



NatNet API

User's Guide

Version 2.10.0
May 2016

Technical support
help.naturalpoint.com
1-888-965-0435

TABLE OF CONTENTS

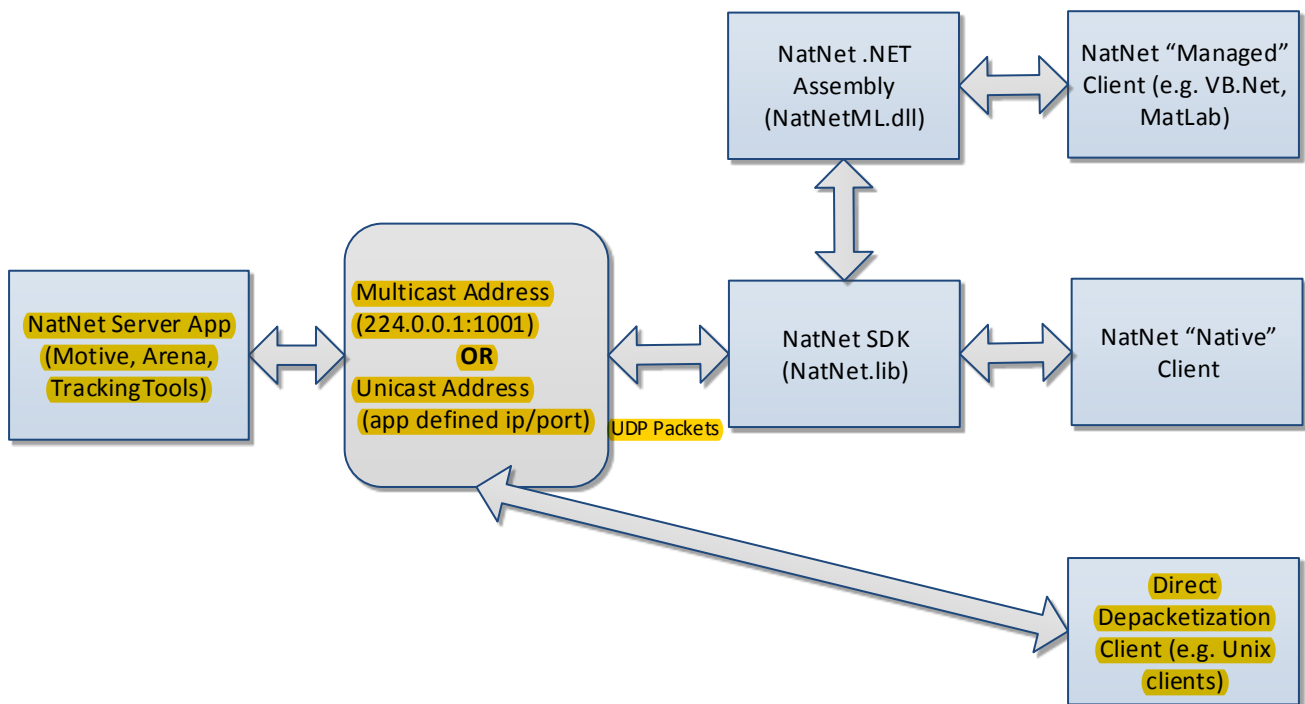
NatNet Overview	3
SDK Contents	4
Folder Contents.....	4
Running the Samples	6
Running the Simple Client-Server Sample	6
Running the rigid body sample (SampleClient3D)	7
Running the .NET sample	8
Running the Unity3D sample	11
Using the NatNet SDK	12
Building a Native Client to Receive NatNet Data	12
Building a Native Server to Send NatNet Data	12
Building a Managed .NET Client to Receive NatNet Data	12
API Reference.....	12
NatNet Data Types	13
NatNetClient Class	14
Description	14
Constructor & Destructor Documentation	19
Member Function Documentation	19
Appendix A : Bitstream Syntax.....	23
Building a Direct Depacketization Client (Without NatNet).....	23
Technical Support	24

NATNET OVERVIEW

The NatNet SDK is a Client/Server networking SDK for sending and receiving NaturalPoint data across networks. NatNet uses the UDP protocol in conjunction with either Point-To-Point Unicast or IP Multicasting for sending data.

The following diagram outlines the major component communication of a typical NatNet setup.

Figure 1 – NatNet Component Overview



A NatNet Server has 2 threads and 2 sockets, one for sending data, and one for receiving/sending commands.

NatNet servers and clients can exist either on a same machine or separate machines. Additionally, multiple NatNet clients can connect to a single NatNet server. When a NatNet server is configured to use IP Multicast, the data is only sent once, to the Multicast group.

SDK CONTENTS

The NatNet SDK consists of:

NatNet Library	Native C++ networking library (headers, static library (.lib) and dynamic import library (.lib/.dll)).
NatNet Assembly	Managed .NET assembly (NatNetML.dll) for use in .NET compatible clients.
NatNet Samples	Sample projects and executables designed to be quickly integrated into your own code.

FOLDER CONTENTS

Folder	Contents
\include	NatNet SDK header files. Client applications should include these.
\lib	Static and dynamic library files for the NatNet SDK.
\lib\x64	64-bit versions of the library files.
\Samples	VisualStudio 2005 samples. Use the solution file here to open all sample projects.
\Samples\bin	Precompiled samples with sample data files.
\Samples\BroadcastSample	Sample application illustrating how to use remote record trigger in Motive using XML formatted UDP broadcast packets.
\Samples\Matlab (Explanation Below)	Sample MATLAB code file (.m) for using MATLAB with the NatNet managed assembly (NatNETML.dll).
\Samples\MatlabWrapper	Wrapper class for the NatNetML assembly members.
\Samples\MayaPlugin	https://github.com/mocap-ca/mayaMocap/tree/master/mayaMotive
\Samples\NatCap	Sample Capture start/stop broadcast app.
\Samples\PacketClient	Simple example showing how to connect to a NatNet multicast stream and decode NatNet packets directly without using the NatNet SDK.
\Samples\PythonClient	Sample Python code file (.py) for using Python with NatNet streaming.

\Samples\SampleClient (Explanation Below)	Sample NatNet console app that connects to a NatNet server, receives a data stream, and writes that data stream to an ascii file
\Samples\SampleClient3D (Explanation Below)	Sample NatNet console app that connects to a NatNet server, receives a data stream, and displays that data in an OpenGL 3D window.
\Samples\SimpleServer	Illustrates the minimum code required to create and send marker data using the NatNet server.
\Samples\TimingClient	This program connects to a NatNet server and can be used as a quick check to determine packet timing information.
\Samples\Unity3D (Explanation Below)	This program connects to a NatNet server, receives a data stream, encodes a skeleton to XML, and outputs XML locally over UDP to Unity.
\Samples\VCRedist	Executable for 32bit Visual C++ Redistributable 2005.
\Samples\WinFormsSample (Explanation Below)	Simple C# .NET sample showing how to use the NatNet managed assembly (NatNETML.dll).



RUNNING THE SAMPLES

Pre-compiled versions of the NatNet samples have been provided in the \Samples\bin folder. These versions can be used to quickly test your application. Please refer to the instructions in this section for information on running specific samples.

Note! The Visual C++ runtime libraries are required to run the samples. If you encounter an error message when attempting to run the samples, especially on machines without Visual C++ installed, please install the VC runtime redistributable package located in Samples\VCRedist. If the problem persists, please try rebuilding the samples using Visual C++, or contact support.

RUNNING THE CONSOLE OUTPUT SAMPLE (SAMPLECLIENT)

1. Start the Optitrack Server (e.g. Motive) and begin streaming data via the Streaming Panel
2. Start the client:
SampleClient.exe [IPAddress] [OutputFilename.txt]

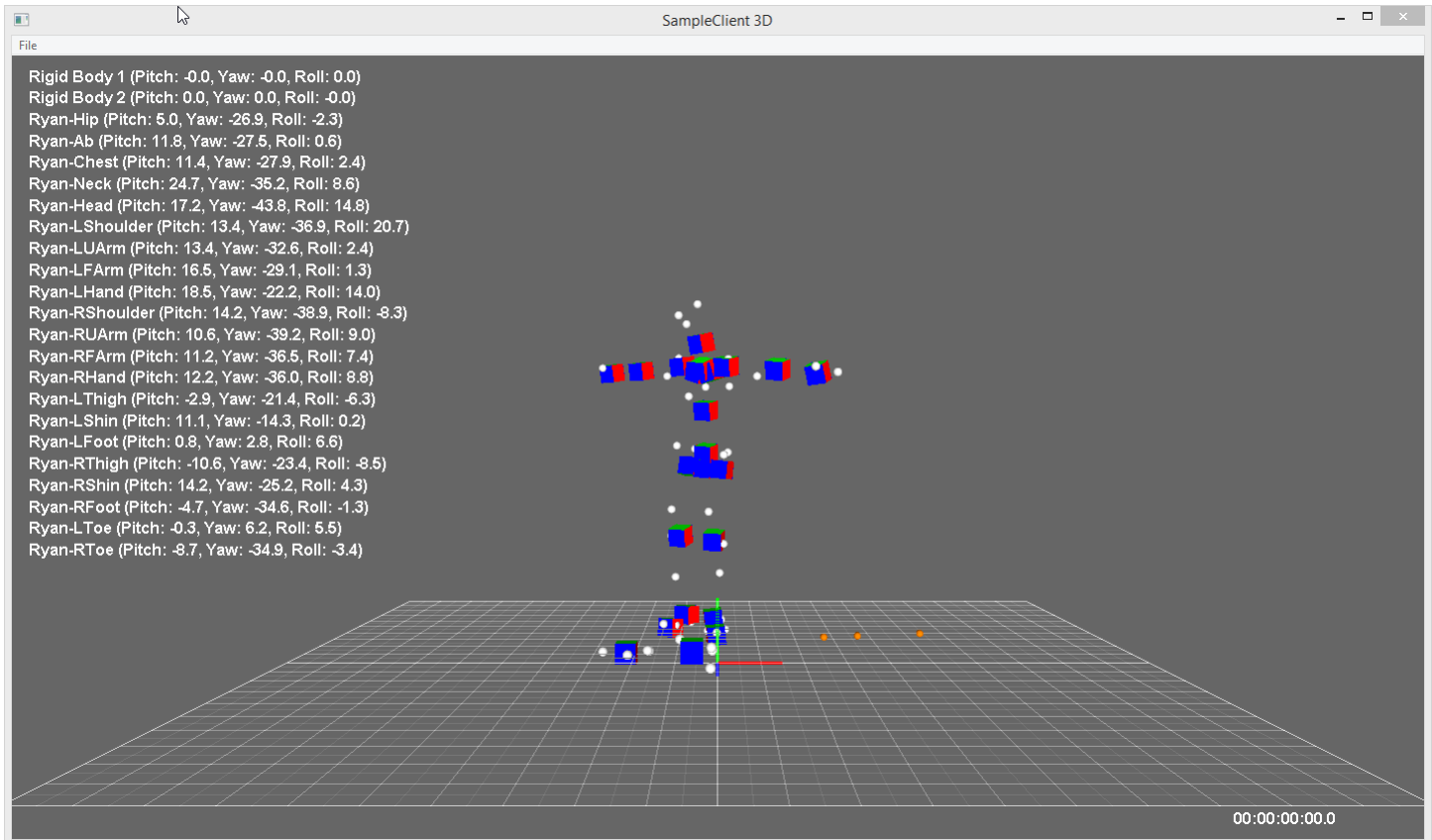
You should begin to see data streaming in the client window or to text file.

Note

- [parameters] are optional.
- If no IP address is specified, the client will assume the server is on the same machine (local machine).

The Rigid Body sample (SampleClient3D) illustrates how to decode NatNet 6DOF Rigid Body and Skeleton Segment data from OptiTrack quaternion format to euler angles and display them in a simple OpenGL 3D viewer. This sample also illustrates how to associate RigidBody/Skeleton Segment names and IDs from the data descriptions with the IDs streamed in the FrameOfMocapData packet.

SampleClient3D - Decoding and drawing labeled rigid body position and orientation (6DOF) data



With Client/Server on same machine:

1. **[Motive]** Load a dataset with rigid body or skeleton definitions
2. **[Motive]** Enable network streaming (Data Streaming Pane -> Check Broadcast Frame Data)
3. **[Motive]** Enable streaming rigid body data (check Stream Options -> Stream Rigid Bodies = True)
4. **[Sample3D]** File -> Connect

With Client/Server on separate machines:

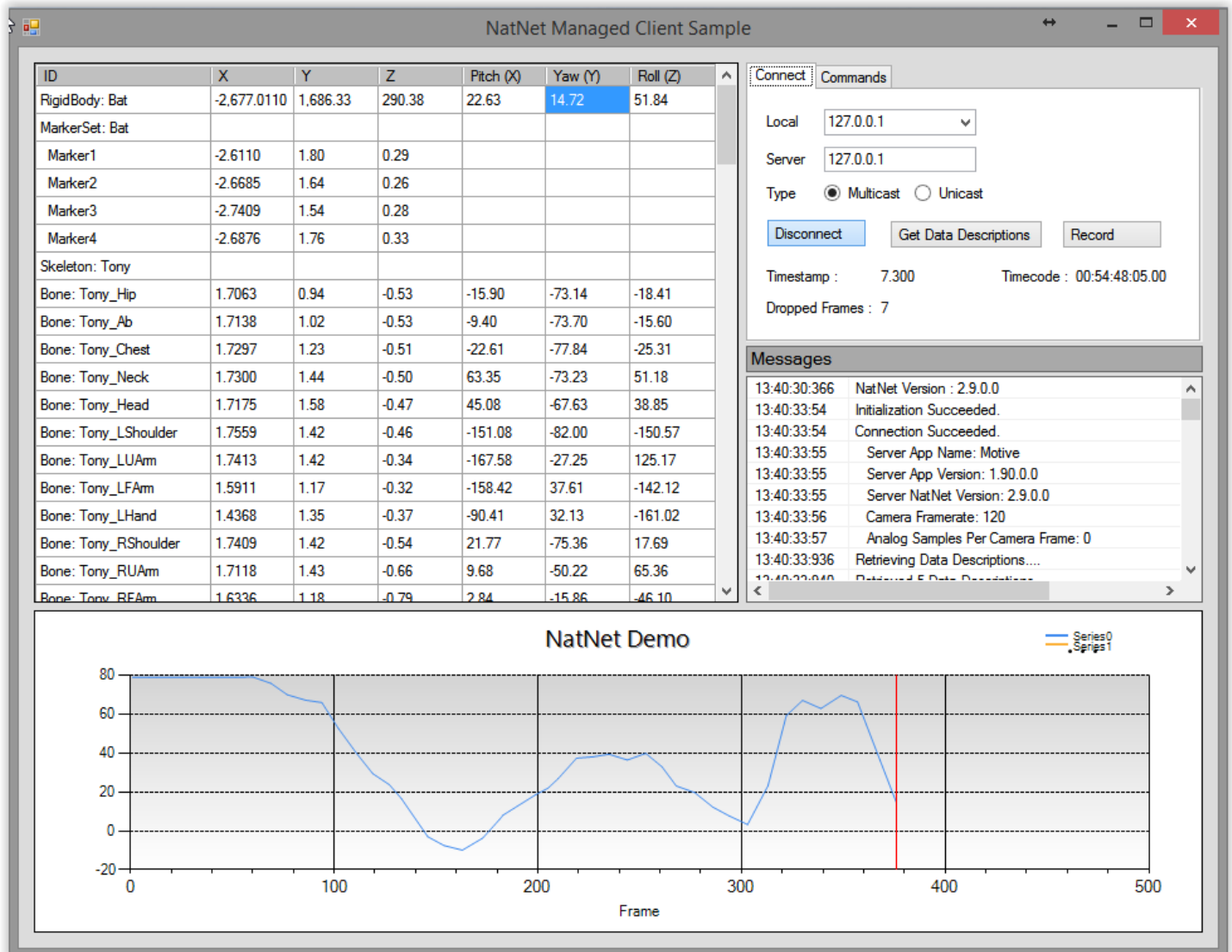
1. **[Motive]** Load a dataset with rigid body or skeleton definitions
2. **[Motive]** Set IP address to stream from (Network Interface Selection -> Local Interface)
3. **[Motive]** Enable network streaming (Data Streaming Pane -> Check Broadcast Frame Data)
4. **[Motive]** Enable streaming rigid body data (check Stream Options -> Stream Rigid Bodies = True)
5. **[Sample3D]** Set Client and Server IP addresses
6. **[Sample3D]** File -> Connect

- **IP Address** IP Address of client NIC card you wish to use.
- **Server IP Address** IP Address of server entered in step 2 above.

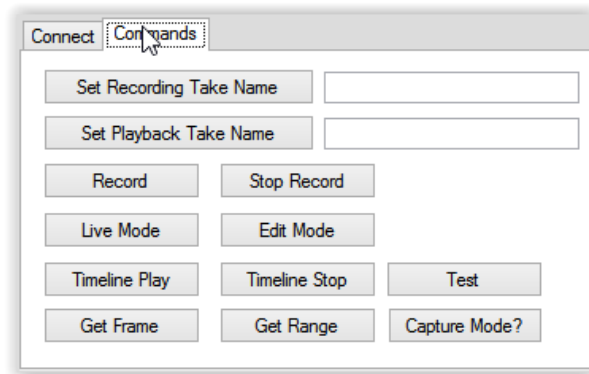
RUNNING THE .NET SAMPLE

1. **[Motive]** Start a NatNet server application (e.g. Motive).
2. **[Motive]** Enable NatNet streaming from the Server application.
3. **[WinFormTestApp]** Start the WinForms sample application from the NatNet Samples folder.
4. **[WinFormTestApp]** Update the “Local” and “Server” IP Addresses as necessary.
5. **[WinFormTestApp]** Press the “Connect” button to connect to the server.
6. **[WinFormTestApp]** Press the “GetDataDesc” button to request and display a detailed description of the Server’s currently streamed objects.
7. **[WinFormTestApp]** Select a Row in the DataGrid to display that value in the graph.

Receiving NatNet data in a .NET Environment



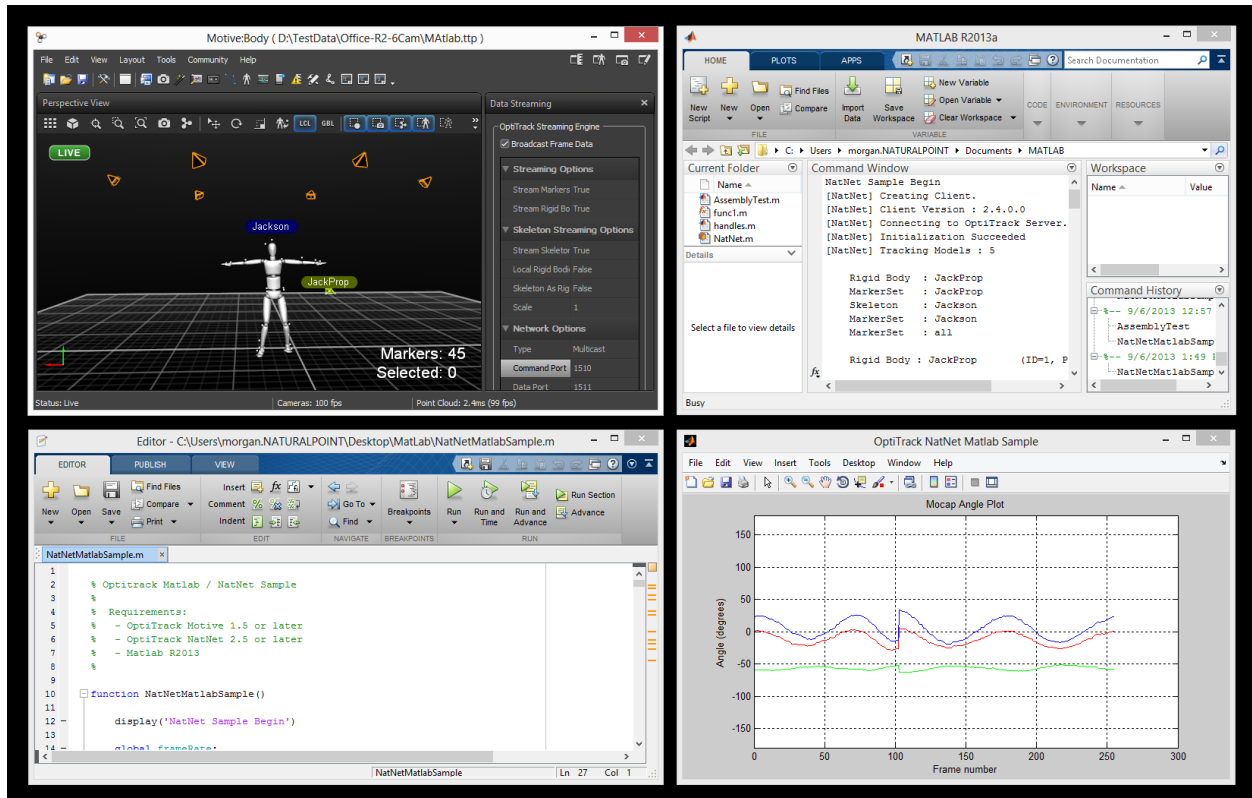
Issuing remote control commands to Motive



RUNNING THE MATLAB SAMPLE

1. **[Motive]** Start a NatNet server application (e.g. Motive).
2. **[Motive]** Enable NatNet streaming from the Server application.
3. **[Matlab]** Start Matlab
4. **[Matlab]** Open the NatNetMatlabSample.m file.
5. **[Matlab]** From the editor window, press **Run**

Real-Time Streaming Mocap data from Motive into MATLAB

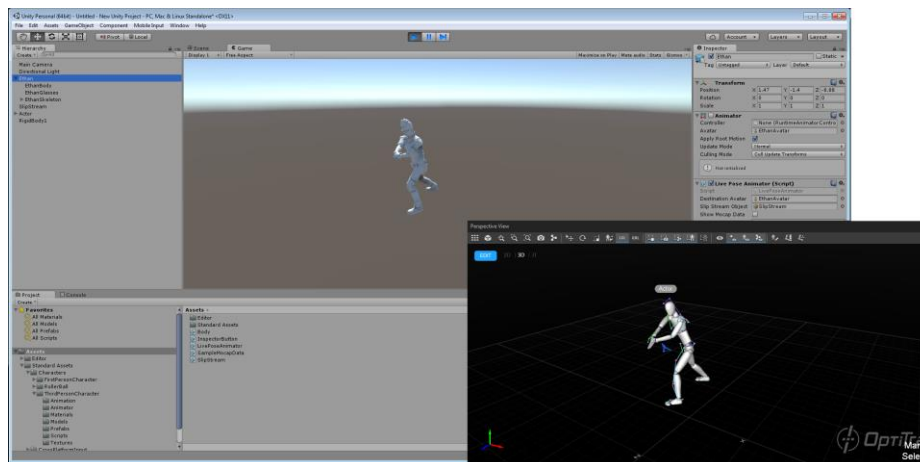


RUNNING UNITY3D SAMPLE

The Unity3D sample application and C# scripts included within the NatNet folder demonstrate streaming rigid body and skeleton tracking data from Motive into Unity via UDP. The following steps show how this sample can be used to animate a character in Unity.

1. **[Motive]** Start Motive.
2. **[Motive]** Have skeleton tracking data available in Motive. You can either open a previously captured *Take* or stream directly from live tracking.
3. **[Motive]** Open the Data Streaming pane, and set the *Local Rigid Body* and *Stream Skeletons* setting to True.
4. **[Motive]** Specify the streaming network address under the Network Interace → Local Interface.
5. **[Motive]** Enable the network streaming from the server application (Motive: Data Streaming Pane → Check Broadcast Frame Data)
6. **[UnitySample]** Run **UnitySample.exe** from the \NatNet SDK\Samples\bin folder. This program connects to the NatNet server, receives tracking data from Motive, and outputs the data into Unity. If this application fails to find the host, go back to Motive and try streaming onto another network interface.
7. **[Unity]** Start Unity.
8. **[Unity]** Create a new project.
9. **[Unity]** Import the C# scripts, or assets, in the \NatNet SDK\Sample\Unity3D folder into Unity.
10. **[Unity]** Create an empty gameobject (GameObject → Create Empty) in the scene and name it SlipStream.
11. **[Unity]** Add the **SlipStream.cs** script component into the SlipStream object.
12. **[Unity]** Under the SlipStream.cs component, type in the streaming Unity3D IP address defined from the server application.
13. **[Unity]** Import a sample character in Unity. (Asset → Import Package → Characters)
14. **[Unity]** From the imported assets folder in the Project panel, load the sample character, Ethan, into the scene. (Assets → Standard Assets → Characters → ThirdPersonCharacters → Models → Ethan)
15. **[Unity]** Select the character, and under the inspector panel, disable the Animator class.
16. **[Unity]** Click Add Component, and add the **LivePoseAnimator.cs** script onto the character.
17. **[Unity]** Enter properties for the LivePoseAnimator component:
 - a. Under the Desintation Avatar, select the avatar that you wish to animate (Ethan → EthanAvatar).
 - b. Under the Slip Stream Object, select the empty object that you've attached the SlipStream.cs script to (SlipStream).
 - c. Under the Actor, select name of the skeleton in Motive that you wish to import.
18. **[Unity]** Double check the settings under the objects (SlipStream and Ethan) and run the project. If all settings are properly configured, Ethan will be animated using tracking data from Motive.

Real-time Streaming into Unity3D



USING THE NATNET SDK

The code samples are the quickest path towards getting NatNet data into your application. We typically recommend you:

1. Identify your application's development/interface requirements (managed, native, etc).
2. Adapt the NatNet sample code from the corresponding NatNet sample application in the samples folder into your application.
3. Use the API reference in the next page for additional information.

The Visual Studio solution file `\Samples\NatNetSamples.sln` will open and build all of the NatNet sample projects.

If you are creating an application from scratch, please refer to the following sections for application specific requirements.

BUILDING A NATIVE CLIENT TO RECEIVE NATNET DATA

Steps for building a NatNet client application/library to receive data from a NatNet server application such as Motive:

1. Adapt the SampleClient sample (SampleClient.cpp) to your application's code.
2. Include NatNetClient.h and NatNetTypes.h
3. Link to NatNetLib.lib (dynamic) **OR** NatNetLibStatic.lib (static)
4. [OPTIONAL] If linking dynamically, define NATNETLIB_IMPORTS and distribute NatNetLib.dll with your application

Note : Be sure to link to ws2_32.lib if linking to NatNetLib statically.

BUILDING A NATIVE SERVER TO SEND NATNET DATA

Steps for building a NatNet server application/library to send/forward NatNet formatted data to a NatNet client application:

1. Adapt SimpleServer (SampleServer.cpp) to your application's code.
2. Include NatNetServer.h and NatNetTypes.h
3. Link to NatNetLib.lib (dynamic) **OR** NatNetLibStatic.lib (static)
4. [OPTIONAL] if linking dynamically, define NATNETLIB_IMPORTS and distribute NatNetLib.dll with your application

Note : Be sure to link to ws2_32.lib if linking to NatNetLib statically.

BUILDING A MANAGED .NET CLIENT TO RECEIVE NATNET DATA

Steps for building a managed NatNet client application.

1. Add the NatNetML.dll .NET assembly as a reference to your VB.NET/C# project.
2. The NatNetML namespace is now available to your code, in addition to intellisense library comments.

Note : When distributing your .NET application, be sure to distribute the NatNetML.dll as well.

API REFERENCE

The NatNET API consist of the following objects:

NatNetClient	The class for communicating with a NatNet Server such as Motive.
NatNetServer	The class for implementing a NatNet server and sending NatNet formatted data packets.
NatNet Data Types	Structures encapsulating data encoded in NatNet packets.
NatNet Assembly	A managed (.NET) class library that can be called by .NET components. The NatNet assembly wraps the underlying native NatNet library, exposing the NatNetClient and NatNet Data Types for use in .NET compatible environments (e.g. VB.NET, C#, LabView, MatLab).

NATNET DATA TYPES

NatNet server applications stream the following types of motion capture data.

NatNet Data Types

Data Type	Description
MarkerSet Data	A named collection of identified markers and the marker positions (X,Y,Z). This list is ordered, padded, point cloud solved, model filled (where occluded). This list also contains special MarkerSet named "all" which is a list of all labeled markers.
RigidBody Data	A named segment with a unique ID, position, and orientation data, and the collection of identified markers used to define it. Marker data is model-solved positions.
Skeleton Data	A named, hierarchical collection of RigidBodies. Marker data is model-solved positions.
Labeled Markers	Ordered, padded, point cloud solved, model filled (where occluded) labeled Marker data (labeled markers not associated with a "MarkerSet").
Unlabeled Markers (Other markers)	List of point cloud solved 3D positions for all markers in the frame that were unlabeled by either the skeleton or the rigid body solver.
Force Plate Data	Force plate channel data (Fx, Fy, Fz, Mx, My, Mz). Force plate data will contain multiple samples per mocap frame, depending upon the force plate acquisition rate.
Timestamp data	Timing information for the frame, including: Frame ID Frame Timestamp SMPTE Timecode (If timecode is present)

NatNet clients can discover what data objects a server application is currently streaming out of band or ahead of time using the **DataSetDescriptions** structure. NatNet clients receive actual data from a server using the **FrameOfMocapData** structure. Both of these packets are delivered to the client via the **DataHandler** callback.

Dataset Descriptions This packet contains a description of the motion capture data sets (MarkerSets, Skeletons, RigidBody) for which a frame of motion capture data will be generated.

Frame of Mocap Data This packet contains a single frame of motion capture data for all the data sets described in the Dataset Descriptions.

In addition to the formal data sets described above, **FrameOfMocapData** also contains additional per-frame tracking data. This additional data is not described in the **DataSetDescriptions** structure as it is typically not known ahead of time or out of band, but on a frame by frame basis:

Labeled Markers Labeled Markers not associated with a pre-defined MarkerSet or RigidBody. This data type is used when MarkerSets and/or RigidBodies are not explicitly defined in the Tracking Application but labeled markers are still generated.

Other Markers All 3D points that were triangulated but not labeled for the given frame.

It is possible for data to be duplicated between structures. For example, a given **FrameOfMocapData** may contain the same marker in **LabeledMarkers** as well as within the **RigidBody** data structure. In those cases, marker id can be used in the client code to correlate if necessary.

The **SampleClient** sample illustrates how to retrieve data descriptions and data and interpret this data.

Please refer to the **NatNetTypes.h** header file or the NatNetML.dll assembly for the most up to date descriptions of the types.

COORDINATE SYSTEM CONVENTIONS

In the NatNet data stream, orientation data is represented as a quaternion. Quaternion orientations are order independent, however they do indicate a handedness. When decomposing quaternions into Euler angles, it is important to consider the coordinate system conventions you want to convert into. An Euler angle convention must account for:

- Rotation order
- Left handed or Right handed
- Static (Global) or Relative (Local) Axes

As an example, the OptiTrack Motive software uses the following "Motive" coordinate system convention:

X (Pitch), Y (Yaw), Z (Roll), Right-Handed (RHS), Relative Axes (aka 'local')

The NatNet SDK includes quaternion to Euler conversion routines. Please refer to the WinForms sample or the SampleClient3D for specific implementation details and usage examples.

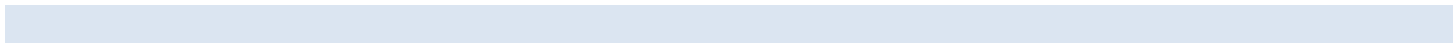
NatNet provides a command / request mechanism for passing commands and request between NatNet server's (Motive) and NatNet Client's (your application). Examples of commands / requests are starting/stopping record, setting the current take name, or querying the server for the current framerate.

Motive Supported Command/Request

Command	Description	Parameters	Returns
UnitsToMillimeters	Request current system's units, in terms of millimeters	none	float
FrameRate	Request current system's tracking framerate	none	float
StartRecording	Start recording	none	none
StopRecording	Stop recording	none	none
LiveMode	Switch to Live mode	none	none
EditMode	Switch to Edit mode	none	None
CurrentMode	Request current mode	none	int
TimelinePlay	Start take playback	none	none
TimelineStop	Stop take playback	none	none
SetPlaybackTakeName	Set playback take	Take name	None
SetRecordTakeName	Set record take name	Take name	None
SetCurrentSession	Set current session	Session name	None
SetPlaybackStartFrame	Set start frame	Frame number	None
SetPlaybackStopFrame	Set stop frame	Frame number	None

SetPlaybackCurrentFrame	Set current frame	Frame number	None
CurrentTakeLength	Request length of current take	None	int
AnalogSamplesPerMocapFrame	Request number of analog samples per motion capture frame	None	int

- Refer to the WinForms example for a complete list and usage of these commands.
- Refer to the ***SendMessage(...)*** and ***SendMessageAndWait(...)*** functions in the API reference for more details.

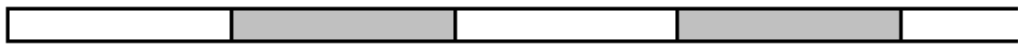


On supported systems, all frames of NatNet data will contain an OptiTrack timecode stamp, which is an extended form of the typical studio SMPTE timecode stamp.

Note: SMPTE Timecode support requires an OptiTrack eSync hub.

Because motion capture frame rates typically exceed standard SMPTE timecode frame rates, an additional “subframe” value is added to the end of the timecode stamp. This “subframe” value is the 0-based, n^{th} “in-between” frame:

Typical OptiTrack Timecode Representation
(120 fps mocap data, 30-fps no-drop SMPTE timecode source)

Mocap Frame	1	2	3	4	5
					
SMPTE	00:00:00:01	00:00:00:01	00:00:00:01	00:00:00:01	00:00:00:02
SubFrame	0	1	2	3	0
OptiTrack Timecode	00:00:00:01.0	00:00:00:01.1	00:00:00:01.2	00:00:00:01.3	00:00:00:02.0

In the above representation, for a typical 120 fps motion capture session synched to a 30 fps no-drop SMPTE timecode studio-sync source, there is a **4 : 1** ratio of motion capture frames to studio frames. The extra motion capture frames are represented by the OptiTrack SubFrame field in the OptiTrack timecode.

The generic form for OptiTrack timecode is:

HH:MM:SS:FF.Y	hours:minutes:seconds:frames.subframe
----------------------	---------------------------------------

OptiTrack timecode is sent to NatNet clients in the form of 2 unsigned integers.

unsigned int Timecode	OptiTrack encoded SMPTE timecode
unsigned int TimecodeSubframe	OptiTrack encoded sub-frame data

Note : The FrameOfMocapData structure contains two entries for timecode information for a given frame of timecode data.

The **Timecode** parameter is interpreted differently when streaming Live or from File Playback (Edit).

In Live mode, the **Timecode** parameter is only valid when SMPTE timecode is present in your Mocap hardware setup, typically when using the eSync and a timecode generator. When present, the **Timecode** parameter will be a correctly formatted SMPTE timecode value.

In Edit Mode, the **Timecode** parameter is the current frame number converted to SMPTE Timecode format.

The NatNet SDK provides helper routines to decode the Timecode parameter into a string friendly format as well as the subframe of mocap data that may exist between whole timecode frames.

Latency is the capture computer's hardware timestamp for the given frame, which is also displayed in Motive in the Camera Preview viewport when camera info is enabled. This is the same whether live or playback from file.

Timecode values should not be used directly, but decoded using the NatNet timecode utility functions:

<pre>bool DecodeTimecode(unsigned int inTimecode, unsigned int inTimecodeSubframe, int* hour, int* minute, int* second, int* frame, int* subframe);</pre>	Helper function to decode OptiTrack timecode data into individual timecode values
<pre>bool TimecodeStringify(unsigned int inTimecode, unsigned int inTimecodeSubframe, char *Buffer, int BufferSize);</pre>	Helper function to decode OptiTrack timecode into a user friendly string in the form "hh:mm:ss:ff:yy"

The following is an example of how to decode timecode using the NatNet helper functions (from the *SampleClient.cpp* example):

```
// decode timecode to values
int hour, minute, second, frame, subframe;
bool bValid = pClient->DecodeTimecode(data->Timecode, data->TimecodeSubframe, &hour, &minute,
&second, &frame, &subframe);

// decode timecode to friendly string
char szTimecode[128] = "";
pClient->TimecodeStringify(data->Timecode, data->TimecodeSubframe, szTimecode, 128);
printf("Timecode : %s\n", szTimecode);
```

DESCRIPTION

NatNetClient is a complete C++ class for connecting to NatNet server applications, such as Motive.

CONSTRUCTOR & DESTRUCTOR DOCUMENTATION**NatNetClient::NatNetClient ()**

Creates a new (multicast) instance of a NatNet Client.

NatNetClient::NatNetClient (int iConnectionType)

Creates a new instance of a NatNet Client using the specified connection protocol.

Parameters:

iConnectionType *Type of connection (0 = Multicast, 1 = Unicast).*

NatNetClient::~~NatNetClient ()

Destructor.

NatNetClient::Uninitialize()

Disconnects from server.

MEMBER FUNCTION DOCUMENTATION**int NatNetClient::GetDataDescriptions (sDataDescriptions ** pDataDescriptions)**

Requests a description of the current streamed data objects from the server app. This call blocks until request is responded to or times out.

Parameters:

pDataDescriptions *Array of Data Descriptions.*

Returns:

On success, number of data objects. 0 otherwise.

sFrameOfMocapData * NatNetClient::GetLastFrameOfData ()

Retrieves the most recently received frame of mocap data.

Returns:

Frame of Mocap Data

int NatNetClient::GetServerDescription (sServerDescription * pServerDescription)

Requests a description of the current NatNet server the client is connected to. This call blocks until request is responded to or times out.

Parameters:

pServerDescription *Description of the NatNet server.*

Returns:

On success, number of data objects. 0 otherwise.

int NatNetClient::Initialize (char * szLocalAddress, char * szServerAddress)

int NatNetClient::Initialize (char * szLocalAddress, char * szServerAddress, int HostCommandPort)

int NatNetClient::Initialize (char * szLocalAddress, char * szServerAddress, int HostCommandPort, int HostDataPort)

Initializes client socket and attempts to connect to a NatNet server at the specified address.

Parameters:

szLocalAddress *IP address of client*

szServerAddress *IP address of server*

HostCommandPort *server command port (default = 1510)*

HostDataPort *server data port (default = 1511)*

Returns:

0 if successful, error code otherwise

void NatNetClient::SetMulticastAddress (char * szMulticast)

Sets the NatNet server multicast group/address to connect to. SetMulticastAddress() must be called before calling Initialize(...).

Parameters:

szCommand *application defined Message string*

void NatNetClient::NatNetVersion (unsigned char Version[4])

Retrieves the version of the NatNet library the client is using.

Parameters:

Version *version array (form: major.minor.build.revision)*

void NatNetClient::SendMessage (char * szCommand)

Sends a message to the server and returns. Response will be delivered in-band.

Parameters:

szCommand application defined Message string

int NatNetClient::SendMessageAndWait (char * szCommand, int tries, int timeout, void ** Response, int * pnBytes)

Sends an application-defined message to the NatNet server and waits for a response.

Parameters:

*szCommand Application defined message.
tries Number of times to try and send the message
timeout time to wait for response (in milliseconds) before timing out
Response Application defined response.
pnBytes Number of bytes in response*

Returns:

0 if successful, error code otherwise.

int NatNetClient::SendMessageAndWait (char * szCommand, void ** Response, int * pnBytes)

Sends an application-defined message to the NatNet server and waits for a response.

Parameters:

*szCommand Application defined message.
Response Application defined response.
pnBytes Number of bytes in response.*

Returns:

0 if successful, error code otherwise.

int NatNetClient:: SetDataCallback(void (*CallbackFunction)(sFrameOfMocapData *FrameOfData, void* pUserData), void* pUserData /*=NULL*/)

Sets the data callback function for NatNet frame delivery. This function will be called whenever NatNet receives an in-band data (e.g. frame of data).

Parameters:

*CallbackFunction Callback Function
pUserData User-Definable data*

Returns:

0 if successful, error code otherwise.

void NatNetClient::SetVerbosityLevel (int *iLevel*)

Sets the message reporting level for internal NatNet messages.

Parameters:

iLevel Verbosity level (see Verbosity level in **NatNetTypes.h**)

int NatNetClient::Uninitialize ()

Disconnects from the current NatNet Server.

Returns:

0 if successful, error code otherwise.



APPENDIX A : BITSTREAM SYNTAX

In order to provide the most current bitstream syntax, the NatNet SDK includes a testable working depacketization sample that decodes NatNet Packets directly without using the NatNet client library.

Note: Decoding packets directly is not recommended. The bitstream packet syntax is subject to change, requiring an application to rebuild against the latest NatNet library. NatNet packets should only be decoded directly where use of the NatNet library is not possible.

Using the NatNet client library protects client applications from future bitstream syntax changes.

BUILDING A DIRECT DEPACKETIZATION CLIENT (WITHOUT NATNET)

For situations where you would like to receive a NatNet data stream but it is not possible to use the NatNet client library (e.g. on an unsupported platform such as Unix), you can use the PacketClient sample as a template for depacketizing NatNet packets directly.

1. Adapt the PacketClient sample (PacketClient.cpp) to your application's code.
2. Regularly update your code with each revision to the NatNet bitstream syntax.

NaturalPoint is committed to providing best-in-class technical support.

In order to provide you with the most up to date information as quickly as possible, we recommend the following procedure:

1. Update to the latest software. For the latest versions of OptiTrack software, drivers, and SDK samples, please visit our downloads section:

<http://www.optitrack.com/downloads/>

2. Check out the OptiTrack FAQs:

<http://www.optitrack.com/support/faq/general.html>

3. Check the forums. Very often a similar issue has been reported and solved in the forums:

<http://forum.naturalpoint.com/>

4. Contact technical support:

Phone: 541-753-6645

Fax: 541-753-6689

Email Form: <http://www.optitrack.com/contact/>

Mail: NaturalPoint, Inc.
P.O. Box 2317
Corvallis, OR 97339