DMC v2 COMM Protocol

The DMC v2 communications protocol is the real-time software interface between Dragonframe 4+ software and our DMC-16/DMC+. Third-party vendors may implement the same protocol on their devices to achieve real-time control via Dragonframe.

Unlike the v1 protocol, or the Arduino DFMoco protocol, the v2 protocol is binary rather than ASCII. The protocol also includes message IDs and checksums. The goal is to reduce message size and provide greater reliability.

Any reference to a motor number is 1-based. Keep this in mind, despite the fact that your code probably uses zero-based motor numbers.

In this document, the communication is from the perspective of Dragonframe. So a ‘request’ is a message from Dragonframe to the device.

The receiving buffer on the device must be at least 1048 bytes.

Document Revision History

2024-08-13 - Added real-time camera trigger options
2024-02-13 - Added optional parameters to MSG_RT_SHOOT_FRAME
2022-05-14 - Added real-time looping/ping-ponging.
2022-03-11 - Added notes about sending out position updates.
2022-02-25 - Updated MSG_MOTOR_JOG speed description.
2021-10-12 - Added errors for 'not in position' and pre- and post-roll failures.
2021-04-06 - Added aim safe distance to virtual configuration
2020-10-02 - Added list of possible capabilities to provide in HI message.
    Added PROTOCOL VERSION to end of HI message.
    Added MSG_RT_UPLOAD_MOVE_TRIGGERS message.
    Added MSG_VIRT_AIM_POINT message.
2020-01-25 - Fixed OK response code value
2018-03-20 - Updated protocol to 2.2
2018-03-10 - Changed MSG_VIRT_JOG_ON_LINE from 0x0204 to 0x0206, and changed parameters
2018-01-28 - Fixed ACK_OK value. Added MSG_RT_END
2018-01-24 - Fixed MOTOR_GET_POSITION frame time description
2017-07-14 - Initial revision for sharing.

Capabilities

A device will advertise different capabilities in response to the **MSG_HI** message.

Motor Control

If the device advertises that it supports at least one motor, Dragonframe expects it to handle basic motor control. This includes the following commands:
MSG_MOTOR_STATUS
MSG_MOTOR_MOVE
MSG_MOTOR_STOP
MSG_MOTOR_STOP_ALL
MSG_MOTOR_GET_POSITION
MSG_MOTOR_RESET_POSITION
MSG_MOTOR_JOG
MSG_MOTOR_CONFIGURE
MSG_MOTOR_SET_SPEED
MSG_MOTOR_SET_LIMITS
MSG_MOTOR_HARD_STOP

DMX Lighting

If the device advertises that it supports at least one light, Dragonframe expects it to handle the DMX lighting command:
MSG_DMX

I/O

MSG_GIO_OUT
MSG_GIO_IN
MSG_GIO_CAM

Real-time Moves

If the device advertises real-time capabilities, Dragonframe expects it to handle the following commands:
MSG_RT_UPLOAD_MOVE_BEGIN
MSG_RT_UPLOAD_MOVE_AXIS
MSG_RT_UPLOAD_MOVE_DMX (only if the device also support DMX)
MSG_RT_UPLOAD_MOVE_END

MSG_RT_POSITION_FRAME
MSG_RT_RUN_MOVE
MSG_RT_SHOOT_FRAME
MSG_RT_GO
MSG_RT_END
MSG_RT_JOG_ALL
MSG_GO_MOTION2

Virtuals

If the device advertises virtuals, Dragonframe expects it to handle the following commands:
MSG_VIRT_CONFIG
MSG_VIRT_MOVE
MSG_VIRT_STOP
MSG_VIRT_JOG
MSG_VIRT_JOG_ON_LINE
MSG_VIRT_GET_POSITION
MSG_VIRT_AIM_POINT

Message Format

Every message has a **Header**. Additionally, some messages contain a variable-length **Data** section.

All multi-byte values are sent in Little-Endian order.
A WORD is a two-byte value.
A DWORD is a four-byte value.

Section	Size	Description
Marker	2 BYTES	Start of message. Byte sequence: { ‘D’ ‘F’ }
ID	DWORD	App-defined message ID. Usually a sequence number.
Type	WORD	Message type.
Length	WORD	Length of data section, in bytes.
Data	**Length**	Depends on message type.
Checksum	WORD	Checksum bytes. Explained at the end of document.

Message Types

Most messages have different data sections depending on if the message is coming from Dragonframe or from the device. The data coming from Dragonframe is described in the **Request Data** section, the data coming from the device is described in the **Response Data** section.

MSG_FLAG_ACK [0x8000]

The device must respond to every message it receives. Some requests have specific responses that they expect. Some just expect a response with a MSG_FLAG_ACK set and a response code.

The presence of this flag means that the **Data** section will contain a response code, rather than any other expected data.

The response should use the message **Type** and **ID** from the received message.

Response Data

Section	Size	Description
Response Code	WORD	OK (0x0010) or appropriate Error Code

Response Code	Value	Description
OK	0x0010	Received, understood, and acted on message
ERR_CHECKSUM	0x0011	Checksum didn’t match
ERR_MOVING	0x0012	Can’t handle this command while the rig is moving
ERR_UNSUPPORTED	0x0013	The message type is not known or supported
ERR_RANGE	0x0014	A parameter was out of range
ERR_GENERAL	0x0015	General Error
ERR_NOT_IN_POSITION	0x0016	Rig is not in position to perform requested operation. For example, if you attempt to jog all but aren't on a frame position.
ERR_PREROLL	0x0017	A real-time move preroll failed, usually because preroll would send motors past limits.
ERR_POSTROLL	0x0018	A real-time move postroll failed, usually because a limit check failed.
ERR_SOFT_UP	0x0020	Software upper limit hit
ERR_SOFT_LOW	0x0021	Software lower limit hit
ERR_HARD_UP	0x0022	Hardware upper limit hit
ERR_HARD_LOW	0x0023	Hardware lower limit hit

MSG_HI [0x0001]

Request basic information about the device. Dragonframe always starts with this request. Also, the device should issue a MSG_HI whenever it starts up. This is helpful to Dragonframe to determine if the device had a random reset. (Some devices can maintain their USB connection while resetting the controller.)

Request Data

None

Response Data

Section	Size	Description
NAME	32 BYTES	Device name, up to 32 bytes, UTF-8 encoded. Fill with \0 characters if less than 32 bytes
FW MAJOR	BYTE	FW major number
FW MINOR	BYTE	FW minor number
FW REV	BYTE	FW revision number
MOTOR COUNT	BYTE	0-32: Number of motors the device supports
DMX COUNT	WORD	0-512: Number of DMX channels the device supports
GIO OUT COUNT	BYTE	0-32: Number of general IO output triggers (relay, logic out)
GIO INPUT COUNT	BYTE	0-32: Number of input triggers
HW LIMIT COUNT	BYTE	0-32: How many hardware limit set inputs the device has
UPLOAD FRAME COUNT	DWORD	How many frames of position/dmx data can the device hold
CAPABILITIES	DWORD	Other capabilities the device can advertise
PROTOCOL VERSION	WORD	2

Capabilities

Each bit in the capabilities parameter represents functionality the device supports.

Value	Capability
0x0001	Real-time moves (upload move / jog all / play)
0x0002	Go-motion shoot frame
0x0004	Virtual mode Boom/Swing/Track
0x0008	Virtual mode Swing/Pan
0x0010	Virtual mode Y/Swing/Track
0x0020	Virtual mode X/Y/Z
0x0040	Aim point
0x0080	Go-motion shoot frame version #2
0x0100	Couple motors
0x0200	Real-time looping or ping-pong playback
0x0400	Real-time camera trigger (video or stills)

MSG_DMX [0x0020]

Set DMX light values. The request specifies the initial channel to change, and then one or more light values.
If RAMP is set to 1, the device should ramp the light value of each channel up or down to reach the target channel, rather than switching to it immediately.

Request Data

Section	Size	Description
RAMP	BYTE	1=ramp, 0=immediate
START CHANNEL	WORD	First channel of lighting values, 1-512
LIGHT VALUE	BYTE	LIGHT VALUE for START CHANNEL
LIGHT VALUE	BYTE	LIGHT VALUE for START CHANNEL + 1
LIGHT VALUE	BYTE	LIGHT VALUE for START CHANNEL + 2

LIGHT VALUE	BYTE	etc

Response Data

None (MSG_FLAG_ACK + Response Code)

MSG_GIO_OUT [0x0021]

Set the state for general I/O output triggers.

Request Data

Section	Size	Description
TRIGGERS	DWORD	General I/O output trigger values

Response Data

None (FLAG_ACK + Response Code)

MSG_GIO_IN [0x0022]

Request the state for general I/O input triggers. Dragonframe may request the input state with this message. The device should also send this message, unsolicited, any time the inputs change. (Of course, the device should use some hysteresis to ensure the inputs don't change frantically.)

Request Data

None (MSG_FLAG_ACK + Response Code)

Response Data

Section	Size	Description
TRIGGERS	DWORD	General I/O input trigger values

MSG_GIO_CAM [0x0023]

Test the camera outputs.

Request Data

Section	Size	Description

TRIGGERS	DWORD	Camera flags

GIO_CAM_SHUTTER [0x0001]
GIO_CAM_METER [0x0002]

Response Data

None (FLAG_ACK + Response Code)

MSG_MOTOR_STATUS [0x0030]

Request the motor (and dmx) status. This is only a status of whether something is moving/adjusting.

Request Data

None

Response Data

Section	Size	Description
MOTOR STATUS	DWORD	Bit 1=motor moving, 0=not moving
DMX STATUS	BYTE	1=DMX adjusting, 0=Not adjusting

MSG_MOTOR_MOVE [0x0031]

Instruct the device to move a motor to a new position.

Request Data

Section	Size	Description
MOTOR	BYTE	Motor number, 1-MOTOR_COUNT
POSITION	DWORD	Signed int32 - the motor step position

Response Data

Section	Size	Description
MOTOR STATUS	BYTE	1=motor moving, 0=not moving

MSG_MOTOR_STOP [0x0032]

Instruct the device to stop a motor.

Request Data

Section	Size	Description
MOTOR	BYTE	Motor number, 1-MOTOR_COUNT

Response Data

None (MSG_FLAG_ACK + Response Code)

MSG_MOTOR_STOP_ALL [0x0033]

Instruct the device to stop all motors. The device should decelerate. If the device gets a second STOP_ALL soon after a first, it should HARD STOP. Meaning it should decelerate faster than normal. (If a controller may move large/heavy equipment, it should always decelerate rather than simply stop sending signals.)

Request Data

Section	Size	Description
Flags	DWORD	(Optional) 0x01 means silent (do not flash any warnings on device)

Response Data

None (MSG_FLAG_ACK + Response Code)

MSG_MOTOR_GET_POSITION [0x0034]

Request the motor positions and move ‘time’.
The device should send these out, about every 0.10 seconds, if any of the motors are moving, to keep Dragonframe informed of the positions.

Request Data

NONE

Response Data

Section	Size	Description
MOVE TIME	DWORD	Current position in move (if playing back), in thousandths of a frame. (Frame 2 is ‘2000’, e.g.)
MOTOR-1 POS	DWORD	Motor 1 position
MOTOR-2 POS	DWORD	Motor 2 position
...	...	...
MOTOR-N POS	DWORD	Final motor position

MSG_MOTOR_RESET_POSITION [0x0035]

Instruct the device to reset a motor position to a new value. This does not move the motor. After changing the internal position and responding to this message, the device should send a MSG_MOTOR_GET_POSITION message to confirm the new position.

Request Data

Section	Size	Description
MOTOR	BYTE	Motor number, 1-MOTOR_COUNT
POSITION	DWORD	New position

Response Data

None (MSG_FLAG_ACK + Response Code)

MSG_MOTOR_JOG [0x0036]

Instruct the device to jog/inch a motor.
A speed of 1 means the device should move very slowly, for precise adjustments.
A speed of 10,000 means the device should jog using the max velocity and acceleration.
A value in between 1 and 10,000 should provide an appropriate speed between 1% and 100% of the jog speed.

Request Data

Section	Size	Description
MOTOR	BYTE	Motor number, 1-MOTOR_COUNT
SPEED	WORD	1=inching, 10000=jog at max velocity
DESTINATION	DWORD	Step position of target position (limited to min/max of signed int32)

Response Data

None (MSG_FLAG_ACK + Response Code)

MSG_MOTOR_CONFIGURE [0x0037]

Configure the motor.

Request Data

Section	Size	Description
MOTOR	BYTE	Motor number, 1-MOTOR_COUNT
FLAGS	BYTE	0x01 - motor enabled. 0x02 - blur enabled

Response Data

None (MSG_FLAG_ACK + Response Code)

MSG_MOTOR_SET_SPEED [0x0038]

Set the motor speed and acceleration.

Request Data

Section	Size	Description
MOTOR	BYTE	Motor number, 1-MOTOR_COUNT
MAX VELOCITY	DWORD	The maximum steps per second the motor can go
MAX ACCEL	DWORD	The maximum acceleration, in steps/second/second

Response Data

None (MSG_FLAG_ACK + Response Code)

MSG_MOTOR_SET_LIMITS [0x0039]

Set the motor software and hardware limits.

Request Data

Section	Size	Description
MOTOR	BYTE	Motor number, 1-MOTOR_COUNT
LOWER ENABLE	BYTE	0x01 enabled, 0x00 not
LOWER LIMIT	DWORD	The lower software step limit
UPPER ENABLE	BYTE	0x01 enabled, 0x00 not
UPPER LIMIT	DWORD	The upper software step limit
HW SET	BYTE	Hardware limit set, or 0x00 if none set Flag 0x80 swaps high/low

Response Data

None (MSG_FLAG_ACK + Response Code)

MSG_MOTOR_HARD_STOP [0x003A]

The device can send this when a limit has been hit, or if the device's "emergency stop" button has been pressed.

Request Data

None

Response Data

Section	Size	Description
REASON	BYTE	0=E-Stop/General, 1=Upper Limit, 2=Lower Limit, 100=Engine Exception
MOTOR	BYTE	Motor number, 1-MOTOR_COUNT (only present if REASON is 1 or 2)

MSG_RT_UPLOAD_MOVE_BEGIN [0x0100]

Initial message for uploading move data. Any previously loaded move will be cleared.

Request Data

Section	Size	Description
START FRAME	DWORD	The starting frame of the move data
END FRAME	DWORD	The ending frame of the move data

Response Data

None (MSG_FLAG_ACK + Response Code)

MSG_RT_UPLOAD_MOVE_AXIS [0x0101]

Send a section of axis frame data.

Request Data

Section	Size	Description
MOTOR	BYTE	The motor

START INDEX	DWORD	The starting index of the move data (start at zero). If the high bit is set (0x80000000), then this is the _last_ set of POSITION data for this channel. That means the device must repeat the final value until the end of the uploaded move frame range (specified with UPLOAD_MOVE_BEGIN)
POSITION	DWORD	Position
...	...	...
POSITION	DWORD	Final position of this section of move data

Response Data

None (MSG_FLAG_ACK + Response Code)

MSG_RT_UPLOAD_MOVE_DMX [0x0102]

Send a section of DMX frame data.

Request Data

Section	Size	Description
CHANNEL	WORD	The DMX channel
START INDEX	DWORD	The starting index of the move data (start at zero). If the high bit is set (0x80000000), then this is the _last_ set of DMX data for this channel. That means the device must repeat the final value until the end of the uploaded move frame range (specified with UPLOAD_MOVE_BEGIN)
LEVEL	BYTE	DMX light level
...	...	...
LEVEL	BYTE	Final level of this section of DMX data

Response Data

None (MSG_FLAG_ACK + Response Code)

MSG_RT_UPLOAD_MOVE_TRIGGERS [0x0104]

Introduced in Protocol: 2
Send a section of trigger data.
Only sent if the DMX program has triggers.
Only sends frame/value pairs where one or more triggers is on.

Request Data

Section	Size	Description
MASK	DWORD	The trigger mask. Should be the same every time.
		Repeat the following frame/value pair as long as needed
FRAME	DWORD	Frame index (frame - start)
VALUES	DWORD	Trigger values
FRAME	DWORD	Frame index (frame - start)
VALUES	DWORD	Trigger values

Response Data

None (MSG_FLAG_ACK + Response Code)

MSG_RT_UPLOAD_MOVE_END [0x0103]

Final message for uploading move data.

Request Data

None

Response Data

None (MSG_FLAG_ACK + Response Code)

MSG_RT_POSITION_FRAME [0x0110]

Instructs the device to send all motors (and DMX if applicable) to the uploaded frame position.

Request Data

Section	Size	Description
FRAME	DWORD	The frame to send all motors to

Response Data

None (MSG_FLAG_ACK + Response Code)

MSG_RT_RUN_MOVE [0x0111]

Prepares the device to run a section of the move live.
The rig must move into a pre-roll position that it calculates, so that it can accelerate into full speed after “pre-roll time”.
Once the rig moves into pre-roll position, it must wait for a MSG_RT_GO to begin the move.

Request Data

Section	Size	Description
FPS	DWORD	Speed in FPS * 1000
START FRAME	DWORD	Start frame to play
END FRAME	DWORD	End frame to play
PRE-ROLL TIME	DWORD	Pre-roll time, in ms
POST-ROLL TIME	DWORD	Post-roll time, in ms
SYNC DMX	BYTE	1=Play back upload DMX, 0=don’t play DMX
BLOOP LOCATION	DWORD	GIO Outputs for “bloop” signal
BLOOP DMX CHANNEL	WORD	0=None, 1-512 = a channel for the BLOOP
BLOOP TIME	WORD	Bloop time (approx) in ms
FLAGS	WORD	Only set if capability 0x0200 (real-time looping) or 0x0400 (real-time camera) are set. 0x01 is ping-pong 0x02 is loop

		0x10 is camera trigger at start/stop of move (video) 0x20 is camera trigger for every frame (stills)
CAMERA OPEN ANGLE	WORD	Only used if camera trigger for every frame, 0-360 degrees
CAMERA CLOSE ANGLE	WORD	Only used if camera trigger for every frame, 0-360 degrees

Response Data

None (MSG_FLAG_ACK + Response Code)

MSG_RT_SHOOT_FRAME [0x0112]

Set up a blur-motion frame capture

Request Data

Section	Size	Description
FRAME	DWORD	A frame in the uploaded move
DIRECTION	BYTE	1=forward, 0=backward
EXPOSURE TIME	DWORD	Exposure Time in ms
BLUR PERCENT	WORD	Percentage * 10 (500 = 50%)
MOTOR #1	BYTE	Motor number
MOTOR #1 POS A	DWORD	Motor position at angle 0
MOTOR #1 POS B	DWORD	Motor position at angle 360
MOTOR #2	BYTE	Motor number
MOTOR #2 POS A	DWORD	Motor position at angle 0
MOTOR #2 POS B	DWORD	Motor position at angle 360
...		Add motors to blur, as needed

Response Data

None (MSG_FLAG_ACK + Response Code)

MSG_RT_SHOOT_FRAME2 [0x0115]

Set up a blur-motion frame capture (alternate version)
The sets of motor numbers and positions are optional. Each motor provided will be part of the blur.

Request Data

Section	Size	Description
FRAME	DWORD	A frame in the uploaded move
EXPOSURE TIME	DWORD	Exposure Time in ms
OPEN ANGLE	WORD	Open shutter time, expressed as angle
CLOSE ANGLE	WORD	Close shutter time, expressed as angle
MOTOR #1	BYTE	Motor number
MOTOR #1 POS A	DWORD	Motor position at angle 0
MOTOR #1 POS B	DWORD	Motor position at angle 360
MOTOR #2	BYTE	Motor number
MOTOR #2 POS A	DWORD	Motor position at angle 0
MOTOR #2 POS B	DWORD	Motor position at angle 360
...		Add motors to blur, as needed

Response Data

None (MSG_FLAG_ACK + Response Code)

MSG_RT_GO [0x0113]

After a MSG_RT_RUN_MOVE or MSG_RT_SHOOT_FRAME the device may move the motors into position. When the device has stopped moving, a MSG_RT_GO command will perform the run or shoot of the frame.

If the device is running a live move (configured via MSG_RT_RUN_MOVE), it must send a MSG_MOTOR_GET_POSITION each time it reaches a new frame in the move. (If it is slightly after, that is ok.)

Request Data

None

Response Data

None (MSG_FLAG_ACK + Response Code)

MSG_RT_END [0x0114]

After a MSG_RT_RUN_MOVE or MSG_RT_SHOOT_FRAME has finished shooting, the device sends MSG_RT_END to notify Dragonframe that the process is over.

Request Data

None

Response Data

Note

MSG_RT_JOG_ALL [0x0120]

Jog all motors towards the destination frame.
The rig must already be on a frame position that is uploaded to the device.

Request Data

Section	Size	Description
FPS	DWORD	Speed in FPS * 1000
DESTINATION	DWORD	A frame number in the upload move

Response Data

None (MSG_FLAG_ACK + Response Code)

MSG_RT_STOP_LOOP [0x0116]

If the device supports 'real-time looping' capability, it must support this command.
The currently running loop/ping-pong must stop after it reaches the next end point.

Request Data

None

Response Data

None (MSG_FLAG_ACK + Response Code)

MSG_VIRT_CONFIG [0x0200]

Configure virtuals.

Request Data

Section	Size	Description
TYPE	BYTE	0=none 1=boom-swing-track 2=swing-pan, 3=y-swing-track 4=x-y-z
depends	depends	Depends on type

TYPE = 0 (None)
No further request data.

TYPE = 1 (Boom-Swing-Track)

Section	Size	Description
BOOM motor	DWORD	BOOM motor (up/down)
BOOM spu	DWORD	BOOM steps per unit
BOOM pos	DWORD	BOOM position (in units) * 100000
SWING motor	DWORD	SWING motor (up/down)
SWING spu	DWORD	SWING steps per unit
SWING pos	DWORD	SWING position (in units) * 100000
TRACK motor	DWORD	TRACK motor (up/down)

TRACK spu	DWORD	TRACK steps per unit
TRACK pos	DWORD	TRACK position (in units) * 100000
PAN motor	DWORD	PAN motor (up/down)
PAN spu	DWORD	PAN steps per unit
PAN pos	DWORD	PAN position (in units) * 100000
TILT motor	DWORD	TILT motor (up/down)
TILT spu	DWORD	TILT steps per unit
TILT pos	DWORD	TILT position (in units) * 100000
ROLL motor	DWORD	ROLL motor (up/down)
ROLL spu	DWORD	ROLL steps per unit
ROLL pos	DWORD	ROLL position (in units) * 100000
BOOM LENGTH	DWORD	Boom length * 1000
BOOM EXT	DWORD	Boom extension length * 1000
NODAL OFF X	DWORD	Nodal offset for x * 1000
NODAL OFF Y	DWORD	Nodal offset for y * 1000
NODAL OFF Z	DWORD	Nodal offset for z * 1000
Boom Compensation Table	DWORD * 121	Optional Boom Compensation Data. Position of Boom arm at -60, -59, … 0, 1, 2, … 59, 60 degrees
SAFE DISTANCE	DWORD	Safe distance * 1000 (optional)

TYPE = 2 (Swing-Pan)

Section	Size	Description
SWING motor	DWORD	SWING motor (up/down)
SWING spu	DWORD	SWING steps per unit
PAN motor	DWORD	PAN motor (up/down)

PAN spu	DWORD	PAN steps per unit

Response Data

None (MSG_FLAG_ACK + Response Code)

MSG_VIRT_MOVE [0x0201]

Send the virtual motor to a new position.

Request Data

Section	Size	Description
VIRTUAL MOTOR	BYTE	Virtual motor number
POSITION	DWORD	Target position * 100000

Response Data

None (MSG_FLAG_ACK + Response Code)

MSG_VIRT_STOP [0x0202]

Stop the virtual motor.

Request Data

Section	Size	Description
VIRTUAL MOTOR	BYTE	Virtual motor number

Response Data

None (MSG_FLAG_ACK + Response Code)

MSG_VIRT_JOG [0x0203]

Instruct the device to jog/inch a virtual motor.
A speed of 1 means the device should move very slowly, for precise adjustments.
A speed of 10,000 means the device should jog using the max velocity and acceleration.
For now, those are the only speeds sent.
In the future, we may send intermediate values to signify a target % of the max velocity.

Request Data

Section	Size	Description
MOTOR	BYTE	Motor number, 1-MOTOR_COUNT
SPEED	WORD	1=inching, 10000=jog at max velocity
DESTINATION	DWORD	Step position of target position (limited to min/max of signed int32)

Response Data

None (MSG_FLAG_ACK + Response Code)

MSG_VIRT_JOG_ON_LINE [0x0206]

Instruct the device to jog along the direction of the camera.
Speed is -10000->10000, with 1 being the slowest and 10000 being the fastest.

Request Data

Section	Size	Description
AXIS	BYTE	0=X, 1=Y, 2=Z (camera line), 3=PAN, 4=TILT
SPEED	WORD	1=inching, 10000=jog at max velocity

Response Data

None (MSG_FLAG_ACK + Response Code)

MSG_VIRT_GET_POSITION [0x0205]

Request the virtual motor positions.

Request Data

NONE

Response Data

Section	Size	Description
vTrack POS	DWORD	Virtual track position * 100000
vEW POS	DWORD	Virtual east/west position * 100000
vNS POS	DWORD	Virtual north/south position * 100000
vPan POS	DWORD	Virtual pan position * 100000
vTilt POS	DWORD	Virtual tilt position * 100000
vRoll POS	DWORD	Virtual roll position * 100000
Aim pt enabled	BYTE	This and below are only if device supports aim point
aim-x	DWORD	X * 1000
aim-y	DWORD	Y * 1000
aim-z	DWORD	Z * 1000

MSG_VIRT_AIM_POINT [0x0207]

Introduced in Protocol: 2
Configure aim point.

Request Data

Section	Size	Description
ENABLE	BYTE	Enable/disable aim point
AIM-X	DWORD	Aim x position * 1000
AIM-Y	DWORD	Aim y position * 1000

AIM-Z	DWORD	Aim z position * 1000

Response Data

The response contains the same data. The message can also be sent from the DMC at any point to confirm the aim point configuration.

Section	Size	Description
ENABLE	BYTE	Enable/disable aim point
AIM	DWORD	Aim x position * 1000
AIM-Y	DWORD	Aim y position * 1000
AIM-Z	DWORD	Aim z position * 1000

Computing and Validating the Checksum

The protocol uses a Fletcher-16 checksum with K=8 and modulus of 255.
https://en.wikipedia.org/wiki/Fletcher%27s_checksum#Fletcher-16
Further, outgoing messages add check bytes, so that the final computed checksum will be zero on the receiver end.

Example C code to compute checksum:

```
uint16_t computeChecksum(uint8_t * data, int bytes)
{
    alt_u16 sum1 = 0, sum2 = 0;
    size_t tlen;

    while (bytes)
    {
        tlen = ((bytes >= 20) ? 20 : bytes);
        bytes -= tlen;
        do
        {
            sum2 += sum1 += *data++;
            tlen--;
        } while (tlen);
        sum1 %= 0xff;
        sum2 %= 0xff;
    }
    return (sum2 << 8) | sum1;
}
```

For received messages, this checksum should be zero. If it is not the device should respond with response code **ACK_ERR_CHECKSUM**.

For outgoing messages, the checksum should be computed with the above function, and then the check bytes should be calculated as follows:

```
// assuming our outbound message is stored in 'messageBuffer'
// uint8_t * messageBuffer;
// int messageLength;

alt_u16 csum = computeChecksum(messageBuffer, messageLength);
alt_u8 c0,c1,f0,f1;
f0 = csum & 0xff;
f1 = (csum >> 8) & 0xff;
c0 = 0xff - ((f0 + f1) % 0xff);
```

```
c1 = 0xff - ((f0 + c0) % 0xff);

// add check bytes to end of message
messageBuffer[messageLength++] = c0;
messageBuffer[messageLength++] = c1;
```