RTC3D protocol

This defines the protocol used in First Principles and WaveFront to stream data through TCP/IP.

Table of contents

1 Introduction

The RT Server software is used to provide real-time capture data from motion capture equipment to third party applications interested in data rather than in operating the equipment itself. The real time processing results can be retrieved through a TCP/IP connection in real time. This document describes the protocol used in that TCP/IP connection.

2 How to connect to the RT Server

The RT Server is set up to listen to TCP/IP port on the computer it is running on. NDI First Principles has a server listening at port 3020 and NDI Wave Front at port 3030 as default.

2.1 Nagling

The TCP protocol by default uses a performance improvement called Nagle's algorithm that reduces the bandwidth used by the TCP connection. In the case of a real time server that sends small amounts of data in each frame, this algorithm should be turned off. Otherwise the server (and client) will wait to fill a full TCP packet, or until the previous packet has been acknowledged by the receiver, before sending it to the client (or the server).

3 Architecture compatibility

The RT Server should be able to communicate successfully with clients from any computer architecture. To avoid problems, there are two things that need to be considered: byte order and structure packing.

3.1 Byte order

The byte order of data pieces larger than one byte is very important. Different computer architectures use different byte orders. Windows (or more correctly computers based on Intel x86 processors) use Little-Endian (which means that the least significant byte comes first) while most others use Big-Endian. Since the IP protocol (which TCP is based on and all internet traffic and most network traffic is built on) is available on most architectures, data sent over TCP/IP must have a specified byte order. The byte order used by most TCP/IP-based protocols is called network byte order and it has been defined to be Big-Endian. The RT Server should follow this standard but also allow

for limited use of Little-Endian byte order to facilitate the use of the protocol for Windows clients. See the 4.3 Byte order section.

3.2 Structure packing

Many processors require that data is aligned on certain boundaries in memory. This is normally taken care of by the compiler of a computer program. The RT Server uses packing 4 .This means, for example, that if an array of structs is transferred between the server and the client, and each struct is made up of a 32-bit int and an 8-bit int, the size of each element in the array will be 8 bytes, not 5. The specification of the protocol is very specific about the exact layout of binary data sent between the server and the client.

3.3 Floating point values

The floating point type used by the RT Server is defined by IEEE 754 – single precision 32 bit. It exceeds the level of noise of 3D and analog data being transferred if right units are used. (at the range of ~15000 mm (mV) the maximum float error is ~0.4 micro meter (micro Volt)).

4 Basic protocol

This section describes the basic-level protocol used when communicating with the RT Server.

4.1 General-purpose data packets

All transmissions between the server and the client take place inside general-purpose data packets. Each packet starts with a header with two fields: a size field and a type field. Then the data of the specified type and size follows the header. To read data transmitted from the server, first read the four-byte size field and then the remaining number of bytes specified by this field.
NB: The Size field specifies the size of the whole packet including the size of the Size field itself.

General-purpose data packet header:

Size in bytes	Name	Description

4	Size	The total size of the data packet including these four bytes denoting the size.
4	Type	The type of data in the packet

After the header follows the actual data of the packet:

Size – 8	Data	Whatever data that the Type field says it is.

NB: A general-purpose data packet sent to or from a RT Server is not a type of TCP data packet. TCP is defined as a data stream. RT Server data packets are part of the RT Server protocol defined on top of the TCP stream.

4.2 Data types

The Type field of the general-purpose data packet header is a number that should be interpreted according to the table below. These are the data types that are defined in the protocol so far.

Type no	**Name**	**Description**
0	Error	The last command generated an error. The error message is sent as a string of ASCII characters
1	Command string / Command succeeded	A command sent to the server, or a response from the server to a command indicating that the command was successful. This is a sequence of single-byte ASCII characters.
2	XML	Data sent by the server in the form of XML, or data sent to the server in the form of XML. This too is a sequence of single-byte ASCII characters, but the sequence should be parsed as XML.
3	Data frame	One sample of real time data sent from the server. The contents of the frame may vary depending on the commands/settings sent to the server.
4	No Data	This packet type has no body. It indicates that a measurement has finished or is not yet started.
5	Complete C3D file	A complete C3D file sent from the server.

4.3 Byte order

As explained above the default byte order for data sent in the protocol is Big-Endian. There are, however, some kinds of data that are always Little-Endian

(Complete C3D file). The byte order of the Data frames, on the other hand, can be set by the client to the preferred byte order (by using the command SetByteOrder). And the byte order of strings and XML data is not affected by byte order at all since the data is single-byte.

To summarize, here is a table of the byte orders followed by the RT Server protocol:

Item	Byte order	Comments
General-purpose data packet header	Big-Endian	ALWAYS Big-Endian, even if the contents of the general-purpose data packet use Little-Endian byte order.
Strings and XML data types	Not affected	Since the strings in the protocol are defines as single-byte ASCII characters, they are not affected by byte order issues. You can't order a single byte.
Complete C3D file data type	Little-Endian	NB: The Size and Type fields of the header are still Big-Endian!
All other data types	Can be changed by the client Default: Big-Endian	By sending the SetByteOrder command, the byte order can be changed.

5 Commands

5.1 Basic protocol for commands

A command is sent in a data packet of type string (as seen above). The command strings may or may not be null-terminated. The layout of a typical command sent to the server is shown in the table below. The table includes the general-purpose data packet header:

Size in bytes	Name	Value
4	Size	20 (4+4+12) – Laid out like this (ALWAYS Big Endian).

<table>
<tr><td></td><td></td><td>
<table>
<tr><td>Byte</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>Value</td><td>0</td><td>0</td><td>0</td><td>20</td></tr>
</table>
</td></tr>
<tr><td>4</td><td>Type</td><td>1 – Laid out like this (ALWAYS Big Endian).

<table>
<tr><td>Byte</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>Value</td><td>0</td><td>0</td><td>0</td><td>1</td></tr>
</table>
</td></tr>
<tr><td>12</td><td>Data</td><td>“Version 1.0” – laid out like this (with a NULL char to terminate it, which is not required).

<table>
<tr><td>Byte</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td></tr>
<tr><td>Value</td><td>V</td><td>e</td><td>r</td><td>s</td><td>i</td><td>o</td><td>n</td><td>\32</td><td>1</td><td>.</td><td>0</td><td>\0</td></tr>
</table>
</td></tr>
</table>

5.2 Syntax

- A command contains no space characters.
- A command can have zero or more parameters, all of them space-separated.
- Parameters contain no space characters either.
- Commands and parameters are case insensitive.

5.3 List of available commands

In the description of the commands, number parameters are designated by an *n*, optional parameters are designated by enclosing brackets [] and choices between possible values are designated by a slash /.

5.3.1 Version

Parameters: n.n

The first thing that a client should do after connecting to the RT Server is to send the string “Version 1.0” to the server. This will ensure that the protocol described in this document is followed by the server.

Example: “Version 1.0”

5.3.2 SetByteOrder

Parameters: BigEndian / LittleEndian

The second thing a client should do after connecting to the RT Server is to change the byte order (unless the client uses Big Endian data, that is).

NB: Only some of the data sent by the server is affected by this setting. Most importantly, the general-purpose data packet header is never affected by it. See the 4.3 Byte order.

Example: "SetByteOrder LittleEndian"

5.3.3 SendParameters

Parameters: [All] [General] [3D] [Analog] [Force] [6D] [Events]

This command retrieves the settings for the requested component(s) of the RT Server in XML format. The XML parameters are described in the 6 XML parameters section.
As default, it sends 'All' data if not specified otherwise.

Example: "SendParameters 3D Force"

5.3.4 SendCurrentFrame

Parameters: [All] [3D] [Analog] [Force] [6D]

This command returns the current frame of real time data from the server.
Points worth noting are:

- The frame is composed of the parts specified in the parameters to the command. The exact layout of the data frame in different situations is described in the *7* Data frames section.
- If there is no ongoing measurement (either it has not started or it has already finished), a special type of data frame is sent to the client (see the *7* Data frames section).
- If a measurement is ongoing but there is no new frame of data available, the server waits until the next frame of data is available before sending it to the client.
- As default, it sends 'All' data if not specified otherwise.

Example: "SendCurrentFrame 3D Analog"

5.3.5 StreamFrames

Parameters: FrequencyDivisor:n / Frequency:n / AllFrames [All] [3D] [Analog] [Force] [6D] [Events]

or

Parameters: Stop

This command makes the RT Server start streaming data frames in real time. Points worth noting are:

- Each frame is composed of the parts specified in the parameters to the command. The exact layout of the data frame in different situations is described in the *7* Data frames section.
- The rate at which the frames are sent depends on several factors:
  - The **measurement frequency** used when acquiring the slowest data frame component specified to be included in the transmitted frames.
    The transmission rate cannot be greater than this frequency.
  - The **real time processing frequency** set on the server. This may differ greatly from the measurement frequency. For example the server may be measuring at 1000 Hz but trying to calculate real time frames only at 50Hz. The transmission rate cannot be greater than this frequency either.
  - The **processing time** needed for each frame of data on the server. This may also be a limiting factor – The server may not have time to process and transmit frames at the rate specified as the real time processing frequency.
  - The **frequency specified by the client**.
    The client has three ways of specifying the preferred data rate of the server. If the client specifies a higher rate than it can handle in real time buffering will occur in the TCP/IP stack at the client side and the client will experience lagging.
    - **FrequencyDivisor:n**
      With this setting, The RT Server transmits every n:th processed real time frame to the client. Please note that this may not be the same as every n:th frame of the measurement (se real time processing frequency above).
      Example: The RT Server is measuring in 200 Hz and real time tracking in 100 Hz. If a client specifies "FrequencyDivisor:4" the server will send data at a rate of 25Hz.
    - **Frequency:n**
      With a specific frequency setting, the RT Server will

transmit frames at a rate of approximately n Hz. Example: The server is measuring in 200 Hz and real time tracking in 100 Hz. If a client specifies "Frequency:60" the server will send data at an approximate rate of 60Hz. This means that usually every other frame is transmitted, but once in a while two frames in a row are transmitted (to reach 60Hz instead of 50).

  - **AllFrames**
    When a client specifies AllFrames in the StreamFrames command, every real time frame processed by the RT Server is transmitted to the client.

- When the measurement is finished, or has not yet started, a special data frame signaling that no data is available, is sent to the client.
- To stop the data stream before it has reached the end of the measurement or to prevent data from being sent if a new measurement is started after the first was finished, one can send the StreamFrames command again, specifying the Stop parameter (ie "StreamFrames Stop").
- As default, it sends 'All' data if not specified otherwise.

Example: "StreamFrames Frequency:30 3D Analog"

5.3.6 Bye

This command lets server know the client is disconnecting.

6 XML parameters

XML is used to exchange parameters between the server and the client. This has several benefits, including extensibility. Clients should not assume that the XML sent by the server looks exactly like the examples below – there may be any number of other items included as well, and there may be differences in white-space etc, but if you use a standard XML parser to look for the items you are interested in this will not be a problem.

When requesting more than one type of parameters at the same time, all of them are placed in the same `<RT_Parameters Ver='1.00'>` block. Their order is not defined.

The command "SendParameters" without any parameters sends all parameters described below.

6.1 General parameters

In response to the command “SendParameters General” the RT Server will reply with the following string of XML:

```
<RT_Parameters Ver=’1.00’>
    <General>
         <Server>
               <Name></Name>
               <Ver>1.02.03</Ver>
               <IPadd>192.168.0.20</IPadd>
               <Port>12345</Port>
               <Stats>
                  <FramesSent>123456</FramesSent>
                  <FramesPerSec>33.12</FramesPerSec>
               </Stats>
         </Server>
    </General>
</RT_Parameters>
```

The general parameters respond with the basic information on server and its average streaming performance during a session.

6.2 3D parameters

In response to the command “SendParameters 3D” the RT Server will reply with the following string of XML:

```
<RT_Parameters Ver=’1.00’>
    <The_3D>
        <Frequency>60.00</Frequency>
         <Unit>mm</Unit>
         <Markers>
               <Marker id = ‘1’>
                     <Label>marker 1</Label>
                     <Description> </Description>
               </Marker>
        </Markers>
    </The_3D>
</RT_Parameters >
```

It assumes all markers will be measured in the same units. The default unit is ”mm”.

The labels are the names of the markers in the current measurement.

Note: XML element names can’t begin with a number, that’s why the element for 3D parameters is called *The_3D*.

6.3 6D parameters

Some systems can provide more accurate position of a rigid body tool than that computed on the basis of marker positions alone. It would be beneficial to be able to exploit such feature by being able to directly read the transformation of such tools in addition to 3D information of contained markers.

```
<RT_Parameters Ver='1.00'>
<The_6D>
   <Frequency>60.00</Frequency>
   <Tools>
         <Tool id='1'>
               <Label> </Label>
               <Description> </Description>
               <Markers>
                     <Marker id='12'> </Marker>
                     <Marker id='14'> </Marker>
                     <Marker id='20'> </Marker>
               </Markers>
         </Tool>
   </Tools>
<The_6D>
</RT_Parameters>
```

Transformation data items correspond to tools listed in this section. Markers listed as part of a tool are referenced in the order as defined in a tool definition.

6.4 Analog parameters

In response to the command "SendParameters Analog" the RT Server will reply with the following string of XML:

```
<RT_Parameters Ver='1.00'>
    <Analog>
         <Channels>
               <Channel id = '1'>
                     <Label>EMG1</Label>
                     <Description> </Description>
                     <Unit>mV</Unit>
                     <Frequency>2400.00</Frequency>
               </Channel>
        </Channels>
    </Analog>
</RT_Parameters>
```

Analog data can be in general measured in different units per channel. The default unit is ”mV”.

6.5 Force parameters

In response to the command “SendParameters Force” the RT Server will reply with the following string of XML:

```
<RT_Parameters Ver=‘1.00’>
    <Force>
        <Plates>
        <Plate id = ‘1’>
            <Frequency>2400.00</Frequency>

             <Label>First force plate</Label>
             <Location>
                 <Corner id=‘1’>
                       <Unit>mm</Unit>
                       <Point>
                             <X>1000.00</X>
                             <Y>0.00</Y>
                             <Z>0.00</Z>
                       </Point>
                </Corner>
                <Corner id=‘2’>
                     <Unit>mm</Unit>
                     <Point>
                             <X>1600.00</X>
                             <Y>0.00</Y>
                             <Z>0.00</Z>
                     </Point>
                </Corner>
                <Corner id=‘3’>
                     <Unit>mm</Unit>
                     <Point>
                             <X>1600.00</X>
                             <Y>400.00</Y>
                             <Z>0.00</Z>
                     </Point>
                </Corner>
                <Corner id=‘4’>
                     <Unit>mm</Unit>
                     <Point>
                             <X>1000.00</X>
                             <Y>400.00</Y>
                             <Z>0.00</Z>
                     </Point>
                </Corner>
            </Location>

            <Origin>
                 <Unit>mm</Unit>
                 <Point>
                         <X>-4.00</X>
                         <Y>5.00</Y>
                         <Z>-40.00</Z>
                 </Point>
            </Origin>
            <Channels>
                <Channel id = ‘1’>
                     <Unit>N</Unit>
                     <Label>Force X </Label>
                </Channel>
                 <Channel id = ‘2’>
```

```
                        <Unit>N</Unit>
                        <Label> Force Y </Label>
                    </Channel>
                .
                .
                .
                <Channel id = ‘6’>
                        <Unit>Nm</Unit>
                        <Label>Moment Z</Label>
                    </Channel>
                <Unit>Nm</Unit>
                <Label>Moment 1</Label>
            </Channel>

                </Channels>
                <AnalogChannels>
                    <Channel id=’analog channel id’> </Channel>
                    <Channel id=’6’></Channel>
                    <Channel id=’7’></Channel>
                    <Channel id=’8’></Channel>
                    <Channel id=’8’></Channel>
                    <Channel id=’10’></Channel>
                <AnalogChannels>
                <ExcitationVoltage>5</ExcitationVoltage>
                <AmplifierGain>4000</AmplifierGain>
                <Calibration_Matrix>
                    <Unit>microV</Unit>
                    <Row id=’1’>
                        <Col id=‘1’>1.00</Col>
                        <Col id=‘2’>0.00</Col>
                        <Col id=‘3’>0.00</Col>
                        <Col id=‘4’>0.00</Col>
                        <Col id=‘5’>0.00</Col>
                        <Col id=‘6’>0.00</Col>
                    </Row>
                </Calibration_Matrix>
            </Plate>
            </Plates>
        </Force>
</RT_Parameters>
```

The parameters for force plates follow roughly the standard of the C3D file format (www.c3d.org). The server is expected to transform all force plate signals to the force and moment values and there is no need for any conversion on the client side. If the client is interested in raw analog data from force plate, *AnalogChannels* section contains id’s of the corresponding analog channels.

6.6 Events Parameters

In response to the command “SendParameters Events” the RT Server will reply with the following string of XML describing what types of events it is ready to share with a client:

```
<RT_Parameters Ver=’1.00’>
   <Events>
          <Event id=’0x123456’>
```

```
                    <Label>Button</Label>
                    <Description>Button action</Description>

                    <Params>
                       <Param id='1'>
                          <Type>int</Type>
                          <Description>Button id number
                          </Description>
                       </Param>

                       <Param id='2'>
                          <Type>int</Type>
                          <Description>ON/OFF state</Description>

                          <ExpValues>
                            <ExpValue id='1'>
                               <Value>1</Value>
                               <Label>ON</Label>
                               <Description>Button was pressed
                               </Description>
                            </ExpValue>

                            <ExpValue id='2'>
                               <Value>0</Value>
                               <Label>OFF</Label>
                               <Description>Button was released
                               </Description>
                            </ExpValue>
                          </ExpValues>
                       </Param>
                    </Params>

            </Event>
      </Events>
</RT_Parameters>
```

The event is identified by its ID unique for the particular server manufacturer only. The event can contain up to 3 optional parameters to further describe the event. Event Parameter section's role is to allow client user to see what kinds of events are available and select its own actions for those hshe's interested in.
Parameter uses 4 bytes interpreted as either (signed) "int" (default), "float" or "string" (4 characters long, no termination '\0').

7 Data frames

Each data frame is made up of zero or more components, as specified in the commands SendCurrentFrame or StreamFrames. The frame starts with a Count field that specifies the number of components in the frame. Every component starts with a general component header – identical to the general-purpose packet header described above.
NB: The general component header – just like the whole data frame – follows the byte order specified by the client through the SetByteOrder command.

7.1 General data frame layout

Data frame layout:

Size in bytes	Name	Description
4	ComponentCount	The number of data components in the data packet.

Repeat ComponentCount times:

4	ComponentSize	The size of the component including the ComponentType, ComponentSize, FrameNumber, padding fields. Note: Byte order depends on what was requested by SetByteOrder.
4	ComponentType	The type of the component. Defined below. Note: Byte order depends on what was requested by SetByteOrder.
4	FrameNumber	The number of this frame.
8	TimeStamp	microseconds from start
Size – 20	ComponentData	Component-specific data. Defined below.

7.2 Data frame component types

The ComponentType field of the data component header is a number that should be interpreted according to the table below. These are the data frame component types that are defined in the protocol so far.

Type no	Name	Description
1	3D	3D values
2	Analog	Analog values
3	Force	Force values
4	6D	Transformation values
5	Event	IDs and parameters of events

7.3 The 3D data frame component

The markers of the 3D data always follow the labels of the 3D parameters. The same number of markers are sent each frame, and in the same order as the labels of the 3D parameters. If a marker is missing from the frame, its X, Y and Z coordinates will have all their 32 bits set – this signifies a negative quiet Not-A-Number according to the IEEE 754 floating point standard.

Size in bytes	Name	Description
4	MarkerCount	The number of markers in this frame.

Repeated MarkerCount times:

4	X	X coordinate of the marker, 32-bit float.
4	Y	Y coordinate of the marker, 32-bit float.
4	Z	Z coordinate of the marker, 32-bit float.
4	Reliability	Residual

7.4 The Analog data frame component

Size in bytes	Name	Description
4	ChannelCount	The number of channels in this frame.

Repeated ChannelCount times:

4	Voltage	The voltage of the channel in this frame.

7.5 The Force data frame component

Size in bytes	Name	Description
4	PlateCount	The number of force plates in this frame.

Repeated MarkerCount times:

4	FX	X coordinate of the force, 32-bit float.
4	FY	Y coordinate of the force, 32-bit float.
4	FZ	Z coordinate of the force, 32-bit float.
4	MX	X coordinate of the moment, 32-bit float.
4	MY	Y coordinate of the moment, 32-bit float.
4	MZ	Z coordinate of the moment, 32-bit float.

X, Y, Z of the force application point are calculated based on the force, the moment and the force plate position so it is wasteful to stream this higher level information.

7.6 The 6D data frame component

Size in bytes	Name	Description
4	ToolCount	The number of tools reported in this frame.

Repeated ToolCount times:

4	Q0	Quaternion rotation q0, 32-bit float
4	Qx	Quaternion rotation qx, 32-bit float
4	Qy	Quaternion rotation qy, 32-bit float
4	Qz	Quaternion rotation qz, 32-bit float
4	X	X coordinate of the translation, 32-bit float.
4	Y	Y coordinate of the translation, 32-bit float.
4	Z	Z coordinate of the translation, 32-bit float.
4	Error	RMS marker fit to rigid body error

Transformation is composed of quaternion rotation and translation. Quaternion rotation vector should is normalized $|Q| = 1$.

7.7 The Event data frame component

Events are sent by server. Their meaning is defined by manufacturer in the 6.6 Events Parameters section. If the client doesn't recognize the event, it should ignore it.

Size in bytes	Name	Description
4	EventCount	The number of events reported in this frame.

Repeated EventCount times:

4	ID	integer ID of the event
4	param1	Optional event parameter (0xFFFFFFFF if unused is expected)
4	param2	Optional event parameter (0xFFFFFFFF if unused is expected)
4	param3	Optional event parameter (0xFFFFFFFF if unused

		is expected)

Since each server can use its own event definitions, the client user will first have to review the Event parameters sent by the server and assign them to some client actions