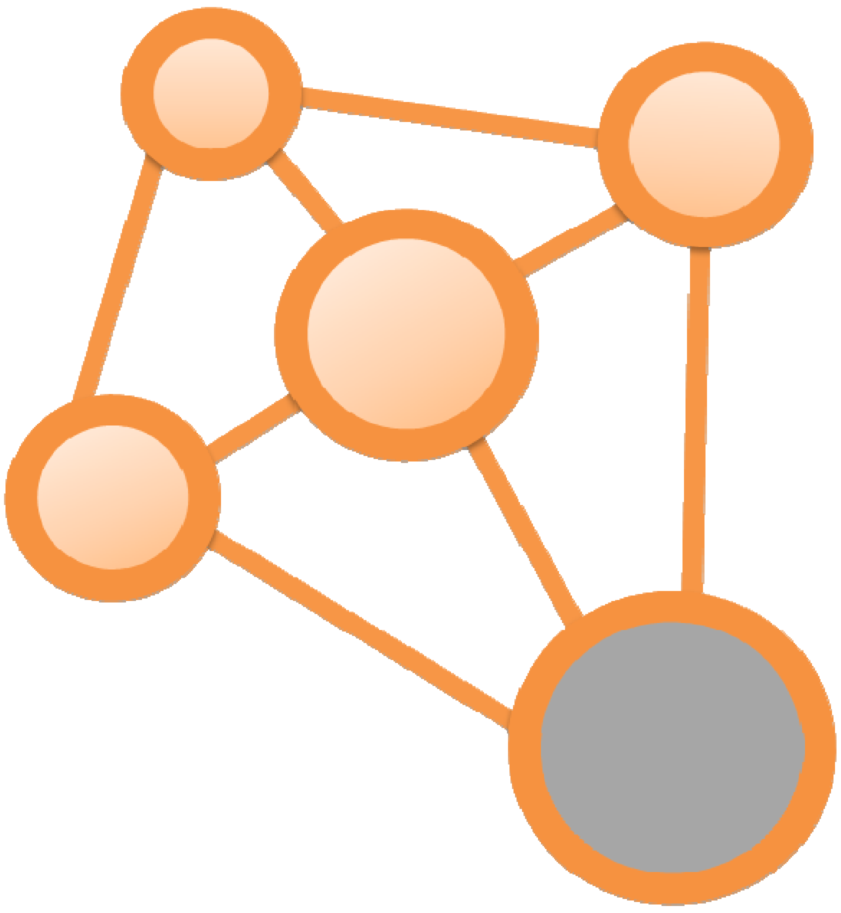
MVN Studio real-time network streaming

Protocol Specification

Document MV0305P, Revision E, February 2012



Xsens Technologies B.V.

phone	+31 88 97367 00
fax	+31 88 97367 01
email	info@xsens.com
internet	**www.xsens.com**

Revisions

Revision	Date	By	Changes
E	February 2012	DOS	Updated for release MVN Studio 3.3

Table of Contents

1 Introduction

MVN Studio, 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 animations for example. The data transfer to other applications is primarily file based when using MVN Studio. With the XME API (SDK) there are many other options.

However, in many scenarios it is attractive to keep the ease of use of MVN Studio, but still being able to receive and process the motion capture data in **real-time** in another application, even on a 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 Studio Real-time Network Streaming. The streaming feature enables computers that run MVN Studio to stream the captured data over a network to other client computers.

1.1 Perceived Usage

1.1.1 Usage in real-time pre-viz 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 Studio 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.

To support scenarios like this for usage with **Autodesk Motion Builder as the client application** specifically, Xsens has developed a plug-in for Motion Builder that works with the protocol specified in this document to receive motion capture data in real-time. This client plug-in can be purchased separately.

1.1.2 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 datagloves. 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 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 Studio itself.

1.2 Protocol

MVN Studio real-time network streaming protocol is based on UDP and is specified in this document.

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. **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) 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 Studio 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 whose length depends upon the configuration of MVN Studio. Common sampling rates lie between 60 and 120 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.

2.4.1 Header

The number of captured body segments is contained in the datagram header along with additional information:

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 avatar ID
7 bytes reserved for future use

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 • Position of joints and Euler angles of segments. • Y-Up, right-handed • *Supported by MVN Studio as a network client*
02	Pose data • Positions of joints and Quaternions of segments • Z-Up, right-handed • *Supported by MVN Studio as a network client*
03	Pose data • Positions selected defined points (simulating optical markers), typically 38-46 points. Multiple data sets are available. • This protocol is used with the Motion Builder plug-in v1.0. • ***Not** supported by MVN Studio as a network client. However, MVN Studio has only a limited ability of re-integrating these marker positions into proper segment positions, so it looks anywhere from sort of correct to very weird.*
04	MotionGrid Tag data • Contains x, y, z positions for unassigned MotionGrid tags.
10	Character information → scale information • 3-float offset and 9-float rotation matrix for each segment • *Supported by MVN Studio as a network client and intended for MVN Studio avatar mesh specifically*
11	Character information → prop information • Character strings containing the names of the used props • *Supported by MVN Studio as a network client*
12	Character information → meta data • name of the avatar • MVN avatar ID (Xbus Master ID) • << more can be added later >> • *Supported by MVN Studio as a network client*

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.

Sample Counter
The sample counter is a 32-bit unsigned integer value which is incremented by one each time a new set of motion sensor 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.

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.

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 sensors that were being 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.

Time code
MVN Studio 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.

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

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

Segment data Euler (type 01)
Information about each segment is sent as follows.

4 bytes segment ID, in the range 1-30 (see paragraph 3.1)
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.

23 segments will be send, followed by a maximum of four props, if they exist. If the avatar has assigned MotionGrid tags, these will be included in remaining prop locations. For instance, if there are two props and three tags, the message will contain 23 segments, 2 props and 2 tags. The third tag will not be send. The rotation for a tag will be (0, 0, 0).

Segment data quaternion (type 02)
Information about each segment is sent as follows.

4 bytes segment ID, in the range 1-30 (see paragraph 3.1)
4 bytes x–coordinate of sensor position
4 bytes y–coordinate of sensor position
4 bytes z–coordinate of sensor position
4 bytes q1 rotation – sensor rotation quaternion component 1 (re)
4 bytes q2 rotation – sensor rotation quaternion component 1 (i)
4 bytes q3 rotation – sensor rotation quaternion component 1 (j)
4 bytes q4 rotation – sensor rotation quaternion component 1 (k)

Total: 32 bytes per segment

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

23 segments will be send, followed by a maximum of four props, if they exist. If the avatar has assigned MotionGrid tags, these will be included in remaining prop locations. For instance, if there are two props and three tags, the message will contain 23 segments, 2 props and 2 tags. The third tag will not be send. The rotation quaternion for a tag will be (1, 0, 0, 0).

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 avatar will be send, using the same position format as the markers.

MotionGrid Tag data (type 04)
This message contains the positions of the unassigned MotionGrid tags. Information about each available unassigned tag is sent as follows:

4 bytes tagId
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.

Please note: only updated tags will be send. If the network streamer is configured to use frame skipping, only the latest position for a tag will be send. The number of tags and the order of the tags inside the message will thus vary between messages.

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

Rotation (Euler)
The rotation of a captured sensor in the Euler representation is always stored as a 3D vector composed of three 32-bit float values. The unit of these float values is in degree.

Rotation (Quaternion)
The rotation of a captured sensor 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.

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

Point ID
The ID of a point depends on the ID of the segment it is attached to, which is defined in the MVN SDK documentation and the local ID it has in the segment. These local IDs are not documented and are subject to change, although the currently defined IDs *should* not change. The ID is sent as a 4-byte integer, defined as 100 * 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 100*1 + 13 = 113.

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 Studio 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.

2.6 Character information

Scale information (type 10)
This packet contains scaling information about the character. This information is only useful for MVN Studio, since it only applies to a mesh of a specific size. However, there may be a way to put this information to use for other meshes.

The data sent is:

4 bytes segment ID, in the range 1-30
4 bytes x–offset of segment relative to default position
4 bytes y–offset of segment relative to default position
4 bytes z–offset of segment relative to default position
9x4 bytes matrix containing the rotation and scale of the segment relative to the default

Total: 52 bytes per segment

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

Prop information (type 11)
This packet contains information about the number of used props and their names. MVN Studio uses the name to load an appropriate mesh to display the prop.
When the item count in the header is 0, there are no defined props and the rest of the message will be empty.

4 bytes segment ID, in the range 24-30 (usually 24 or 25)
n bytes containing a null-terminated string with the name of the prop

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 Studio
xmid: contains the Xbus Master ID as shown in MVN Studio
color: contains the color of the avatar as used in MVN Studio, the format is hex RRGGBB

More tags may be added later, so any implementation should be able to skip unknown and unused tags. Also, this packet may contain different tags each time to reduce network load.

3 Data Types

3.1 Segment IDs

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**	**23**
Left Toe	**23**	**24**
Prop1	**24**	**25**
Prop2	**25**	**26**
Prop3	**26**	**27**
Prop4	**27**	**28**