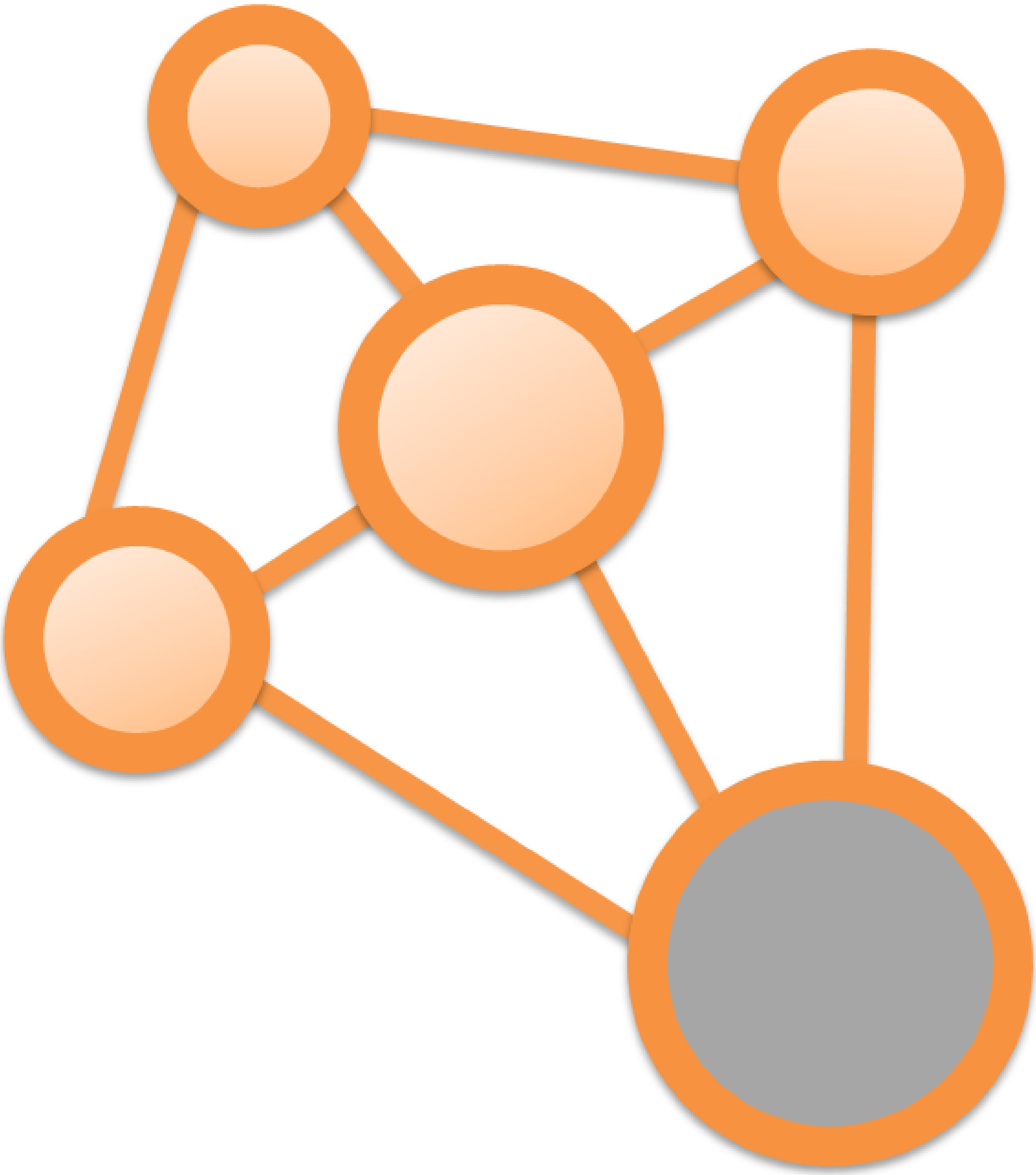
MVN Analyze/Animate real-time network streaming

Protocol Specification

Document MV0305P, Revision J, November 2017



Xsens Technologies B.V.

Pantheon 6a
P.O. Box 559
7500 AN Enschede
The Netherlands

phone +31 (0)88 973 67 00
fax +31 (0)88 973 67 01
e-mail info@xsens.com
internet www.xsens.com

Xsens North America, Inc.

10557 Jefferson Blvd,
Suite C
CA-90232 Culver City
USA

phone 310-481-1800
fax 310-416-9044
e-mail info@xsens.com
internet www.xsens.com

Revisions

Revision	Date	By	Changes
E	February 2012	DOS	Updated for release MVN Analyze/Animate 3.3
F	June 2013	AOD	Updated for release MVN Analyze/Animate 3.5
G	December 2013	CMO	Updated for release MVN Analyze/Animate 3.5.2
H	October 2014	JMU	Updated for release MVN Analyze/Animate 4.0 (new HW + support for TCP protocol)
I	November 2014	PVR	Indicate more clearly which data types are used by MotionBuilder, Maya and Unity3D
J	February 2015	JMU	Updated for MVN Analyze/Animate 4.1: Additional datagrams for expanded network streaming defined
K	November 2017	HBE	Updated naming for MVN 2018

Table of Contents

1 Introduction

MVN Analyze/Animate, developed by Xsens, is a tool to capture and compute the 6DOF motion data of an inertial sensor-driven system. It allows the export of the data to third party applications such as Motion Builder, making the data available to drive rigged characters in e.g. animations. The data transfer to other applications is primarily file based when using MVN Analyze/Animate. With the XME API (SDK) there are many other options.

In many situations it is attractive to keep the ease of use of MVN Analyze/Animate, while receiving the motion capture data in real-time in another application, even on another PC possibly physically remote from the MVN system.

This document defines a network protocol specification for this purpose. It describes the transport medium, the given data and the datagrams to be sent and received over the network, as well as the control sequences the server and clients will use to communicate states and requests during the sessions. The network communication is mainly required to be fast/real-time, other quality criteria are secondary.

This document describes MVN Analyze/Animate Real-time Network Streaming. The streaming feature enables computers that run MVN Analyze/Animate to stream the captured data over a network to other client computers.

1.1 *Perceived Usage*

1.1.1 Usage in real-time previsualization and simulation VR setups

Many software packages (e.g. MotionBuilder) and experimental VR rigs use single computers to do specific processing and hardware interfacing tasks, such as driving motion platforms, real-time rendering to a screen, or interfacing with a motion capture device. In this scenario, a PC set up with MVN Analyze/Animate could service one (or more) motion captured persons. This requires immediate, regularly timed delivery of state (pose) packets. The UDP protocol is most suitable for this task because it delivers packets without congestion control and dropped packet checking. MVN Analyze/Animate real-time network streaming protocol is based on UDP and is specified in this document.

To support scenarios like this for usage with 3rd party tools as a client application, Xsens has developed several plug-ins. MVN Analyze/Animate plug-ins are available for **Autodesk Motion Builder**, **Autodesk Maya** and **Unity3D**. These tools use protocols specified in this document to receive motion capture data in real-time.

The client side plug-ins for MotionBuilder and Maya can be requested and purchased separately at Xsens.
The Unity3D plug-in is available for free at: https://www.assetstore.unity3d.com/en/#!/content/11338 (Version: 1.0 (Apr 25, 2014), Size: 1.6 MB, this requires Unity 4.0.1 or higher)

1.1.2 Network Streamer and Network monitor

To send motions from MVN Analyze/Animate, go to Options > Preferences > Miscellaneous > Network Streamer and choose the desired protocol. The motion can also be received by MVN Analyze/Animate go to File > open Network Monitor. The Network Monitor will also show the local axis for each segment. Also note that the red triangle in the origin is representing the x-axis.

1.1.3 Usage in multi-person or other complex motion capture setups

In roll-your-own motion capture setups, often additional data is captured. An example could be medical data, or data gloves. Another setup might capture multiple subjects at once. The TCP protocol would be most suitable for this task as this protocol guarantees that the data stream is completely sent, potentially at the expense of near real-time delivery. However UDP also suffices in a well-designed network setup as there will be nearly no, or very little, packet loss.

Advantages for motion capture setup builders include:

- Not necessary to interface with XME API (SDK).
- Processing CPU time required for inertial motion capture is done on a separate PC, freeing up resources for other processing;
- Calibration and real-time pre-viewing (e.g. for assessment of motion capture quality) can be done on the processing PC using MVN Analyze/Animate itself.

2 Transport Medium

2.1 Network Environment

The network environment will be assumed to be a local 100 Mbit Ethernet network, larger network topologies are not considered and can be covered by file transfer of the already given file export functionality or later extensions to the network protocol. Thus, few packet loss or data corruption during transfer is to be expected, as well as constant connectivity.

2.2 Network Protocol

Network communication uses a protocol stack, thus the streaming protocol will be implemented on top of a given set of protocols already available for the network clients. In this case, the layers to build upon are IP and UDP (or TCP, which is also supported). **IP** (Internet Protocol, RFC 791) is the network layer protocol used in Ethernet networks and defines the source and destination of the packets within the network. Upon this, **UDP** (User Datagram Protocol, RFC 768) is used to encapsulate the data. The **UDP** Protocol is unidirectional, and contrary to **TCP** (Transmission Control Protocol, RFC 793) it is stateless and does not require the receiver to answer incoming packets. This allows greater speed.

2.3 Default Port

The default Port to be used on the network is 9763. This Port is derived from the XME API (9=X, M=6, E=3). MVN Analyze/Animate server will default to this Port.

It is of course possible to define an arbitrary Port if needed.

2.4 Datagram

The motion capture data is sampled and sent at regular time intervals for which the length depends upon the configuration of MVN Analyze/Animate. Common sampling rates lie between 60 and 240 Hertz. The update rate of the real-time network stream can be modified separately. The data content in the datagram is defined by the specific protocol set, but basically, the positions and rotation of all segments of the body at a sampling instance are sent away as one or more UDP datagrams.

Each datagram starts with a 24-byte header followed by a variable number of bytes for each body segment, depending on the selected data protocol. All data is sent in 'network byte order', which corresponds to big-endian notation.

Framed text indicates items that are sent as part of the datagram.

2.4.1 Header

The header contains the type of the data and some identification information, so the receiving end can apply it to the right target.

Datagram header

6 bytes ID String
4 bytes sample counter
1 byte datagram counter
1 byte number of items
4 bytes time code
1 byte character ID
7 bytes reserved for future use

2.4.1.1 ID String

The so-called ID String is an ASCII string which consists of 6 characters (not terminated by a null character). It serves to unambiguously identify the UDP datagram as containing motion data of the format according to this specification. Since the values in the string are characters, this string is not converted to a big-endian notation, but the first byte is simply the first character, etc.

These are the ASCII and hexadecimal byte values of the ID String:

ASCII	M	X	T	P	0	1
Hex	4D	58	54	50	30	31

M: M for MVN
X: X for Xsens
T: T for Transfer
P: P for Protocol

##: Message type. The first digit determines what kind of packet this is and the second digit determines the format of the data in the packet

Message type	Description
01	Pose data (**Euler**) ← MotionBuilder +Maya • Absolute position and orientation (Euler) of segments • Y-Up, right-handed • This type is used by the Motion Builder + Maya plug-in • *Supported by MVN Analyze/Animate network monitor*
02	Pose data (**Quaternion**) ← MVN Analyze/Animate Network Monitor • Absolute position and orientation (Quaternion) of segments • Default mode Z-Up, right-handed or Y-Up • *Supported by MVN Analyze/Animate network monitor*
03	Pose data (**Positions only, MVN Optical marker set 1**) • Positions of selected defined points (simulating optical markers), typically 38-46 points. Multiple data sets are available. • This datagram is used by the Motion Builder plug-in v1.0. • ***Partially*** *supported by MVN Analyze/Animate network monitor. MVN Analyze/Animate has a limited ability to re-integrate these marker positions into a character. The segment orientations will not be updated. Therefore, when* <u>*only this datagram*</u> *is received, the resulting character can appear incorrect.*
04	Deprecated: MotionGrid Tag data
05	Pose data (**Unity3D**) • Relative position and orientation (Quaternion) of segments • Uses alternative segment order • Left-handed for Unity3D protocol • *Supported by MVN Analyze/Animate network monitor*
10	Deprecated, use 13: Character information → scale information
11	Deprecated, use 13: Character information → prop information
12	Character information → meta data • name of the character • MVN character ID (BodyPack or Awinda Station ID) • << more can be added later >> • *Supported by MVN Analyze/Animate network monitor*

Message type	Description
13	Character information → scaling information, including prop and null-pose *Supported by MVN Analyze/Animate network monitor*
20	Joint Angle data • Joint definition and angles • ***NOT*** *supported by MVN Analyze/Animate network monitor.*
21	Linear Segment Kinematics • Absolute segment position, velocity and acceleration • ***Partially*** *supported by MVN Analyze/Animate network monitor. MVN Analyze/Animate has a limited ability to re-integrate this data into a character. The segment orientations will not be updated. Therefore, when <u>only this datagram</u> is received, the resulting character can appear incorrect.*
22	Angular Segment Kinematics • Absolute segment orientation, angular velocity and angular acceleration • ***Partially*** *supported by MVN Analyze/Animate network monitor. MVN Analyze/Animate has a limited ability to re-integrate this data into a character. The segment positions will not be updated. Therefore, when <u>only this datagram</u> is received, the resulting character can appear incorrect.*
23	Motion Tracker Kinematics • Absolute sensor orientation and free acceleration • Sensor-local acceleration, angular velocity and magnetic field • ***NOT*** *supported by MVN Analyze/Animate network monitor.*
24	Center of Mass • Absolute position of center of mass • ***NOT*** *supported by MVN Analyze/Animate network monitor.*
25	Time Code • Time code string • ***NOT*** *supported by MVN Analyze/Animate network monitor.*

Please note that the message type is sent as a string, not as a number, so message type “03” is sent as hex code 0x30 0x33, not as 0x00 0x03.

2.4.1.2 Sample Counter

The sample counter is a 32-bit unsigned integer value which is incremented by one, each time a new set of motion tracker data is sampled and sent away. Note that the sample counter is not to be interpreted as a time code, since the sender may skip frames.

2.4.1.3 Datagram Counter

The size of a UDP datagram is usually limited by the MTU (maximum transmission unit, approx. 1500 bytes) of the underlying Ethernet network. In nearly all cases the entire motion data that was collected at one sampling instance will fit into a single UDP datagram. However, if the amount of motion data becomes too large then the data is split up into several datagrams.

If motion data is split up into several datagrams then the datagrams receive index numbers starting at zero. The datagram counter is a 7-bit unsigned integer value which stores this index number. The most significant bit of the datagram counter byte is used to signal that this datagram is the last one belonging to that sampling instance. For example, if motion data is split up into three datagrams then their datagram counters will have the values 0, 1 and 0x82 (hexadecimal). If all data fits into one UDP datagram (the usual case) then the datagram counter will be equal to 0x80 (hexadecimal).
The sample counter mentioned above can be used to identify which datagrams belong to the same sampling instance because they must all carry the same sample counter value but different datagram

counters. This also means that the combination of sample counter and datagram counter values is unique for each UDP datagram containing (part of the) motion data.

NOTE: For practical purposes this will not be an issue with the MVN streaming protocol. If problems are encountered, check your MTU settings.

2.4.1.4 Number of items

The number of items is stored as an 8-bit unsigned integer value. This number indicates the number of segments or points that are contained in the packet. Note that this number is not necessarily equal to the total number of motion trackers that were captured at the sampling instance if the motion capture data was split up into several datagrams. This number may instead be used to verify that the entire UDP datagram has been fully received by calculating the expected size of the datagram and comparing it to the actual size of the datagram.

2.4.1.5 Time code

MVN Analyze/Animate contains a clock which starts running at the start of a recording. The clock measures the elapsed time in milliseconds. Whenever new captured data is sampled the current value of the clock is sampled as well and is stored inside the datagram(s) as a 32-bit unsigned integer value representing a time code.

2.4.1.6 Character ID

MVN Analyze/Animate now supports multiple characters in one viewport. This byte specifies to which character the data belongs. In a single-character setup this value will always be 0. In multi-character cases, they will *usually* be incremental. However, especially during live streaming, one of the characters may disconnect and stop sending data while others will continue, so the receiver should be able to handle this.
Each character will send its own full packet.

2.4.1.7 Reserved bytes for future use

The left-over bytes at the end of the datagram header are reserved for future versions of this protocol.

2.5 Pose data

2.5.1 Segment data Euler (type 01)

This protocol was originally developed and optimized for the MotionBuilder and Maya plug-in.

Information about each segment is sent as follows.

```
4 bytes segment ID See 2.5.9
4 bytes x–coordinate of segment position
4 bytes y–coordinate of segment position
4 bytes z–coordinate of segment position
4 bytes x rotation –coordinate of segment rotation
4 bytes y rotation –coordinate of segment rotation
4 bytes z rotation –coordinate of segment rotation
```

Total: 28 bytes per segment

The coordinates use a Y-Up, right-handed coordinate system for Euler protocol.

The number of segments recorded will be sent, followed by the amount of props used, if any. Maximum number of segments is 23, for full body and maximum number of props is 4.

2.5.2 Segment data quaternion (type 02)

This protocol reflects the internal format of MVN Analyze/Animate.

Information about each segment is sent as follows.

```
4 bytes segment ID See 2.5.9
4 bytes x–coordinate of segment position
4 bytes y–coordinate of segment position
4 bytes z–coordinate of segment position
4 bytes q1 rotation – segment rotation quaternion component 1 (re)
4 bytes q2 rotation – segment rotation quaternion component 1 (i)
4 bytes q3 rotation – segment rotation quaternion component 1 (j)
4 bytes q4 rotation – segment rotation quaternion component 1 (k)
```

Total: 32 bytes per segment

The coordinates use a Z-Up, right-handed coordinate system.

The number of segments recorded will be sent, followed by the amount of props used, if any. Maximum number of segments is 23, for full body and maximum number of props is 4.

2.5.3 Point position data (type 03)

Information about each point is sent as follows.
This data type is intended to emulate a Virtual (optical) Marker Set.

```
4 bytes point ID
this is 100x the segment ID + the point ID for a marker
this is the tagId for a tag
4 bytes x–coordinate of point position
4 bytes y–coordinate of point position
4 bytes z–coordinate of point position
```

Total: 16 bytes per point

The coordinates use a Y-Up, right-handed coordinate system.

After the points all MotionGrid tags assigned to the character will be sent, using the same position format as the markers.

2.5.4 MotionGrid Tag data (type 04)

This message has become deprecated.

2.5.5 Segment data Unity3D (type 05)

Information about each segment is sent as follows.

```
4 bytes segment ID See 2.5.9
4 bytes x–coordinate of segment position
4 bytes y–coordinate of segment position
4 bytes z–coordinate of segment position
4 bytes q1 rotation – segment rotation quaternion component 1 (re)
4 bytes q2 rotation – segment rotation quaternion component 1 (i)
4 bytes q3 rotation – segment rotation quaternion component 1 (j)
4 bytes q4 rotation – segment rotation quaternion component 1 (k)
```

Total: 32 bytes per segment.

The pelvis segment uses global positions and rotation, while the other segments only use local rotation and relative positions. Segments follow pelvis position based on the character model hierarchy within Unity3D.

Unity3D mode uses quaternion data, where the coordinates use a Y-Up, left-handed coordinate system.

A total of 23 segments will be sent. Props are not supported.

2.5.6 Position

The position of a captured segment is always stored as a 3D vector composed of three 32-bit float values. The unit is cm.

2.5.7 Rotation (Euler)

The rotation of a captured segment in the Euler representation is always stored as a 3D vector composed of three 32-bit float values. The unit is degrees.

2.5.8 Rotation (Quaternion)

The rotation of a captured segment in the Quaternion representation is always stored as a 4D vector composed of four 32-bit float values. The quaternion is always normalized, but not necessarily positive-definite.

2.5.9 Segment ID

The IDs of the segments are listed in paragraph 3.1. The segment ID is sent as a normal 4-byte integer.

2.5.10 Point ID

Note that since many more options have been added to the streamed data of MVN Analyze/Animate 4.1, the following section contains new information.

The ID of a point depends on the ID of the segment it is attached to and the local ID it has in the segment. These local IDs are documented in the MVN User Manual. The ID is sent as a 4-byte integer, defined as 256 * segment ID + local point ID.

Example:
The Sacrum point on the Pelvis segment has local ID 13, and the Pelvis has ID 1, so the ID of the point is sent as 256*1 + 13 = 269.

2.5.11 Float and integer values over the network

All integer values mentioned above are stored in big-endian byte order inside the UDP datagrams with the function htonl() into the network by MVN Analyze/Animate and ntohl() out in the client. In other words: the most significant byte (MSB) is stored first. This is the same byte order that is used for other Internet protocols, so standard conversion functions should be available on all computer systems.

2.5.12 String values over the network

Strings are utf-8 encoded. They are preceded by the size of the string as a 32-bit signed integer and NOT 0-terminated.

2.6 Character information

2.6.1 Scale information (type 10)

This message has become deprecated. It is superseded by message 13.

2.6.2 Prop information (type 11)

This message has become deprecated. It is superseded by message 13.

2.6.3 Meta data (type 12)

This packet contains some meta-data about the character. This is in a tagged format, each tag is formatted as “tagname:” and each tagline is terminated by a newline. Each value is a string that can be interpreted in its own way.
Defined tags are:
name: contains the name as displayed in MVN Analyze/Animate
xmid: contains the BodyPack/Awinda-station ID as shown in MVN Analyze/Animate
color: contains the color of the character as used in MVN Analyze/Animate, the format is hex RRGGBB

More tags may be added later, so any implementation should be able to skip unknown and unused tags. This packet may contain different tags each time to reduce network load. The order of the tags can vary from packet to packet.

2.6.4 Scale information (type 13)

This packet contains scaling information about the character.

It contains two sections. The first contains the null pose definition. This is a T-pose with all orientations set exactly to identity, which is why the orientations are not sent.

```
4 bytes: the number of segments as an unsigned integer
For each segment:
        String: the name of the segment
        3-component vector: the position of the origin of the segment in the null pose
```

The second part contains point definitions that can be used to scale a mesh:

```
4 bytes containing an unsigned integer: the number of points
For each point:
        2 bytes: the id of the segment containing the point
        2 bytes: the point id of the point within the segment
        A string containing the name of the segment
        4 bytes: unsigned integer containing flags describing the point’s characteristics
        3-component vector: the position of the point relative to the segment origin in the null pose
```

2.7 Additional Information

These datagrams provide additional data, but do not by themselves define a full pose.

2.7.1 Joint Angles (type 20)

Information about each joint is sent as follows.

```
4 bytes point ID of parent segment connection. See 2.5.10
4 bytes point ID of child segment connection. See 2.5.10
4 bytes floating point rotation around segment x–axis
4 bytes floating point rotation around segment y–axis
4 bytes floating point rotation around segment z–axis
```

Total: 20 bytes per segment

The coordinates use a Z-Up, right-handed coordinate system.

2.7.2 Linear Segment Kinematics (type 21)

Information about each segment is sent as follows.

4 bytes segment ID See 2.5.9
4 bytes x–coordinate of segment position
4 bytes y–coordinate of segment position
4 bytes z–coordinate of segment position
4 bytes x component of segment global velocity
4 bytes y component of segment global velocity
4 bytes z component of segment global velocity
4 bytes x component of segment global acceleration
4 bytes y component of segment global acceleration
4 bytes z component of segment global acceleration

Total: 40 bytes per segment

The coordinates use a Z-Up, right-handed coordinate system.

2.7.3 Angular Segment Kinematics (type 22)

Information about each segment is sent as follows.

4 bytes segment ID See 2.5.9
4 bytes q1 rotation – segment rotation quaternion component 1 (re)
4 bytes q2 rotation – segment rotation quaternion component 1 (i)
4 bytes q3 rotation – segment rotation quaternion component 1 (j)
4 bytes q4 rotation – segment rotation quaternion component 1 (k)
4 bytes x component of segment global angular velocity
4 bytes y component of segment global angular velocity
4 bytes z component of segment global angular velocity
4 bytes x component of segment global angular acceleration
4 bytes y component of segment global angular acceleration
4 bytes z component of segment global angular acceleration

Total: 44 bytes per segment

The coordinates use a Z-Up, right-handed coordinate system.

2.7.4 Motion Tracker Kinematics (type 23)

Information about each motion tracker is sent as follows.

```
4 bytes segment ID to which the tracker is attached See 2.5.9
4 bytes q_re tracker global orientation
4 bytes q_i tracker global orientation
4 bytes q_j tracker global orientation
4 bytes q_k tracker global orientation
4 bytes x–coordinate of tracker global free acceleration
4 bytes y–coordinate of tracker global free acceleration
4 bytes z–coordinate of tracker global free acceleration
4 bytes x component of segment local acceleration
4 bytes y component of segment local acceleration
4 bytes z component of segment local acceleration
4 bytes x component of segment local angular velocity
4 bytes y component of segment local angular velocity
4 bytes z component of segment local angular velocity
4 bytes x component of segment local magnetic field
4 bytes y component of segment local magnetic field
4 bytes z component of segment local magnetic field
```

Total: 68 bytes per segment.

Only data for segments with a tracker is sent. So it's important to check the segment ID for this datagram.

The coordinates use a Z-Up, right-handed coordinate system.

2.7.5 Center of Mass (type 24)

Information about the center of mass is sent as follows.

```
4 bytes x-coordinate of center of mass position
4 bytes y-coordinate of center of mass position
4 bytes z-coordinate of center of mass position
```

Total: 12 bytes

The coordinates use a Z-Up, right-handed coordinate system.

2.7.6 Time Code (type 25)

Information about time code is sent as follows.

```
12 byte string formatted as such HH:MM:SS.mmm
```

Total: 12 bytes

3 Data Types

3.1 Segment IDs

Table 1: Euler and Quaternion protocols

Segment Name	Segment Index	Segment ID
Pelvis	0	1
L5	1	2
L3	2	3
T12	3	4
T8	4	5
Neck	5	6
Head	6	7
Right Shoulder	7	8
Right Upper Arm	8	9
Right Forearm	9	10
Right Hand	10	11
Left Shoulder	11	12
Left Upper Arm	12	13
Left Forearm	13	14
Left Hand	14	15
Right Upper Leg	15	16
Right Lower Leg	16	17
Right Foot	17	18
Right Toe	18	19
Left Upper Leg	19	20
Left Lower Leg	20	21
Left Foot	21	22
Left Toe	22	23
Prop1	24	25
Prop2	25	26
Prop3	26	27
Prop4	27	28

Table 2: Unity3D protocol

Segment Name	Segment Index	Segment ID
Pelvis	0	1
Right Upper Leg	1	2
Right Lower Leg	2	3
Right Foot	3	4
Right Toe	4	5
Left Upper Leg	5	6
Left Lower Leg	6	7
Left Foot	7	8
Left Toe	8	9
L5	9	10
L3	10	11
T12	11	12
T8	12	13
Left Shoulder	13	14
Left Upper Arm	14	15
Left Forearm	15	16
Left Hand	16	17
Right Shoulder	17	18
Right Upper Arm	18	19
Right Forearm	19	20
Right Hand	20	21
Neck	21	22
Head	22	23