

Communication Minimum Signal Strength	
Property Key	**0xF3AD** PROP_COMM_MIN_SIGNAL ("com.minsignal")
Description	Minimum signal strength required to establish connection.
Attributes	UInt8, Optional
Value Byte:Len	***Value Field Description***
0:1	Typically a value between 0 and 31, inclusive.

Notes:

- The client use this value to compare against the value returned from a "AT+CSQ" request from the modem.

Communication Minimum Signal Strength	
Property Key	**0xF3AF** PROP_COMM_ACCESS_PIN ("com.pin")
Description	Access PIN/Password
Attributes	UInt8, Optional
Value Byte:Len	***Value Field Description***
0:X	Access pin/password

Notes:

- The client may choose to keep this value secret and return a 0-length value for property requests.
- This PIN may be used as needed by the client for access control.

Remote server port	
Property Key	**0xF3B2** PROP_COMM_UPLOAD_PORT ("com.uplport")
Description	Communication settings 'upload' file transfer port
Attributes	UInt16, Optional
Value Byte:Len	***Value Field Description***
0:2	Remote host port number for out-of-band file transfers

Notes:

- The client may choose to make this read-only.
- This value must be supplied by your DMT service provider (ie. the service to which you are connecting).

Server supports custom event packets	
Property Key	**0xF3C0** PROP_COMM_CUSTOM_FORMATS ("com.custfmt")
Description	True if the server supports custom formats for this client
Attributes	Boolean
Value Byte:Len	***Value Field Description***
0:1	Non-zero if the server supports custom formats

Notes:

- This is a hint to whether or not the DMT service provider will support custom formats from this device. The client may use this information to decide whether it should attempt sending custom event packets.

Server supported encodings	
Property Key	**0xF3C1** PROP_COMM_ENCODINGS ("com.encodng")
Description	Mask indicating the encodings supported by the server.
Attributes	Uint8 (mask)
Value Byte:Len	***Value Field Description***
0:1	Bitmask indicating supported encodings 0x01 - Binary (always true) 0x02 - Ascii Base64 (always true) 0x04 - Ascii Hex (always true) 0x08 - Ascii CSV (server support is optional) 0xF0 - reserved

Notes:

- This is a hint to whether or not the DMT service provider will support the specified encoding for this device.

Bytes read by client	
Property Key	**0xF3F1** PROP_COMM_BYTES_READ ("com.rdcnt")
Description	Number of bytes read by client
Attributes	UInt32, Optional
Value Byte:Len	***Value Field Description***
0:4	Number of bytes read since reset

Notes:

- This is for information purposes only and the client is not required to implement this property.

Bytes written by client	
Property Key	**0xF3F2** PROP_COMM_BYTES_WRITTEN ("com.wrcnt")
Description	Number of bytes written by client
Attributes	UInt32, Optional
Value Byte:Len	***Value Field Description***
0:4	Number of bytes written since reset

Notes:

- This is for information purposes only and the client is not required to implement this property.

A.8) GPS Sampling Configuration Properties

GPS sample interval	
Property Key	**0xF511** PROP_GPS_SAMPLE_RATE ("gps.smprate")
Description	GPS sample interval
Attributes	UInt16
Value Byte:Len	***Value Field Description***
0:2	Number of seconds between GPS samples

Notes:

- This value represent the amount of time to wait between GPS location acquisition and analysis. This value is typically a short amount of time, somewhere between 5 and 30 seconds

GPS aquire wait	
Property Key	**0xF512** PROP_GPS_AQUIRE_WAIT ("gps.aquwait")
Description	Amount of time to block when waiting for a current GPS fix
Attributes	UInt16
Value Byte:Len	***Value Field Description***
0:2	Number of milliseconds to block (1 to 5000 milliseconds).

Notes:

- This value should be in the range of 0 to 5000 milliseconds.
- '0' is defined by the client, but typically means that last valid fix should be immediately used.

GPS expiration	
Property Key	**0xF513** PROP_GPS_EXPIRATION ("gps.expire")
Description	GPS Expiration
Attributes	UInt16
Value Byte:Len	***Value Field Description***
0:2	Number of seconds after which the GPS fix is considered stale

Notes:

- The behavior of the client when a GPS fix has expired is unspecified. The client may wish to send a diagnostic/error message to the server.

GPS Update System Clock	
Property Key	**0xF515** PROP_GPS_CLOCK_DELTA ("gps.updclock")
Description	GPS Update System Clock
Attributes	UInt16, Optional
Value Byte:Len	***Value Field Description***
0:2	If non-zero, update the system clock with the latest GPS time if the time differences exceeds this property value. A value of '0' indicates to the client that the system clock should never be updated with the GPS time.

Notes:

- The use of this property is determined by the client.

GPS accuracy threshold	
Property Key	**0xF521** PROP_GPS_ACCURACY ("gps.accuracy")
Description	GPS Accuracy threshold
Attributes	Uint16, Optional
Value Byte:Len	***Value Field Description***
0:2	GPS accuracy threshold in meters

Notes:

- A GPS fix should be rejected if it's accuracy falls outside this threshold. For example, if the value is set to 80 meters and the accuracy of a given GPS fix is determined to be 100 meters, then the GPS fix should be rejected and another GPS fix should be acquired.
- Support for this property is optional (not all clients may have the ability to determine the accuracy of a GPS fix). The client may return the error DIAG_PROPERTY_INVALID_ID if it cannot support this property.

GPS minimum speed	
Property Key	**0xF522** PROP_GPS_MIN_SPEED ("gps.minspd")
Description	GPS Minimum speed
Attributes	Uint16
Value Byte:Len	***Value Field Description***
0:2	Minimum GPS speed in 0.1 KPH units

Notes:

- GPS reported speed values <= this value will be considered stopped and will be reported in location events as 0 KPH.
- The purpose of this property is to adjust for inaccuracies in some GPS modules which can report a significant speed value, even when the device is not moving.

GPS minimum distance delta	
Property Key	**0xF531** PROP_GPS_DISTANCE_DELTA ("gps.dstdelt")
Description	Minimum distance delta
Attributes	UInt32
Value Byte:Len	***Value Field Description***
0:4	The minimum distance that the device has to move (in meters) for distance accumulation (ie. Odometer).

Notes:

- The device must move this number of meters before a distance accumulation (ie. odometer) is performed. (The new GPS fix should then be stored in PROP_ODOMETER_0_GPS).
- This value should be larger than the accuracy capability of the GPS module. Setting this value too low (eg. 20 meters) may cause the device to accumulate distance even though the device isn't moving. The value should not be less than the value specified for PROP_GPS_ACCURACY. For non-WAAS enabled GPS modules, this value probably should not be less that 500 meters. For WAAS enabled modules, this value could probably be around 200 meters. Experiment with this and check the results for yourself.

A.9) Geofence/GeoZone properties

Geofence/GeoZone administrative commands	
Property Key	**0xF542** PROP_GEOF_ADMIN ("gf.admin")
Description	Geofence/GeoZone administrative commands
Attributes	Command, Write-Only, Optional
Value Byte:Len	***Value Field Description***
0:1	0x10 = Add GeoZone
1:2	Zone-ID
3:2	Mask 0xE000: Type, Mask 0x1FFF: Radius (meters)
5:6	6-byte Encoded Latitude/Longitude #1 (See Appendix E for encoding details).
11:6	6-byte Encoded Latitude/Longitude #2 (See Appendix E for encoding details).
	The above template (ZoneID to Lat/Lon) may be repeated up to 15 times per packet
Value Byte:Len	***Value Field Description [optional]***
0:1	0x11 = Add GeoZone
1:4	Zone-ID
5:2	Mask 0xE000: Type, Mask 0x1FFF: Radius (meters)
7:8	8-byte Encoded Latitude/Longitude #1 (See Appendix E for encoding details).
15:8	8-byte Encoded Latitude/Longitude #2 (See Appendix E for encoding details).
	The above template (ZoneID to Lat/Lon) may be repeated up to 11 times per packet
Value Byte:Len	***Value Field Description***
0:1	0x20 = Remove GeoZone
1:2	Zone-ID
Value Byte:Len	***Value Field Description***
0:1	0x30 = Save Changes

Notes:

- The above represents a standard method for defining arrival/departure GeoZones. The way GeoZones are internally managed are client dependent and as such it may not be possible for the client to implement this particular method. In which case, a client which cannot support this method of GeoZone definition should at least respond with COMMAND_FEATURE_NOT_SUPPORTED if this command property is called.
- In some client implementations, it may be more desirable to write the entire set of GeoZones to the client. In these cases, the use of the "Server File Upload" packet types may be preferred.

Geofence/GeoZone table entry count	
Property Key	**0xF547** PROP_GEOF_COUNT ("gf.count")
Description	The number of entries currently in the Geofence/GeoZone table
Attributes	UInt16, Read-Only, Optional
Value Byte:Len	***Value Field Description***
0:2	Current number of Geofence/GeoZone entries in the table.

Geofence/GeoZone table version	
Property Key	**0xF548** PROP_GEOF_VERSION ("gf.version")
Description	The number of entries currently in the Geofence/GeoZone table
Attributes	ASCIIZ String, Optional
Value Byte:Len	***Value Field Description***
0:X	The current table version string.

Notes:

- The format of this string may be defined by the server

.

Geofence arrival delay	
Property Key	**0xF54A** PROP_GEOF_ARRIVE_DELAY ("gf.arr.delay")
Description	Geofence arrival delay in seconds
Attributes	UInt16, Optional
Value Byte:Len	***Value Field Description***
0:2	Number of seconds that the device must be in a geofence before is is considered "arrived".

Notes:

- This purpose of this property it to prevent devices being marked as 'arrived' when they are only passing through.

Geofence departure delay	
Property Key	**0xF54D** PROP_GEOF_DEPART_DELAY ("gf.dep.delay")
Description	Geofence departure delay in seconds
Attributes	UInt16, Optional
Value Byte:Len	***Value Field Description***
0:2	Number of seconds that the device must be outside a geofence before is is considered "departed".

Notes:

- The value for this property is generally small and prevents devices being marked as 'departed' when they only left briefly. This is generally only necessary to prevent oddball bouncing GPS locations from causing multiple improper arrival/departure messages.

Current geofence	
Property Key	**0xF551** PROP_GEOF_CURRENT ("gf.current")
Description	Geofence ID in which the device is currently sitting
Attributes	UInt32, Optional
Value Byte:Len	***Value Field Description***
0:4	Current Geofence ID

Notes:

- This value should generally be set by the device itself at it enters or leave predefined geofenced areas.

Active alarmed geocorridor	
Property Key	**0xF562** PROP_CMD_GEOC_ADMIN ("gc.admin")
Description	GeoCorridor administrative commands
Attributes	Command, Write-Only, Optional
Value Byte:Len	***Value Field Description***
X:X	TBD

Active alarmed geocorridor	
Property Key	**0xF567** PROP_GEOC_ACTIVE_ID ("gc.active")
Description	The active GeoCorridor ID
Attributes	UInt32, Optional
Value Byte:Len	***Value Field Description***
0:4	Active GeoCorridor ID

Notes:

- This is typically used for identifying the active alarm state geocorridor.
- This value may be set by the device itself as it determines necessary as it enters or leaves predefined geofenced areas

Alarmed geocorridor violation interval	
Property Key	**0xF56A** PROP_GEOC_VIOLATION_INTRVL ("gc.vio.rate")
Description	Geofence violation reporting interval for a sustained geocorridor violation.
Attributes	UInt16, Optional
Value Byte:Len	***Value Field Description***
0:2	Number of seconds between geocorridor violation events.

Notes:

- This value represents the interval (in seconds) between geocorridor violation events during a sustained geocorridor violation.

Alarmed geocorridor violation count	
Property Key	**0xF56D** PROP_GEOC_VIOLATION_COUNT ("gc.vio.cnt")
Description	Maximum number of geocorridor violation messages to send
Attributes	UInt16, Optional
Value Byte:Len	***Value Field Description***
0:2	Maximum number of violation messages to send during a geocorridor violation.

Notes:

- This value represents the number of geocorridor violation events that should be sent once the device has determined that a geocorridor violation has occurred.

A.10) Motion properties

Motion start type	
Property Key	**0xF711** PROP_MOTION_START_TYPE ("mot.start.type")
Description	Motion start type
Attributes	UInt8
Value Byte:Len	***Value Field Description***
0:1	Motion start type (0=GPS kph, 1=GPS meters moved, 2=OBC kph, 3=OBC/GPS kph)

Notes:

- This property defines the meaning of the value for the property PROP_MOTION_START. If this value is '0', then motion-start is defined if KPH. If this value is '1', then motion-start is defined in the number of meters moved. Values 2/3 are reserved for OBC based motion start definitions.
- The client may wish to make this property read-only.

Motion start definition	
Property Key	**0xF712** PROP_MOTION_START ("mot.start")
Description	Motion start definition
Attributes	UInt16
Value Byte:Len	***Value Field Description***
0:2	Definition of start of motion in 0.1 KPH/Meters units

Notes:

- A value of 0 means that stop/stop motion events are not currently enabled.
- Whether this value is interpreted as KPH or Meters depends on the value of the property PROP_MOTION_START_TYPE.

In-Motion interval	
Property Key	**0xF713** PROP_MOTION_IN_MOTION ("mot.inmotion")
Description	In-motion periodic event interval
Attributes	UInt16
Value Byte:Len	***Value Field Description***
0:2	Number of seconds between in-motion events

Notes:

- A value of 0 means that no in-motion events are to be generated.

Motion stop definition	
Property Key	**0xF714** PROP_MOTION_STOP ("mot.stop")
Description	Motion stop definition
Attributes	UInt16
Value Byte:Len	***Value Field Description***
0:2	Definition of end of motion in number of seconds (1 to 65535 sec)

Notes:

- The device will be considered "stopped" after this period of time has elapsed without any motion.

Motion stop definition	
Property Key	**0xF715** PROP_MOTION_STOP_TYPE ("mot.stop.type")
Description	Motion stop type definition
Attributes	UInt8
Value Byte:Len	***Value Field Description***
0:1	0=After-Delay, 1=When-Stopped (2 to 255 reserved)

Notes:

- This value defines the effect of the value for the property PROP_MOTION_STOP. If thie value is '0', then the stop-motion event will be generated with a timestamp at the time the PROP_MOTION_STOP timer has expired. Also, in-motion messages will be generated on a continued schedule interval until the stop-motion event is generated. If this value is '1', then the timestamp of the generated stop-motion event will be the time that the vehicle actually stopped (the stop-motion event is delayed until the stopped timer is expired). Also, in-motion events will only be generated if the vehicle is actually in motion at the time the in-motion event is to be generated.

Dormant interval	
Property Key	**0xF716** PROP_MOTION_DORMANT_INTRVL ("mot.dorm.rate")
Description	Dormant periodic event interval
Attributes	UInt32
Value Byte:Len	***Value Field Description***
0:4	Number of seconds between dormant events (1 to 4294967295 sec)

Notes:

- A value of 0 indicates that dormant events are disabled.
- This value represents the interval (in seconds) between dormant events once the device has determined that it is no longer moving. The number of dormant messages sent is defined by the property PROP_MOTION_DORMANT_COUNT.

Dormant count	
Property Key	**0xF717** PROP_MOTION_DORMANT_COUNT ("mot.dorm.cnt")
Description	Maximum number of dormant messages to send
Attributes	UInt16
Value Byte:Len	***Value Field Description***
0:2	Maximum number of dormant messages to send during dormancy

Notes:

- This value represents the number of dormant messages that should be sent once the device has determined that it is no longer moving. Typically, this value is 0 (indefinite), however it may be desirable to have a limited number of dormant messages sent by the client.

Excess speed	
Property Key	**0xF721** PROP_MOTION_EXCESS_SPEED ("mot.exspeed")
Description	Excess speed limit
Attributes	UInt16
Value Byte:Len	***Value Field Description***
0:2	Definition of excess speed in 0.1 KPH units

Notes:

- An excess speed event should be generated if the current speed exceeds this value.

Moving Interval	
Property Key	**0xF725** PROP_MOTION_MOVING_INTRVL ("mot.moving")
Description	'Mocing' periodic event interval
Attributes	Uint16, Optional
Value Byte:Len	***Value Field Description***
0:2	Number of seconds between 'Moving' events (1 to 65535 sec)

Notes:

- This value represents the minimum interval (in seconds) between 'moving' events if the device determines that it is moving.
- 'Moving' events may operate independently of motion start/stop/in-motion events, and may be generated even if start/stop events are not in use.
- This property is optional. The client may also decide the special conditions under which these events are generated.

A.11) Odometer properties

Odometer/Tripometer	
Property Key	**0xF770** PROP_ODOMETER_0_VALUE ("odo.0.value") ... **0xF77F** PROP_ODOMETER_F_VALUE ("odo.F.value")
Description	Device odometer/tripometer in meters
Attributes	UInt32
Value Byte:Len	***Value Field Description***
0:4	Number of meters that the device has moved since the value was last reset.

Notes:

- PROP_ODOMETER_0_VALUE should be used to represent the number of meters moved since the device was put into service. (This odometer may be read-only.)
- PROP_ODOMETER_1..7_VALUE may be used for special 'tripometer' applications.

Odometer triggers	
Property Key	**0xF780** PROP_ODOMETER_0_LIMIT ("odo.0.limit") ... **0xF78F** PROP_ODOMETER_F_LIMIT ("odo.F.limit")
Description	Device odometer/tripometer triggered alarm point in meters.
Attributes	UInt32
Value Byte:Len	***Value Field Description***
0:4	Once the client has achieved this number of meters it should trigger a corresponding STATUS_ODOM_LIMIT_# event

Notes:

- A value of '0' indicates that no alarm/event should be generated.
- Once this limit is reached, the client should issue a corresponding STATUS_ODOM_LIMIT_# event, however, the decision to reset the odometer or continue to count is left to the client.

Odometer GPS	
Property Key	**0xF790** PROP_ODOMETER_0_GPS ("odo.0.gps") ... **0xF79F** PROP_ODOMETER_F_GPS ("odo.F.gps")
Description	Device odometer GPS (point of last odometer GPS fix)
Attributes	GPS, Read-Only
Value Byte:Len	***Value Field Description***
0:4	GPS fix time
4:3 4:4	Standard-resolution encoded latitude, or High-resolution encoded latitude
7:3 8:4	Standard-resolution encoded longitude, or High-resolution encoded longitude
10:4 12:4	(Standard-resolution) odometer meters, or (High-resolution) odometer meters

Notes:

- These properties may be used by the client to maintain the GPS location state necessary to accumulate GPS-based odometer information. This property only holds a single GPS point, if the client requires a more general state cache for odometer information, the PROP_ODOMETER_#_STATE properties should be used.
- Depending on the degree of accuracy that the client wishes to provide, the client may store/return either a 6-byte, or 8-byte, encoded Lat/Long.
- The server will infer from the length of the data payload which type of encoding is used.
- These properties should be considered read-only, however this is enforced by the client. The client may allow these values to be set if necessary.

A.12) Digital input/output properties

Digital Input State	
Property Key	**0xF901** PROP_INPUT_STATE ("inp.state")
Description	Digital input state
Attributes	UInt32, Optional
Value Byte:Len	***Value Field Description***
0:4	Input state mask (least significant bit is Input #0)

Notes:

- The client may choose to make this value read-only.

Digital Input Configuration	
Property Key	**0xF910** PROP_INPUT_CONFIG_0 ("inp.0.conf") ... **0xF91F** PROP_INPUT_CONFIG_F ("inp.F.conf")
Description	Digital input configuration
Attributes	UInt32, Array, Optional
Value Byte:Len	***Value Field Description***
0:4	Support mask 0x0000000F – Debounce interval (as interpreted by the client) 0x00000010 – Trigger event when state changes to 'On' 0x00000020 – Trigger event when state changes to 'Off' 0x00000080 – High priority (when used with event generation) 0x00000100 – Start elapse-timer when state changes to 'On' 0x00000200 – Start elapse-timer when state changes to 'Off' 0x00001000 – Trigger output cycle when state changes to 'On' 0x00002000 – Trigger output cycle when state changes to 'Off'
4:4	Reserved.

Notes:

- A 'support mask' of '0' indicates that this digital input will be ignored.
- A debounce interval of '0' means that no 'debounce' will occur. This feature is intended for inputs that may undergo several quick state changes before settling down. For instance, a digital input triggered by an ignition switch may go through several on/off/on state changes as the driver attempts to start the vehicle. Instead of recording all of these state changes, this debounce' feature can be used to only record the ignition-on if the input remains 'true' for a short period of time (eg. 20 seconds).
- The interpretation of non-zero debounce values is left to the client.
- The triggered outputs are determined by client implementation.

Digital Output Configuration	
Property Key	**0xF930** PROP_OUTPUT_CONFIG_0 ("out.0.conf") ... **0xF93F** PROP_OUTPUT_CONFIG_F ("out.F.conf")
Description	Digital output configuration
Attributes	UInt32, Array, Optional
Value Byte:Len	***Value Field Description***
0:4	Support mask 0x00000010 – Trigger event when ouput is turned 'On' 0x00000020 – Trigger event when ouput is turned 'Off' 0x00000080 – High priority (when used with event generation)
4:4	Maximum output 'On' time (in milliseconds).

Notes:

- A 'support mask' of '0' indicates that this digital output will be ignored.
- While the 'maximum on' time is specified in milliseconds, it may not be possible for the client to provide that level of granularity. In this case the client may choose to round up to the next nearest second if necessary.

Elapsed time value	
Property Key	**0xF960** PROP_ELAPSED_0_VALUE ("ela.0.value") ... **0xF96F** PROP_ELAPSED_F_VALUE ("ela.F.value")
Description	Device elapsed timer accumulated value in seconds
Attributes	UInt32, Optional
Value Byte:Len	***Value Field Description***
0:4	The current value of the elapsed timer in seconds

Notes:

- The client may choose how often this value is updated.

Elapsed time limit trigger	
Property Key	**0xF980** PROP_ELAPSED_0_LIMIT ("ela.0.limit") ... **0xF98F** PROP_ELAPSED_F_LIMIT ("ela.F.limit")
Description	Device elapsed timer triggered alarm point
Attributes	UInt32, Optional
Value Byte:Len	***Value Field Description***
0:4	Elapsed timer seconds limit

Notes:

- Once the client has achieved this number of seconds it should trigger a corresponding STATUS_ELAPSED_LIMIT_# event, however, the decision to reset the timer or continue to count is left to the client.
- A value of '0' indicates that no alarm/event will be generated.

A.13) Analog/Sensor configuration properties

Power supply undervoltage limit	
Property Key	**0xFB01** PROP_UNDERVOLTAGE_LIMIT ("bat.limit")
Description	Battery/Power-supply undervoltage limit
Attributes	UInt32, Optional
Value Byte:Len	***Value Field Description***
0:4	Undervoltage limit (in millivolts)

Notes:

- When the supply voltage falls below this value, the client should issue a STATUS_LOW_BATTERY event. The client may decide how often this event is to be repeated should the voltage remain below this threshold.
- A value of '0' indicates that no undervoltage alarm/event will be generated.

Sensor configuration	
Property Key	**0xFB10** PROP_SENSOR_CONFIG_0 ("sen.0.conf") ... **0xFB1F** PROP_SENSOR_CONFIG_F ("sen.F.conf")
Description	Sensor configuration
Attributes	UInt32, Array, Optional
Value Byte:Len	***Value Field Description***
0:4	Gain (as defined by the client)
4:4	Offset (as defined by the client)

Notes:

- The units of these values are defined by the client.

Sensor range limit	
Property Key	**0xFB20** PROP_SENSOR_RANGE_0 ("sen.0.range") ... **0xFB2F** PROP_SENSOR_RANGE_F ("sen.F.range")
Description	Sensor high/low range limit trigger
Attributes	UInt32, Array, Optional
Value Byte:Len	***Value Field Description***
0:4	Low range
4:4	High range

Notes:

- The units of this range is defined by the client.

A.14) Temperature configuration properties

Temperature sensor sampling interval	
Property Key	**0xFB60** PROP_TEMP_SAMPLE_INTRVL ("tmp.smprate")
Description	Temperature sensor sampling interval
Attributes	UInt32, Optional
Value Byte:Len	***Value Field Description***
0:4	Temperature sampling interval (in seconds)
4:4	Port 'close' indicator (zero to close port after sampling, non-zero to leave port open)

Notes:

- This sampling interval specifies the interval between temperature sensor samples.
- The port 'close' indicators allow specifying whether the temperature monitor is to be left 'on' between samplings. Some client devices may not have the ability to turn off the temperature monitor, so this value may be ignored.

Temperature sensor reporting interval	
Property Key	**0xFB63** PROP_TEMP_REPORT_INTRVL ("tmp.rptrate")
Description	Temperature sensor reporting intervals
Attributes	UInt32, Optional
Value Byte:Len	***Value Field Description***
0:4	Temperature periodic reporting interval (in seconds)
4:4	Temperature alarm reporting interval (in seconds)

Notes:

- The periodic reporting interval specifies the interval between temperature sensor periodic events.
- The alarm reporting interval specifies the interval between temperature sensor alarm events while the temperature is out of range.

Temperature sensor configuration	
Property Key	**0xFB70** PROP_TEMP_CONFIG_0 ("tmp.0.conf") ... **0xFB77** PROP_TEMP_CONFIG_7 ("tmp.7.conf")
Description	Temperature sensor configuration
Attributes	UInt32, Array, Optional
Value Byte:Len	***Value Field Description***
0:4	Temperature configuration 1 (as defined by the client)
4:4	Temperature configuration 2 (as defined by the client)

Notes:

- This is an optional property for clients which may provide temperature sensors.
- The interpretation of the values provided by this property is left o the client device. For instance, the values could be interpreted as gain and offset for an analog temperature sensor, or it could be used as a temperature convergence factor for averaging.

Temperature sensor range limit	
Property Key	**0xFB80** PROP_TEMP_RANGE_0 ("tmp.0.range") ... **0xFB87** PROP_TEMP_RANGE_7 ("tmp.7.range")
Description	Temperature sensor high/low range
Attributes	Int16, Signed, Array, Optional
Value Byte:Len	***Value Field Description***
0:2	Temperature signed low range (-3276.7C to +3276.7C)
2:2	Temperature signed high range (-3276.7C to +3276.7C)

Notes:

- This is an optional property for clients which may provide temperature sensors.

A.14) Accelerometer properties

Maximum acceptable hard-braking G-force	
Property Key	**0xFBA0** PROP_MAX_BRAKE_G_FORCE ("acc.maxbrake")
Description	Maximum acceptable hard-braking G-force
Attributes	UInt8, Optional
Value Byte:Len	***Value Field Description***
0:1	G-Force range 0.0 to 25.5

Notes:

- This is an optional property for clients which may provide temperature sensors.
- The interpretation of the values provided by this property is left o the client device. For instance, the values could be interpreted as gain and offset for an analog temperature sensor, or it could be used as a temperature convergence factor for averaging.

Appendix 'B' - Standard Status Codes

Status code definitions can also be found in the header file “base/statcode.h”

B.1) Generic codes

Status Code	**0xF010** STATUS_INITIALIZED
Description	General Status/Location information. This event is generated by some initialization function performed by the device.

Notes:

– This contents of the payload must at least contain the current timestamp (and latitude and longitude if available).

Status Code	**0xF020** STATUS_LOCATION
Description	General Status/Location information.

Notes:

– This contents of the payload must at least contain the current timestamp, latitude, and longitude.

Status Code	**0xF030** STATUS_WAYMARK_0 ... **0xF032** STATUS_WAYMARK_2
Description	General Status/Location information. This event is generated by manual user intervention at the device, such as by pressing a "Waymark" button on the UI.

Notes:

– This contents of the payload must at least contain the current timestamp, latitude, and longitude.

Status Code	**0xF040** STATUS_QUERY
Description	General Status/Location information. This event is generally sent as a response to a location request ("ping") from the server.

Notes:

– This contents of the payload must at least contain the current timestamp, latitude, and longitude.

B.2) Motion codes

Status Code	**0xF111** STATUS_MOTION_START
Description	Device start of motion

Notes:

– The definition of motion-start is provided by property PROP_MOTION_START

Status Code	**0xF112** STATUS_MOTION_IN_MOTION
Description	Device in-motion interval

Notes:

– The in-motion interval is provided by property PROP_MOTION_IN_MOTION

Status Code	**0xF113** STATUS_MOTION_STOP
Description	Device stopped motion

Notes:

- The definition of motion-stop is provided by property PROP_MOTION_STOP

Status Code	**0xF114** STATUS_MOTION_DORMANT
Description	Device dormant interval (ie. not moving)

Notes:

- The dormant interval is provided by property PROP_MOTION_DORMANT

Status Code	**0xF11A** STATUS_MOTION_EXCESS_SPEED
Description	Device exceeded preset speed limit

Notes:

- The excess-speed threshold is provided by property PROP_MOTION_EXCESS_SPEED

Status Code	**0xF130** STATUS_ODOM_0 ... **0xF137** STATUS_ODOM_7
Description	Odometer value

Notes:

- The odometer limit is provided by property PROP_ODOMETER_#_LIMIT

Status Code	**0xF140** STATUS_ODOM_LIMIT_0 ... **0xF147** STATUS_ODOM_LIMIT_7
Description	Odometer has exceeded a set limit

Notes:

- The odometer limit is provided by property PROP_ODOMETER_#_LIMIT

B.3) Geofence codes

Status Code	**0xF210** STATUS_GEOFENCE_ARRIVE
Description	Device arrived at geofence

Notes:

- Client may wish to include FIELD_GEOFENCE_ID in the event packet.

Status Code	**0xF230** STATUS_GEOFENCE_DEPART
Description	Device departed geofence

Notes:

- Client may wish to include FIELD_GEOFENCE_ID in the event packet.

Status Code	**0xF250** STATUS_GEOFENCE_VIOLATION
Description	Geofence violation

Notes:

- Client may wish to include FIELD_GEOFENCE_ID in the event packet.

Status Code	**0xF270** STATUS_GEOFENCE_ACTIV
Description	Geofence now active

Notes:
- Client may wish to include FIELD_GEOFENCE_ID in the event packet.

Status Code	**0xF280** STATUS_GEOFENCE_INACTIVE
Description	Geofence now inactive

Notes:
- Client may wish to include FIELD_GEOFENCE_ID in the event packet.

Status Code	**0xF2A0** STATUS_STATE_ENTER
Description	Entered a new state (crossed a stateline boundary)

Status Code	**0xF2B0** STATUS_STATE_EXIT
Description	Exited a state (crossed a stateline boundary)

B.4) Digital input/output (state change) codes

Status Code	**0xF400** STATUS_INPUT_STATE
Description	Current input ON state (bitmask)

Notes:
- Client should include FIELD_INPUT_STATE in the event packet,otherwise this status code would have no meaning.

Status Code	**0xF402** STATUS_INPUT_ON
Description	Input turned ON

Notes:
- Client should include FIELD_INPUT_ID in the event packet, otherwise this status code would have no meaning.
- This status code may be used to indicate that an arbitrary input 'thing' turned ON, and the 'thing' can be identified by the 'Input ID'. This 'ID' can also represent the index of a digital input.

Status Code	**0xF404** STATUS_INPUT_OFF
Description	Input turned OFF

Notes:
- Client should include FIELD_INPUT_ID in the event packet, otherwise this status code would have no meaning.
- This status code may be used to indicate that an arbitrary input 'thing' turned OFF, and the 'thing' can be identified by the 'Input ID'. This 'ID' can also represent the index of a digital input.

Status Code	**0xF406** STATUS_OUTPUT_STATE
Description	Current output ON state (bitmask)

Notes:

- Client should include FIELD_OUTPUT_STATE in the event packet, otherwise this status code would have no meaning.

Status Code	**0xF408** STATUS_OUTPUT_ON
Description	Output turned ON

Notes:

- Client should include FIELD_OUTPUT_ID in the event packet, otherwise this status code would have no meaning.
- This status code may be used to indicate that an arbitrary output 'thing' turned ON, and the 'thing' can be identified by the 'Output ID'. This 'ID' can also represent the index of a digital output.

Status Code	**0xF40A** STATUS_OUTPUT_OFF
Description	Output turned OFF

Notes:

- Client should include FIELD_OUTPUT_ID in the event packet, otherwise this status code would have no meaning.
- This status code may be used to indicate that an arbitrary output 'thing' turned OFF, and the 'thing' can be identified by the 'Output ID'. This 'ID' can also represent the index of a digital output.

Status Code	**0xF420** STATUS_INPUT_ON_00 ... **0xF427** STATUS_INPUT_ON_07
Description	Digital input state changed to ON

Status Code	**0xF440** STATUS_INPUT_OFF_00 ... **0xF447** STATUS_INPUT_OFF_07
Description	Digital input state changed to OFF

Status Code	**0xF460** STATUS_OUTPUT_ON_00 ... **0xF467** STATUS_OUTPUT_ON_07
Description	Digital output state set to ON

Status Code	**0xF480** STATUS_OUTPUT_OFF_00 ... **0xF487** STATUS_OUTPUT_OFF_07
Description	Digital output state set to OFF

Status Code	**0xF4A0** STATUS_ELAPSED_00 ... **0xF4A7** STATUS_ELAPSED_07
Description	Elapsed time

Notes:

– Client should include FIELD_ELAPSED_TIME in the event packet, otherwise this status code would have no meaning.

Status Code	**0xF4B0** STATUS_ELAPSED_LIMIT_00 ... **0xF4B7** STATUS_ELAPSED_LIMIT_07
Description	Elapsed timer has exceeded a set limit

Notes:

– Client should include FIELD_ELAPSED_TIME in the event packet, otherwise this status code would have no meaning.

B.5) Analog/sensor codes

Status Code	**0xF600** STATUS_SENSOR32_0 ... **0xF607** STATUS_SENSOR32_7
Description	32-bit unsigned sensor value

Notes:

– Client should include FIELD_SENSOR32 in the event packet, otherwise this status code would have no meaning.

– The server must be able to convert this 32-bit value to something meaningful to the user. This can be done using the following formula: Actual_Value = ((double)Sensor32_Value * <Gain>) + <Offset>; Where <Gain> & <Offset> are user configurable values provided at setup. For instance: Assume Sensor32-0 contains a temperature value that can have a range of -40.0C to +125.0C. The client would encode -14.7C by adding 40.0 and multiplying by 10.0. The resulting value would be 253. The server would then be configured to know how to convert this value back into the proper temperature using the above formula by substituting 0.1 for <Gain>, and -40.0 for <Offset>: eg. -14.7 = ((double)253 * 0.1) + (-40.0);

Status Code	**0xF620** STATUS_SENSOR32_RANGE_0 ... **0xF627** STATUS_SENSOR32_RANGE_7
Description	32-bit unsigned sensor value out-of-range violation

Notes:

– Client should include FIELD_SENSOR32 in the event packet, otherwise this status code would have no meaning.

B.6) Temperature codes

Status Code	**0xF710** STATUS_TEMPERATURE_0 ... **0xF717** STATUS_TEMPERATURE_7
Description	Temperature value

Notes:

- Client should include at least the field FIELD_TEMP_AVER in the event packet, and may also wish to include FIELD_TEMP_LOW and FIELD_TEMP_HIGH.

Status Code	**0xF730** STATUS_TEMPERATURE_RANGE_0 ... **0xF737** STATUS_TEMPERATURE_RANGE_7
Description	Temperature value out-of-range [low/high/average]

Notes:

- Client should include at least one of the fields FIELD_TEMP_AVER, FIELD_TEMP_LOW, or FIELD_TEMP_HIGH.

Status Code	**0xF7F1** STATUS_TEMPERATURE
Description	All temperature averages [aver/aver/aver/...]

B.7) Miscellaneous Codes

Status Code	**0xF811** STATUS_LOGIN
Description	Generic 'login'

Status Code	**0xF812** STATUS_LOGOUT
Description	Generic 'logout'

Status Code	**0xF821** STATUS_CONNECT
Description	Connect/On

Status Code	**0xF822** STATUS_DISCONNECT
Description	Disconnect/Off

Status Code	**0xF831** STATUS_ACK
Description	Acknowledge

Status Code	**0xF832** STATUS_NAK
Description	Negative Acknowledge

B.9) OBC/J1708 Status

Status Code	**0xF911** STATUS_OBC_FAULT
Description	Generic OBC/J1708 value out-of-range

Status Code	**0xF920** STATUS_OBC_RANGE
Description	Generic OBC/J1708 value out-of-range

Status Code	**0xF922** STATUS_OBC_RPM_RANGE
Description	OBC/J1708 RPM out-of-range

Status Code	**0xF924** STATUS_OBC_FUEL_RANGE
Description	OBC/J1708 Fuel level out-of-range (ie. to low)

Status Code	**0xF926** STATUS_OBC_OIL_RANGE
Description	OBC/J1708 Oil level out-of-range (ie. to low)

Status Code	**0xF928** STATUS_OBC_TEMP_RANGE
Description	OBC/J1708 Temperature out-of-range

Status Code	**0xF930** STATUS_EXCESS_BRAKING
Description	J1708 Detected rapid deceleration/acceleration

B.8) Internal Device Status

Status Code	**0xFD10** STATUS_LOW_BATTERY
Description	Low Battery indication

Status Code	**0xFD13** STATUS_POWER_FAILURE
Description	Power failure indication

Appendix 'C' - Standard Client to Server Packet Types

Client to Server packet definitions can also be found in the header file “base/packets.h”

End of block/transmission ("I'm done talking, no more to say"):
This packet indicates to the server that the client is done transmitting and will be waiting for further instructions from the server.

Byte:Len	*Value (Hex)*	*Description*
0:2	E000	Packet type
2:1	XX	Payload length [0x00 or 0x02]
3:2	XXXX	Fletcher checksum [optional]

Notes:

- The Fletcher checksum is optional, but must only be used when this packet is sent via binary encoding. If included on an ASCII encoded packet,the response from the server is undefined (it may ignore the checksum, or it may respond with an error).
- The Fletcher checksum is calculated on bytes actually transmitted to the server, and includes all bytes since the last sent EOB packet.

End of block/transmission ("I'm done talking, but I do have more to say"):
This packet indicates to the server that the client is done transmitting, however the client does have more to say and will be waiting for further instructions from the server.

Byte:Len	*Value (Hex)*	*Description*
0:2	E001	Packet type
2:1	XX	Payload length [0x00 or 0x02]
3:2	XXXX	Fletcher checksum [optional]

Notes:

- The Fletcher checksum is optional, but must only be used when this packet is sent via binary encoding. If included on an ASCII encoded packet, the response from the server is undefined (it may ignore the checksum, or it may respond with an error).
- The Fletcher checksum is calculated on bytes actually transmitted to the server, and includes all bytes since the last sent EOB packet.

Unique assigned identifier:
This packet identifies the Unique-ID for the following communication session and unique identifies the client to the server. This unique ID is generated and assigned at setup and is configured into the device.

Byte:Len	*Value (Hex)*	*Description*
0:2	E011	Packet type
2:1	XX	Payload length [0x04 to 0x14]
3:6	XX..XX	6 byte unique id

Notes:

- When $E011 is used to identify a client device, $E012 and $E013 must not be used.
- This UniqueID is ideally intended to uniquely identify a specific device among all possible DMTP devices. However, this may not be entirely practical due to the current lack of a central registry.   As a "convention", it is proposed that 4-byte IDs be reserved for ESN numbers, 6-byte IDs be reserved for a future DMTP registry, and 7-byte IDs be reserved for GSM IMEI numbers.

Account identifier:

This packet identifies the Account-ID for the following communication session.

Byte:Len	***Value (Hex)***	***Description***
0:2	E012	Packet type
2:1	XX	Payload length [0x01 to 0x10]
3:X	XX..XX	ASCII account name [case insensitive]

Notes:

- This packet may be optional if and only if the server has some other pre-arranged means of uniquely identifying this device with some Account ID.

Device identifier:

This packet identifies the Device-ID for the following communication session.

Byte:Len	***Value (Hex)***	***Description***
0:2	E013	Packet type
2:1	XX	Payload length [0x01 to 0x10]
3:X	XX..XX	ASCII device name [case insensitive]

Notes:

- This packet may be optional if and only if the server has some other pre-arranged means of uniquely identifying this device within the Account ID.

Standard GPS packet (standard resolution):

This is a standard GPS packet which provides fields for various GPS related fields. The accuracy of the lat/lon encoding will allow for an approximate +/- 12 meter resolution (provided the GPS module itself can support this resolution).

Byte:Len	***Value (Hex)***	***Description***
0:2	E030	Packet type
2:1	XX	Payload length [0x14]
3:2	XXXX	Status code [see 'base/statcode.h']
5:4	XXXXXXXX	Timestamp [POSIX Epoch time]
9:3	XXXXXX	Latitude [+/- 12 meter resolution] (See Appendix E for encoding details).
12:3	XXXXXX	Longitude [+/- 12 meter resolution] (See Appendix E for encoding details).
15:1	XX	Speed [0 to 255 kph]
16:1	XX	Heading [0 to 360 degrees (in 1.41 degree increments)]
17:2	XXXX	Altitude [-32767 to +32767 meters]
19:3	XXXXXX	Distance [0 to 16777215 km]
22:1	XX	Sequence [0 to 255]

Standard GPS packet (high resolution):

This is a standard GPS packet which provides fields for various GPS related fields. The accuracy of the lat/lon encoding will allow for an approximate +/- 2 meter resolution (provided the GPS module itself can support this resolution).

Byte:Len	***Value (Hex)***	***Description***
0:2	E031	Packet type
2:1	XX	Payload length [0x19]
3:2	XXXX	Status code [see 'base/statcode.h']
5:4	XXXXXXXX	Timestamp [POSIX Epoch time]
9:4	XXXXXXXX	Latitude [+/- 2 meter resolution] (See Appendix E for encoding details).
13:4	XXXXXXXX	Longitude [+/- 2 meter resolution] (See Appendix E for encoding details).
17:2	XXXX	Speed [0.0 to 655.3 kph]
19:2	XXXX	Heading [0.00 to 360.00 deg]
21:3	XXXXXX	Altitude [-838860.7 to +838860.7 meters]
24:3	XXXXXX	Distance [0.0 to 1677721.6 km]
27:1	XX	Sequence [0 to 255]

Custom format packets:

These packet types are for custom formats designed for specific applications. When constructing the custom format, the status-code and timestamp should be included, however it is not required. If the status code is missing, then the code STATUS_LOCATION will be substituted. If the timestamp is missing then the time the server receives the data will be used.

Byte:Len	***Value (Hex)***	***Description***
0:2	E070.. ..E07F	Packet type
2:1	XX	Payload length
3:X	XX..XX	Custom data fields [see 'base/events.h']

Get Property value:

This packet is provided on the request of the server and provides the value of a specific internal state property.

Byte:Len	***Value (Hex)***	***Description***
0:2	E0B0	Packet type
2:1	XX	Payload length
3:2	XXXX	Property ID [see 'base/props.h']
5:X	XX..XX	Property value

Custom format template:

This packet identifies the format of any custom packet types. As many of these records may be sent as needed to identify any custom packet type. This packet should be sent in response to the server error NAK_FORMAT_NOT_RECOGNIZED which indicates that the server does not recognize the previously sent custom format packet type.

Byte:Len	***Value (Hex)***	***Description***
0:2	E0CF	Packet type
2:1	XX	Payload length
3:1	XX	Custom client packet type [0x70 to 0x7F]
4:1	XX	Number of fields
5:3	FFFFFF	Field definition [see 'base/events.h'] Bits 23:1 HiRes [800000] 16:7 Type [7F0000] 8:8 Index [00FF00] 0:8 Length [0000FF]
8:3	FFFFFF	Next field definition

Notes:

- When the server receives this packet, it must cache this information for later use when the client again later sends data to the server.

Diagnostic codes:

This packet may be sent by the client to indicate generic diagnostic information. These packets may be noted by the server in some manner, but are otherwise ignored.

Byte:Len	***Value (Hex)***	***Description***
0:2	E0D0	Packet type
2:1	XX	Payload length
3:2	XXXX	Diagnostic code [see 'base/cdiags.h']
5:X	XX..XX	Diagnostic data

Error codes:

This packet may be sent by the client to indicate to the server specific issues, problems, or errors that the device has detected.

Byte:Len	***Value (Hex)***	***Description***
0:2	E0E0	Packet type
2:1	XX	Payload length
3:2	XXXX	Error code [see 'base/cerrors.h']
5:X	XX..XX	Error data

Appendix 'D' - Standard Server to Client Packet Types

Client to Server packet definitions can also be found in the header file “base/packets.h”

Query for response ("I'm done talking, do you have anything to say?")

This packet is an indicator to the client that it should now respond with any data, or other information, that it needs to send to the server. This packets also revokes any "Speak freely" permission that may have been previously given to the client.

Byte:Len	*Value (Hex)*	*Description*
0:2	E000	Packet type
2:1	XX	Payload length [0x00 or 0x01]
3:1	XX	Optional maximum number of events to send during next block.

Notes:

- The optional "maximum number of events" field value may be used to limit the number of events that the client is to send during the next block of events. A value of 0x00 indicates that the client is not to send any events during the next block of packets sent to the server (other packet types may be sent). The client may choose to ignore values greater than that specified by the property "PROP_COMM_MAX_DUP_EVENTS". The server may wish to use this optional field to indicate to the client to only send diagnostic packets, rather than always including a large block of queued events. Client support of this field is optional, and the client may choose to ignore this field.

Permission to the client to speak freely.

This packet is an indicator to the client that it has been given permission to send data as it feels necessary. It does not need to wait for the server to tell it to speak. This permission may be revoked by the server at any time (via server packet "$E000"). The client may relinquish this permission at any time by sending either client packet type "$E000", or "$E001", at which time the server will assume that the client has given up it's "speak freely" permission.

Byte:Len	*Value (Hex)*	*Description*
0:2	E001	Packet type
2:1	XX	Payload length [0x00 or 0x01]
3:1	XX	Optional maximum number of events to send during next block.

Notes:

- See the "$E000" packet definition for information concerning the use of the optional "maximum number of events" field. However, the client may choose to ignore this field in speak-freely mode.
- The server will generally NOT grant "speak freely" permission to the client in "periodically" connected scenarios. "Speak freely" permission is usually reserved for scenarios where there is a prolonged connection between client and server, such as a dedicated link.
- The server is never required to grant "speak freely" permission to the client.
- If the client is granted "speak freely" permission, but does not wish to utilize this feature, it may treat this packet the same as the server "$E000" packet and relinquish this permission with either of the client end-of-block packets.

Acknowledge client events:

This packet is sent in response to having received a block of events from the client.

Byte:Len	***Value (Hex)***	***Description***
0:2	E0A0	Packet type
2:1	XX	Payload length
3:X	XX..XX	Event sequence [optional]

Notes:

- If specified, the sequence number must be equal to the sequence number of the last valid received event. The payload length must also match the length of the sequence number as specified in the client event packet.
- If not specified (not recommended), the client will assume that all sent events were received by the server.

Get property:

This packet is a request to the client to respond with the value of the specified property.

Byte:Len	***Value (Hex)***	***Description***
0:2	E0B0	Packet type
2:1	XX	Payload length
3:2	XXXX	Property ID [see 'base/props.h']
5:X	XX..XX	[optional] Property retrieval arguments, as defined/required by the client.

Set property:

This packet is a command to the client to set the value of the specified property ID to the specified value. This packet may also be used for envoking an action on the client, based on the property id. The client must not send a response to these commands until specifically asked to do so by the server (via "$E000").

Byte:Len	***Value (Hex)***	***Description***
0:2	E0B1	Packet type
2:1	XX	Payload length
3:2	XXXX	Property ID [see 'base/props.h']
5:X	XX..XX	Property value

Server file upload:

This packet contains file upload session information. The server initiates a file-upload session by sending a 'Start' [0x01] packet, followed by multiple 'Data' [0x02] packets, and ending with an 'End' [0x03] packet. Out-Of-Band file transfers may also occur by specifying either 'Get' [0x31], or 'Put' [0x41] requests

Byte:Len	***Value (Hex)***	***Description***
0:2	E0C0	Packet type
2:1	XX	Payload length
3:1	XX	0x01=Start, 0x02=Data, 0x03=End, 0x31=Get, 0x41=Put
4:3	XXXXXX	File Length[Start/End/Get], File Offset[Data]
7:X	XX..XX	Filename[Start/Get/Put], Data[Data], Checksum[End]

- See 'base/upload.c' for implementation details on Start/Data/End packet types.
- The 'Get'/'Put' packet sub-types are specified relative to the client device. Thus 'Get' is a command to the client device to retrieve the named file from the server, and 'Put' is a command to the client device to send the named file to the server.

Server Error:

This packet is an indicator to the client that an error was detected in the packets received from the client. The server may, or may not, ask the client for additional information after this packet. Or, the server may simply send an "$E0FF", indicating that it will be closing the socket. Or, depending on the error, the server may simply close the socket without reporting any additional information to the client.

Byte:Len	***Value (Hex)***	***Description***
0:2	E0E0	Packet type
2:1	XX	Payload length
3:2	XXXX	Error code [see 'base/serrors.h']
5:2	XXXX	Header/type of offending received packet
7:X	XX..XX	Other data as defined by the specific error

End of transmission/session:

This packet is an indication to the client that the session will be closed.

Byte:Len	***Value (Hex)***	***Description***
0:2	E0FF	Packet type
2:1	00	Payload length [0x00]

Appendix 'E' – GPS Latitude/Longitude Encoding

When transmitting GPS coordinates between the client and server, latitude/longitude values are encoded into either 6-byte or 8-byte quantities, depending on desired resolution. A 6-byte encoded coordinate should provide accuracy to about 4+ decimal places in the latitude/logitude which provides a resolution down to about +/- 12 meters. An 8-byte encoded coordinate should provide accuracy to about 5+ decimal places in the latitude/logitude which provides a resolution down to about +/- 2 meters. Note that the choice of a coordinate encoding will still only be as accurate as the capabilities of the GPS receiver itself. For instance, choosing an 8-byte encoding resultion will not provide more accurate data if the GPS receiver itself is only capable of producing a +/- 12 meter resolution.

The following describes the algorithm used by OpenDMTP for encoding/decoding latitude/longitude values.

6-Byte Latitude/Longitude:

Encoding: (latitude/longitude each represent 'double' values)

```
#define POW2_24F   (  16777216.0) // 2^24
UInt32 encLat = ((latitude  -  90.0) * (POW2_24F / -180.0));
UInt32 encLon = ((longitude + 180.0) * (POW2_24F /  360.0));
// The encoded Lat/Lon are now in the lower 3 bytes of encLat/encLon
```

Decoding: (encLat/encLon each represent 3-byte encoded Uint32 values)

```
#define POW2_24F   (  16777216.0) // 2^24
double latitude  = ((double)encLat * (-180.0 / POW2_24F)) +  90.0;
double longitude = ((double)encLon * ( 360.0 / POW2_24F)) - 180.0;
// The decoded lat/lon are now available in latitude/longitude
// (adding 0.5 to the encoded lat/lon before decoding reduces rounding error)
```

8-Byte Latitude/Longitude

Encoding: (latitude/longitude each represent 'double' values)

```
#define POW2_32F   (4294967296.0) // 2^32
UInt32 encLat = ((latitude  -  90.0) * (POW2_32F / -180.0));
UInt32 encLon = ((longitude + 180.0) * (POW2_32F /  360.0));
// The encoded Lat/Lon are now in the 4 bytes of encLat/encLon
```

Decoding: (encLat/encLon each represent 4-byte encoded Uint32 values)

```
#define POW2_32F   (4294967296.0) // 2^32
double latitude  = ((double)encLat * (-180.0 / POW2_32F)) +  90.0;
double longitude = ((double)encLon * ( 360.0 / POW2_32F)) - 180.0;
// The decoded lat/lon are now available in latitude/longitude
// (adding 0.5 to the encoded lat/lon before decoding reduces rounding error)
```