TCP

General

This is a plug-in for all kinds of TCP data Transfer. It acts as a **Server** or as a **Client** within the Transmission Control Protocol. The TCP channel allows simultaneous, bidirectional exchange of **Text** or **Buffer** Content. These two data structure can carry almost any Quest3D content by using the standard channels TextFilter, LoadBufferIntoChannel, and SaveBufferToChannel. Multiple instances of TCP are OK. You may setup more than one connection.

As the underlying TCP transportation is stream oriented, some measure had to be taken to support arbitrary long application data packages. We support two Operation Modes addressing different application scenarios:

1. **Plain** mode refers to single Payloads of up to 1 KB with any application that supports TCP. The typical application scenario is a CSV encoded short message.
2. The **Marshalled** mode is available between Quest3D applications only. Here the size of a data package is only bound by the Quest3D application it selves. With this mode the channel reassembles transport stream data fragments up to the original size. A typical scenario is a video stream, where each video frame is reassembled automatically from dozens of single IP packages. In addition, Marshalled mode allows the exchange of an **applicationID** independent from the payload data transfer. This can be used to signal the recipient a processing instruction, e.g. to differentiate between Texture and 3D Object Data processing, or to implement flow control.

Children

The children can be grouped into three categories: Connection and transmission handling (child 0 to 3), sending (4 to 8), and receiving (9 to 12).

Position Name	when used?	Remarks
0 role	init	0 client marshal (default), 1 server marshal, 2 client raw, 3 server raw
1 IP name	start service	-name or address-
2 port	start service	
3 socket error details	asynchronous	
4 dataType	CallChannel	0 text (default) , 1 buffer
5 applicationId send	CallChannel	-marshalling only-
6 buffer send	CallChannel	
7 text send	CallChannel	
8 # Bytes sent	CallChannel / GetFloat	
9 applicationId received	asynchronous	-marshalling only-
10 buffer received	asynchronous	
11 text received	asynchronous	
12 # Bytes received	asynchronous / GetFloat	

Description

The cannel is operated by using three different parent channel types: a call with a **SetValue** parent controls the channel state. A simple call with **CallChannel** call initiates a transmission of text or buffer data. And a call with a **Value** parent detects incoming messages and gives further feedback. The value calls are mandatory, they link the asynchronous receives with the Quest3D main loop.

The channel state reflects basically the TCP state:

-2: trial expired (trial version only)
-1: no TCP service or failure,
0: waiting for connection (server side only)
1: TCP connection established, ready for transmission
2: Content was sent
3: Content was received
5: Sending and receiving occurred

The basic commands to control the channel state are:

1: start service (connect – server only)
0: disconnect (client only)
-1: stop service

To operate the TCP channel, choose its role (server or client) and the operation mode at the beginning. Provide IP name or address and the port number. Now you can start the TCP service explicitly. Start the server application before you attempt to get the client running. If TCP setup for the server succeeds, the state changes into 0. If the client setup succeeds, its state changes to 1, and the corresponding server changes its state to 1. If any error occurred during TCP setup, child 3 contains the error code. Refer to Windows Socket Errors for details.

If the setup is passed successfully, you are ready to exchange application packages. Any previous content in the buffer and text children is cleared initially. In this phase there are no differences between the server and the client role: Anyone can start a transmission at any time, or terminate a connection. To initiate sending, you have to provide content in either the text or the buffer channel, and set the data type child accordingly. In case of plain transfer, the valid payload size is 1 Byte to 1 KB, in case of marshalled mode 1 Byte to any acceptable size from the application. The TCP channel has been tested to handle image sizes of more than 10 M Pixels. The limitation to text and buffer data types does not impose any restrictions: Use text if you are exchanging plain ASCII or CSV techniques (see Quest3D channel “TextFilter”). Generate Buffer content with the channel “SaveChannelToBuffer”, return with “LoadBufferIntoChannel”. In addition the marshalled mode allows the transfer an applicationID. Use this ID to differentiate between different processing steps, or to initiate a command (e.g. to erase the remote screen). The assignment of IDs is up to the application. Use a 1 Byte text placeholder, if you are only interested in sending commands.

The receiving side delivers payload content in the text or buffer channel. In plain mode, the decision is made according to the send data type chosen in child 4. In marshalled mode, this is done automatically, in accordance with the sender decision. The receive channels are filled asynchronously. This means, independent from the main Quest3D execution loop. Use the Value access to TCP to find out, if any receiving occurred since the last call. Then start the final processing of this incoming data accordingly.

Coalesce operation policy: A sender may coalesce multiple messages and delivers them in one chunk. In marshalled mode operation, this is detected. The following rule applies: Only the first application message is processed, the rest is purged[1].

TCP connections can be terminated by either side with the command -1. The server returns to state 0, the client to state -1. You are now able to start the same connection again or provide a new set of connection parameters. In order to reject incoming TCP connection requests, the server has to be put into state -1 explicitly.

Tips

- Multiple TCP channels in one application are OK. Use different port numbers to avoid binding conflicts.
- Handle firewalls beforehand.
- For testing purposes I recommend the following small utilities: Hercules http://www.hw-group.com/products/hercules/index_en.html and Comm Echo http://www.serialporttool.com/CommEcho.htm
- Choose marshalled mode, if you are using Quest3D on both sides. This is the preferred scenario. Choose plain mode, if you are sure that your payload does not exceed 1 KB in size, and if the partner application is not based on Quest3D.
- TCP starts in disonnected mode. Even with a small scenario, you must apply commands to get it running.
- Remember, that all text and buffer channels are erased during setup. Make sure you fill them only after setup completion.
- The value calls to a receiving channel are mandatory. And are typically used with an if-channel to activate processing of the received data material.
- Text always includes a terminating NULL.
- Avoid coalesce situations.

Legal note: Permission granted for evaluation and educational purposes only (trial version), commercial use requires an explicit license agreement.

Volume, runtime duration, and expiration restrictions are enforced with trial licenses. Details can be found by reading the output of the Debug Window, and on my web page. Published projects are subject to occasional random termination. Be prepared to cope with this situation in your application.

Supported Quest3D Versions

- 4.3.2
- 5.0 (x64), 32 on request only

[1] This ensures stable operation. You can detect this situation by monitoring child 3: It is (persistently) set to 999. It is the applications responsibility to enforce none-coalesce operation: This can be done by different means, including: (a) by adapting flow control via AppID exchange. (b) Padding sender side packages to exceed MTU size.

Known Problems/ Limitations

- Make sure that your applications have ample processing resources, if you are sending continuously. Or implement your own application flow control scheme using applicationId. Otherwise, if package receive rates exceeds their processing rate, the application may crash.
- Reload in the editor may fail to clear all sockets system states. Allow for more time before reusing these system resources.

Revision History

2014 01 09 Maintenance update: Short text in marshalled mode accepted, Coalesce policy enforced.

2012 11 17 initial release, replaces TCPserverText and TCPclientText, substitutes the legacy Quest3D socket channel family.

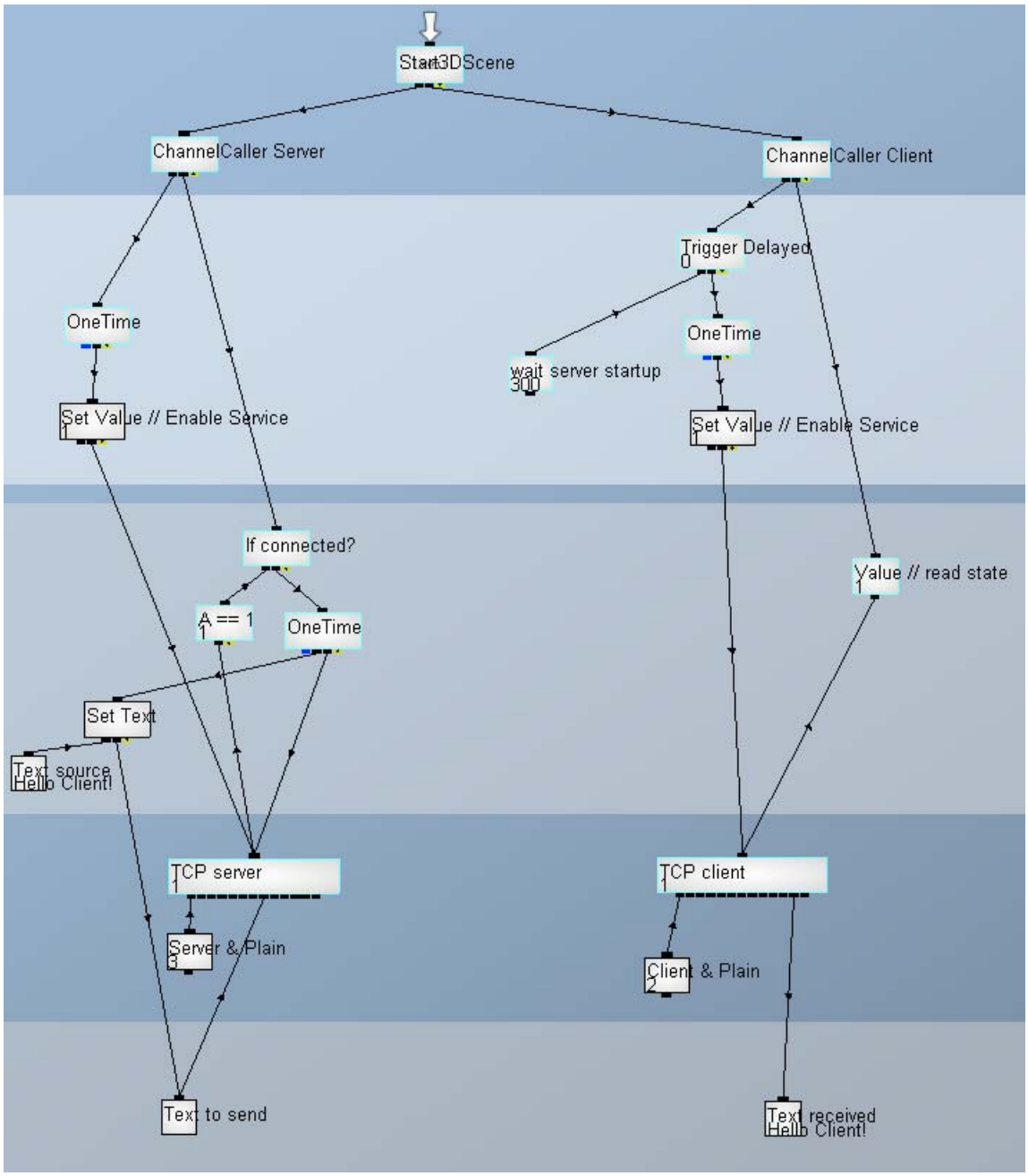
Contact: quest3d.godbersen.eu

Tutorial

We explain a simple scenario based on text exchange first. This should familiarize you with the basic setup and transfer operations. The second scenario is more elaborate. It describes a marshalled transfer of textures.

1. Simple TCP text transfer

File: `MiniTCPtextTransfer-432.cgr`. The goal is to transmit one short text from a sender (here a server) to a receiver (client) via TCP with minimal effort. For simplicity, we locate the server and the client in one application. The screenshot shows the state after completion of the transfer.



The left side is the server part. We have to start the service once with a SetValue of +1. We monitor the server state. If a connection is established (state is equal to 1), we prepare a text to send (SetText) and call the TCP channel one time. Our TCP server need two children: the role as server for plain data transfer is chosen with the value 3. The Text child contains a temporary copy of the text to send. This text channel is erased after completion.

Both sides make use of default settings IP 127.0.0.1, Port 5000, Data Type 0 (Text).

The client side has to wait for the server to become operational, here simulated with a 300 msec delay. An invocation of the SetValue channel starts the connection establishment. A call via a Value channel is necessary for proper operations in a production environment. The role child is set to 2, indicating a client with plain transfer mode. The text received is filled automatically, as a TCP package arrives. It will be eventually overwritten by future receives.

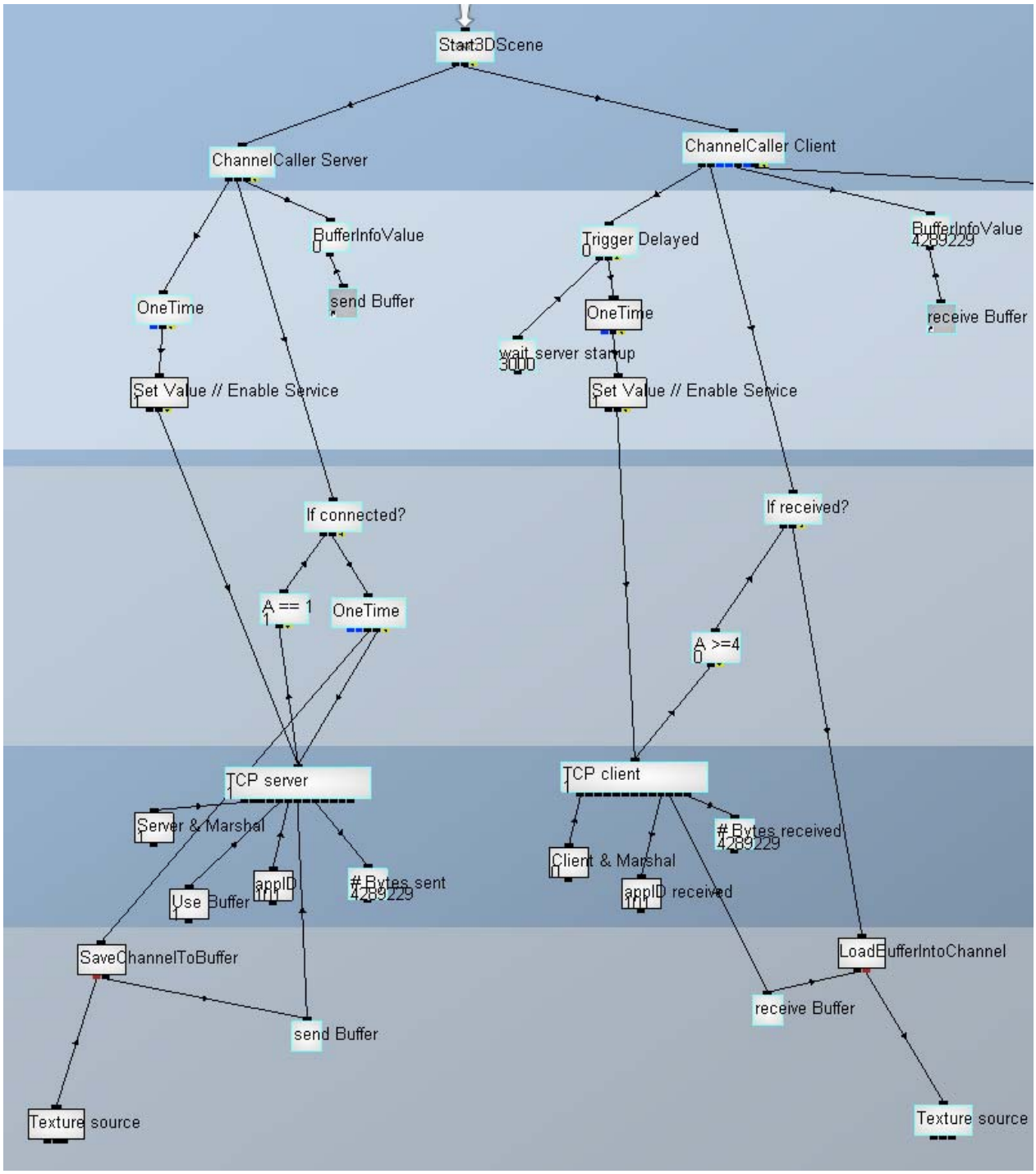
My suggestion for further investigations:

(1) Separate the server and client into different applications. Published versions require a visual feedback, e.g. via TextOut.

(2) Run this application in a distributed environment: Choose a host and find its IP name or address, e.g. 192.168.178.144, and add a text child to provide this information to both sides.

2. Marshalled TCP Texture Transfer

File: `MiniTCPtextureTransfer-432.cgr`. The goal is to transmit one large texture from a sender (here a server) to a receiver (client) via TCP with minimal effort. For simplicity, we locate the server and the client in one application. The screenshot shows the state after completion of the transfer.



The setup is basically the same as in the previous example. The role settings are different to reflect the marshalled transmission.

Observe at the sender side: Use buffer is set to 1 at the server to indicate that we want to transport the content of the buffer child, and the applicationID is set to 101. The send buffer is filled with the texture by SaveChannelToBuffer. The # Bytes send channel indicates that a chunk of more than 4 MB was transmitted.

The receiving side reflects the transmitted volume and the applicationID. LoadBufferIntoChannel copies the buffer content into the receiving texture. This is only done, when the TCP channel returns with the code 4 or higher, indicating that data was received and is ready for processing. Note, that even the source channel name “Texture source” is replaced.

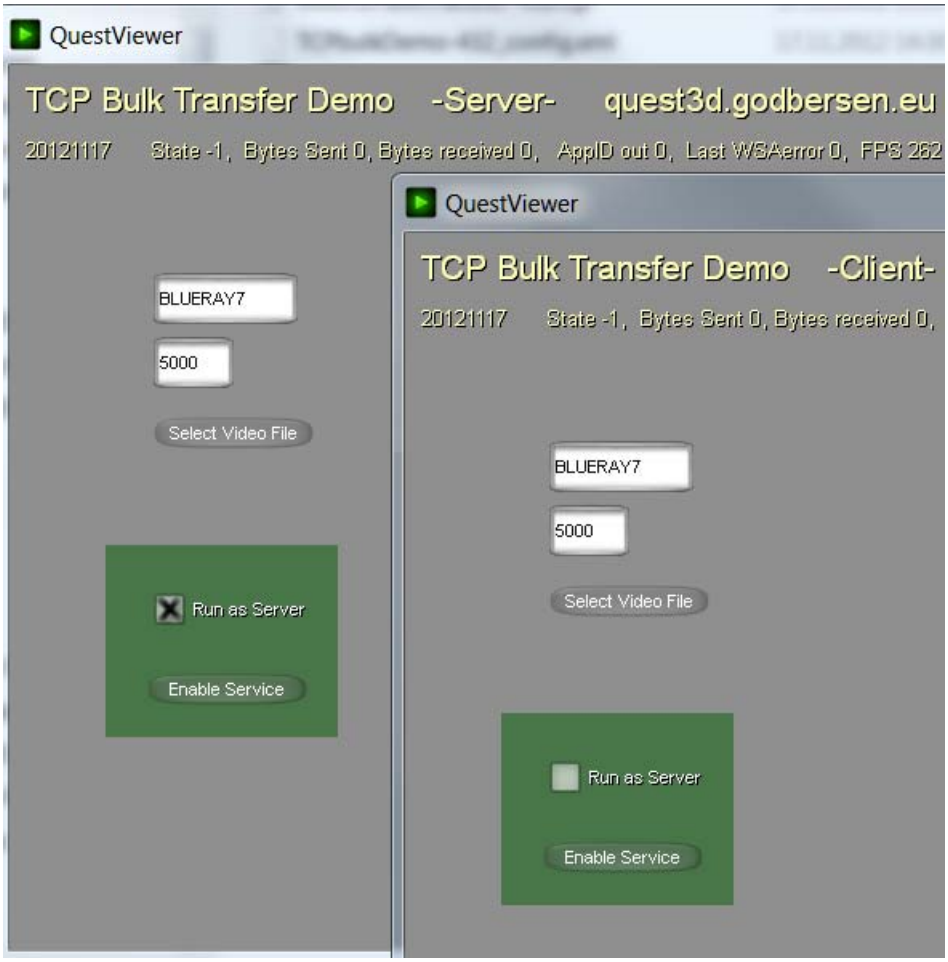
BufferInfoValue is a small utility plug-in to get the size of the buffer content. On the sender side, the buffer was erased after initiating the send process.

More background info: The original size of the texture I used is 18 Pixels (I replaced it in the meantime to lower the footprint). The size of the original jpg file is 4.288.700 Byte. The transmission volume was 4.289.229 Bytes. The difference is due to the Quest3D specific texture and buffer headers. On my system, about 800 LAN packets were used to complete the transport, the actual number is volatile.

Example 3: TCPbulkDemo

This application provides exchange of Text, Textures, Videos (file, webcam, or live screenshots), and 3D Object Data in marshalled mode. See the video https://vimeo.com/52993992. Start this application twice, either on one machine or on different PCs. Files: `TCPbulkDemoTrial-432.exe`, `TCPbulkDemo-432.cgr` (customers only).

A setup dialog appears:

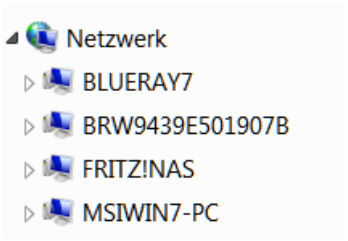


Select a source video file, if you wand to try video transmission. Any Video Quest3D accepts is OK. Set the IP address and port number.

In a local setup, user the IP name “loalhost” or the equivalent IP address 127.0.0.1 both for server and client.

In a distributed setup, you may use the IP address 0.0.0.0 for the server and the appropriate dedicated IP address of the server machine (e.g. 192.168.178.144) on the client side. The actual sever IP address can be obtained with the command “`ipconfig`” in a windows command shell.

Excursion: An easier way is to use names: For example, the Windows Explorer provides a list of accessible local computer names. In my case:

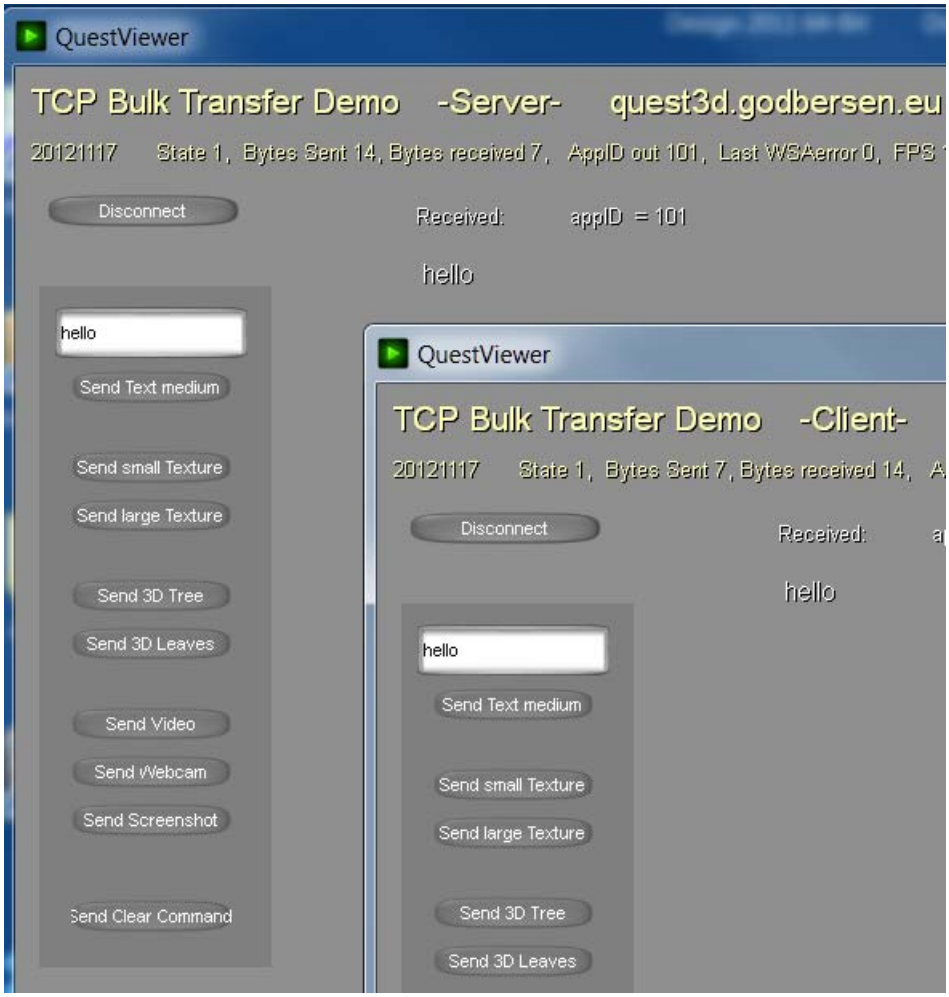


Imagine a scenario with a client located on Blueray7 and a server on MSIWIN7-PC: The entries would be then "MSIWIN7-PC on both sides.

Select "run as a server" in one application. Start the server first with the "Enable Service" button. Then start the client.

If a TCP connection problem occurred, correct the parameter. The reason of failure can be found in the top feedback line under "LastWSAerror". Refer to Windows Sockets Error Codes for details.

Otherwise you will be prompted with a transmission dialog:



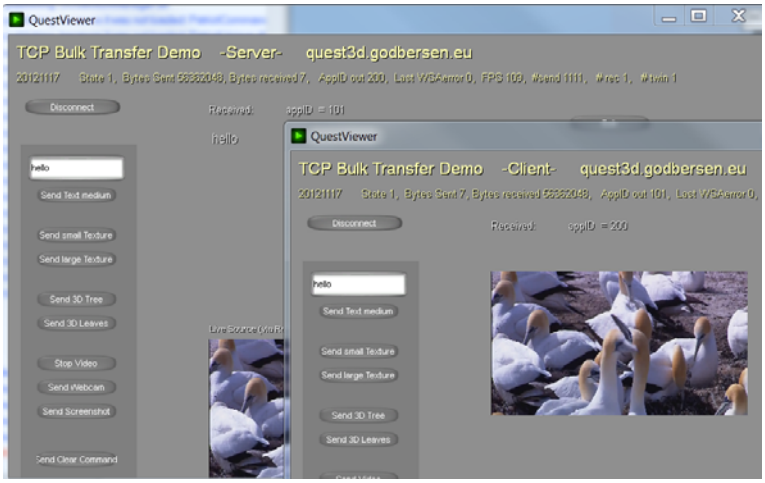
This dialog is the same on both sides. Any peer can start a transmission. To keep things simple, some content is predefined[2]. The screenshot shows the initial exchange of welcome text messages.

[2] The access to a Webcam may be unreliable.

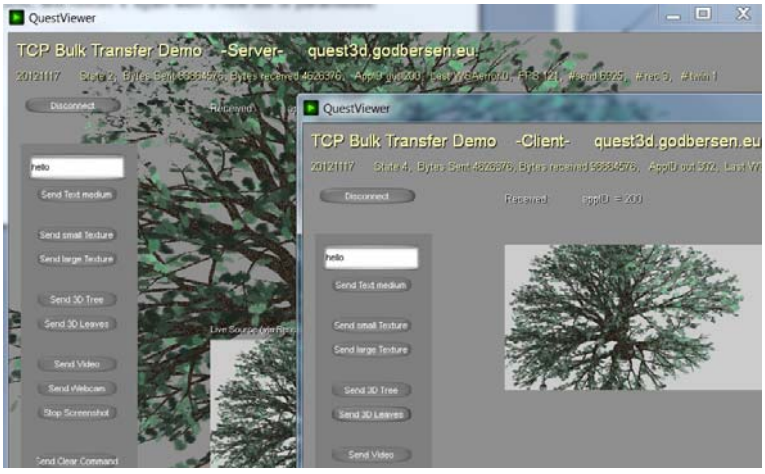
Please note: Even though the predefined content is available on each side for demo purposes, the actual display on the receiving side will be derived from the transmission. This is obvious if you consider the different video and text sources.

The biggest predefined content sizes are about 3 MB for the tree trunk, and the large texture has about 8 Mpixels.

The next screenshot shows the transmission of a video from the server to the client (shown in the lower part, to allow a visual comparison with the receiver video display). The Video resolution is reduced here (frame with 250 KB). However, the TCP channel it selves does not impose any restrictions.



Another scenario: After an initial transmission of 3D Object Data containing a tree trunk from the client to the server, the server returns a live screenshot of its actual rendering to the client.



You can stop the TCP service at any time, and restart it again with a new set of parameters.

Appendix A: Communication Ouside Quest3D

Here are some tips to setup an operation with applications outside the Quest3D realm. Basically, you can connect to any TCP enables application.

1. Decide your communication needs carefully.
2. Buffer transfer is only applicable to Quest3D resources that can be moved in and out of a Buffer channel.
3. Text transfer is very reliable, but may be subject to processing overhead.
4. If marshalled mode is necessary, you have to construct and evaluate my marshal header. Please contact me for more details and assistance.

Test Assistance Commands

DumpToFile

- SetValue 1014: Dump incoming and outgoing TCP payload content to file. This includes the marshalling header and any application paload. Stored in C:\temp*.raw with index number added to the filename. Viewed best with a HEX Editor. For debug purposes only.
- SetValue 1013: disable DumpToFile (default)

Marshal Header

Size 12 Bytes, send once.

numberOfBytesToSend	
payloadDataType	marshallingMode
applicationId	reserved

```
numberOfBytesToSend  (int)        # payload Bytes     at least 1
payloadDataType      (short)      0 or 1            Text / Buffer
marshallingMode      (short)      1                 must always be 1
applicationId        (short)      any
```

Example:: ASCII String “AB”, AppID 11

```
03000000000001000B000000414200
########tttmmmmiiiirrrr
```