

# TestMan Documentation

A framework for flexible inter-program communications.

Authors: Peter Neuhaus, Martin Danneberg

E-Mail: {peter.neuhaus, martin.danneberg}@ifn.et.tu-dresden.de

## 1. Introduction

TestMan consists of a .NET DLL which can be interfaced by many programming languages. Using this interface, it is possible to establish flexible communications between multiple programs. For the underlying communications, TestMan internally uses UDP and TCP packets. UDP is used for short messages, which are broadcasted through the network. For the exchange of large amounts of data between two endpoints, it is possible to use the TCP stream functionality of TestMan.

TestMan addresses clients using the following tuple: (*Type*, *ID*).

- *Type* allows you to specify the client by a number in the range [1,...,255].
- *ID* allows you to distinguish multiple clients of the same *Type*. Value in the range [1,...,254].

A special feature of TestMan is that the IDs are managed by the DLL. That means if you want to create a client with the Type-ID-Tupel (Type=2, ID=1) and there is already a client with the exact same configuration, the DLL will give the new client a new *ID*, which was previously unused/free. For example the DLL could tell the client which requested (Type=2, ID=1) to use (Type=2, ID=2), if ID=2 is free.

TestMan provides three different ways to transmit data

- Commands: Broadcasts key value pairs, which are acknowledged by 4 way handshake
- Data: Broadcast simple key value pairs, no acknowledgement
- TCP stream: Transmit data via TCP stream to a single destination (specified by Type, ID tuple)

## 2. Repository overview

Besides the actual TestMan DLL, the repository consists of additional software and examples on how to use TestMan. The following should provide an overview:

- The source code of the TestMan DLL is contained in ".\UDP-Communications"
- The TestMan TestSuite is contained in ".\UDP-TestSuite" (see section 3)
- The documentation is contained in ".\Documentation"
- ".\Examples" provides examples on how to use and integrate TestMan into C++, Python, MATLAB and LabVIEW
- Additional software is provided in ".\Helpers" and ".\Applications"

## 3. TestSuit

The TestMan TestSuit allows controlling and debugging the DLL. It provides useful information on what happens and it can be a useful tool to debug new TestMan applications. However, it is not required to use it, any TestMan program works also without running the TestSuit! We will use the TestSuite in order to demonstrate the TestMan functionality.

The TestSuite is located at “.\UDP-TestSuite\bin\Debug\UDP-TestSuite.exe”<sup>1</sup>. Running the “UDP-TestSuite.exe” yields:

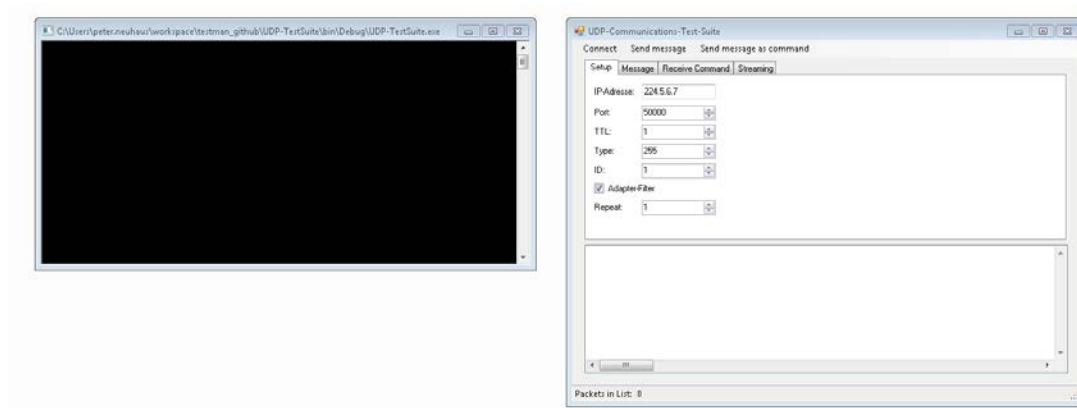


Figure 1: TestSuite consists of a Terminal and a GUI

For example we specify a Type=3 and ID=4 for the TestSuite:

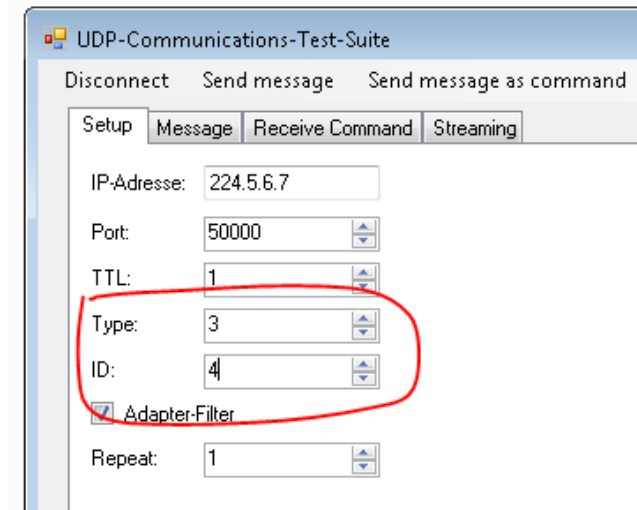


Figure 2: Define Type and ID of TestSuite.

In order to establish a connection between the TestSuite and the TestMan DLL click “Connect”:

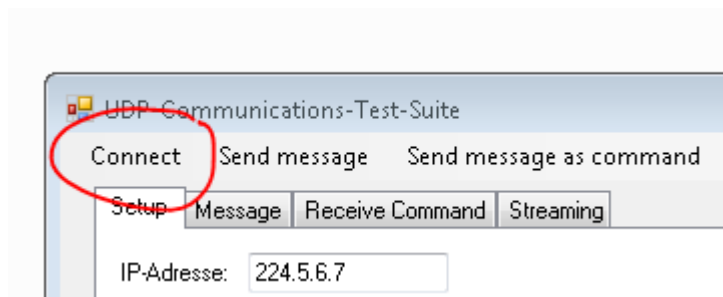


Figure 3: Connect the TestSuite to the TestMan DLL.

<sup>1</sup> Under Linux the .NET executable can be executed using the mono package, <http://www.mono-project.com/>.

The Terminal output should look like this<sup>2</sup>:

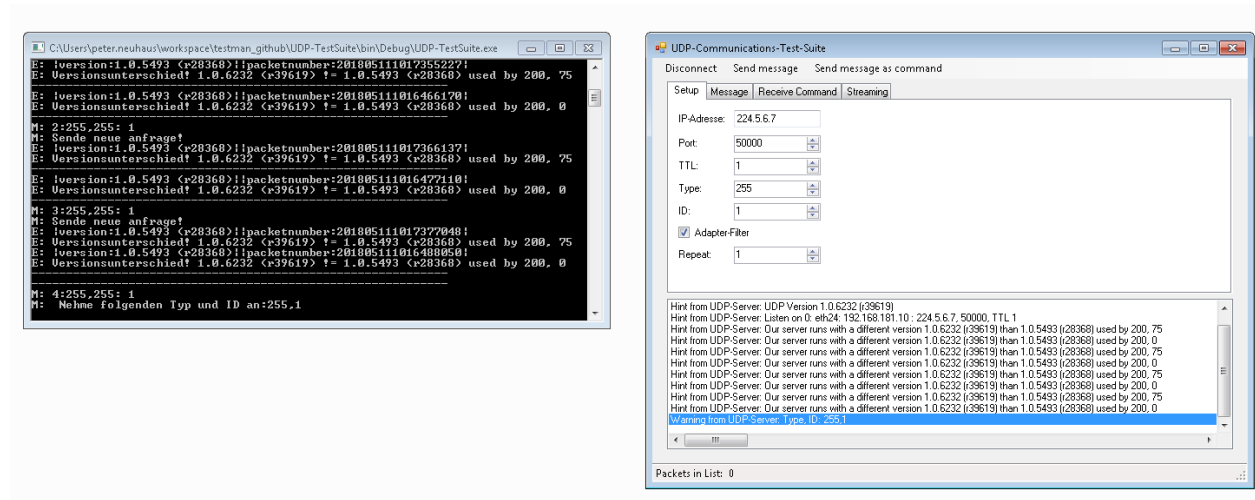


Figure 4: TestSuite Connected.

In order to show how to use the TestSuite, we will transmit simple messages between two TestSuits. Therefore, we firstly start a second TestSuite. To show the TestMan functionality we try to start the second TestSuite with the same Type and ID as the first (Type=3, ID=4). As we can see, TestMan provides the second TestSuite with the ID=5, because the ID=4 is already in use.

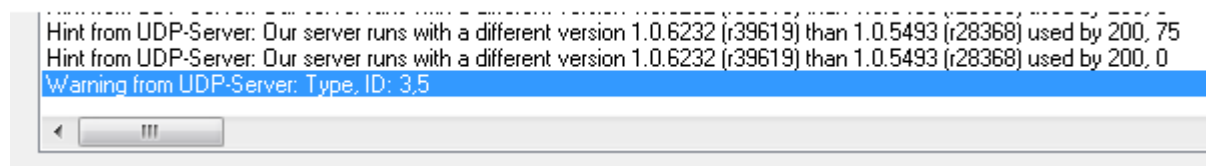


Figure 5: TestMan provides the client with an unused ID if the requested ID is already in use.

To broadcast a simple UDP message, we switch to the “Message” tab in the TestSuite GUI. And add the following (key, value) pair as content of the message:

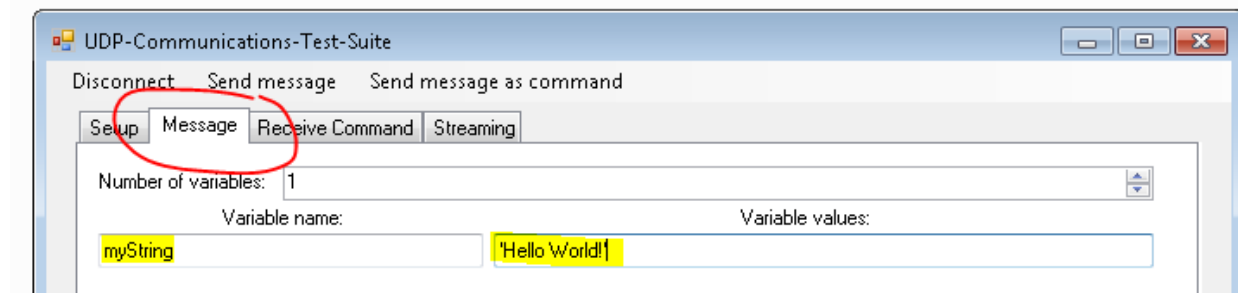


Figure 6: Broadcasting (key, value) pairs using the TestSuite GUI.

<sup>2</sup> Unfortunately, some of the outputs are in German; however, they are not that important.

The message is broadcasted if we click “Send message”:



Figure 7: Broadcast message via clicking “Send message”.

The first TestSuite receives the message from (Type=3, ID=5) in the GUI as well as in the Terminal:

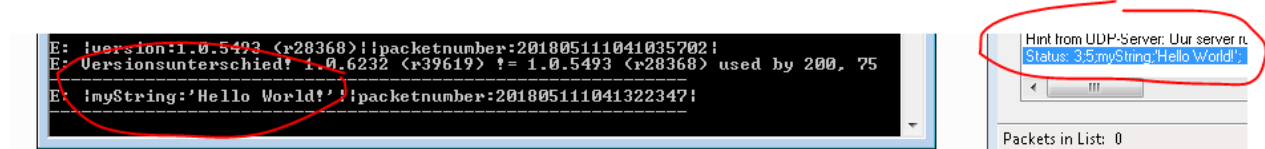


Figure 8: Received messages.

We can also transmit a file via a TCP Stream between two TestSuits. In order to do so we switch to the tab “Streaming”, enter the Type and ID of the receiver and click “Start stream”:

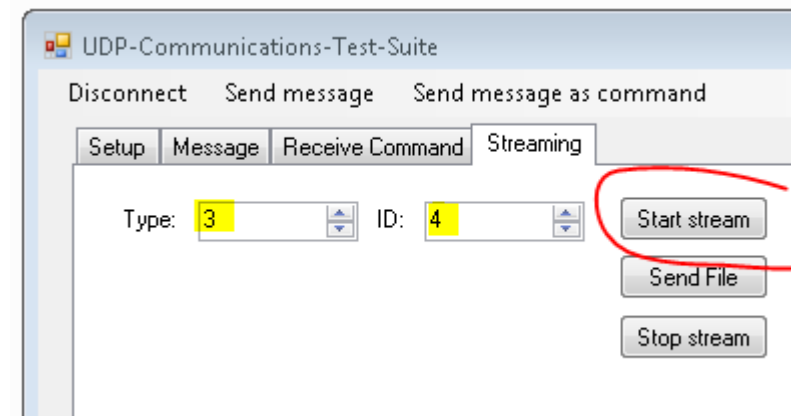


Figure 9: Starting a TCP stream with the TestSuite.

The terminal and the GUI should of the sender (Type=3, ID=5) and the receiver (Type=3, ID=4) should confirm, that the TCP stream has been established. Then by clicking on “Send File”, a dialog opens up, which lets you choose a file to transmit. Then at the receiver GUI a dialog should open, which lets you specify where you want to save the file.

The TestSuite contains additional functionality, which is not documented yet.

## 4. Examples

The repository comes with examples for

- C++
  - Requirements:

- - Boost 1.66 (<http://www.boost.org>). Might run with older versions. Mainly boost.asio is used.
- - C++14 compatible compiler. Has been tested with g++-7 and Visual Studio 2017
- - CMake3 (<https://cmake.org>)
- Additional information under “.\Examples\Cpp\notes.org” and “.\Examples\Cpp\readme.org”
- MATLAB
- Python
- LabVIEW (Comms)

Note that the C++ examples (“.\Examples\Cpp”) are quite different from the other examples. The .NET DLL cannot be integrated into C++. As a result, the code re-implements the functionality in C++ without making use of the DLL. **Also, note that the C++ part comes with its own documentation and readme under “.\Examples\Cpp\readme.org”! and “.\Examples\Cpp\notes.org”.**

### Simple example

Two very simple examples can be found under “.\Examples\Python\simpleExample.py” and “.\Examples\Matlab\simpleExample.m” for Python and MATLAB, respectively. The Python code looks as follows. It consists of four parts. Firstly, the TestMan Server is started. Secondly, the callback function, which handles received packets, is defined. Then the program transmits two commands, followed by the transmission of two messages (data).

```
def main():
    S = UDP_Server(ip="224.5.6.7", port=50000, ttl=1, type_=2, id_=1)
    print("Server started..")

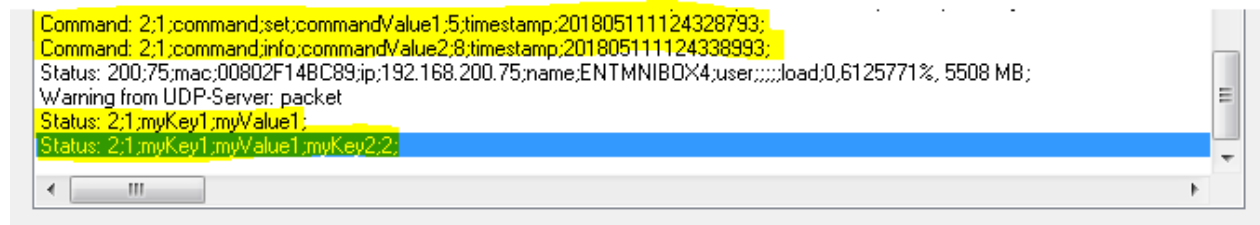
    # Connect to packet receive call back handle. The function 'received' is
    # executed every time a packet is received.
    S.packet_received.connect(Slot(received))
    print("Receive-Handler established")

    # Send commands.
    S.send_command("set", commandValue1=5)
    time.sleep(1)
    S.send_command("info", commandValue2=8)
    time.sleep(1)

    # Send data
    S.send_data(myKey1="myValue1")           # One key=value pair
    time.sleep(1)
    S.send_data(myKey1="myValue1", myKey2=2) # Two key=value pairs
    time.sleep(1)
```

Figure 10: Part of "simpleExample.py".

Running the simpleExample.py yields the following output at a TestSuite:



```
Command: 2;1;command;set;commandValue1;5;timestamp;201805111124328793;  
Command: 2;1;command;info;commandValue2;8;timestamp;201805111124338993;  
Status: 200;75;mac;00802F14BC89;ip;192.168.200.75;name;ENTMNIBOX4;user;load;0.6125771%; 5508 MB;  
Warning from UDP-Server: packet  
Status: 2;1;myKey1;myValue1;  
Status: 2;1;myKey1;myValue1;myKey2;2;
```

Figure 11: Data received at TestSuite.

## TCP Stream Example

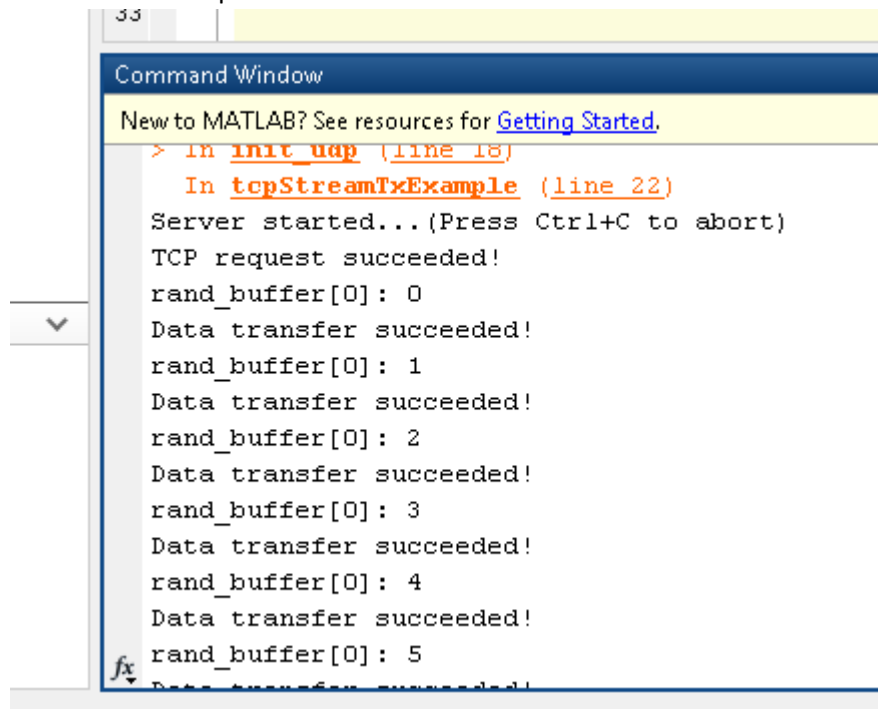
In the following, we demonstrate TestMan interoperability between MATLAB and Python. Firstly, we start the TCP stream receiver in Python using “python tcpStreamRx.py” from “.\Examples\Python”.



```
PS C:\Users\peter.neuhaus\workspace\testman_github\Examples\Python> python .\tcp  
StreamRx.py  
Server started...(Press Ctrl+C to abort)
```

Figure 12: Starting TCP stream transmitter in Python.

Secondly, we run the TCP stream transmitter “tcpStreamTxExample.m” from “.\Examples\Matlab” in MATLAB. The output should look like this:



```
33  
Command Window  
New to MATLAB? See resources for Getting Started.  
> In init_udp (line 18)  
In tcpStreamTxExample (line 22)  
Server started...(Press Ctrl+C to abort)  
TCP request succeeded!  
rand_buffer[0]: 0  
Data transfer succeeded!  
rand_buffer[0]: 1  
Data transfer succeeded!  
rand_buffer[0]: 2  
Data transfer succeeded!  
rand_buffer[0]: 3  
Data transfer succeeded!  
rand_buffer[0]: 4  
Data transfer succeeded!  
rand_buffer[0]: 5  
Data transfer succeeded!
```

Figure 13: MATLAB TCP Stream transmitter output.

The python receiver looks like this:

```
TestManLib 26.04.2018 14:34 File folder
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Receiving 10485760 Bytes...
10485760 Bytes received.
Length of data: 9
data[0:9]: [77, 231, 32, 233, 161, 24, 71, 140, 245]
Received packets: 78
Lost packets: 0
new_data
Receiving 10485760 Bytes...
10485760 Bytes received.
Length of data: 9
data[0:9]: [78, 231, 32, 233, 161, 24, 71, 140, 245]
Received packets: 79
Lost packets: 0
new_data
Receiving 10485760 Bytes...
10485760 Bytes received.
Length of data: 9
data[0:9]: [79, 231, 32, 233, 161, 24, 71, 140, 245]
Received packets: 80
Lost packets: 0
100 MiB received in : 11.113635540008545 seconds
new_data
Receiving 10485760 Bytes...
10485760 Bytes received.
Length of data: 9
data[0:9]: [80, 231, 32, 233, 161, 24, 71, 140, 245]
Received packets: 81
Lost packets: 0
new_data
Receiving 10485760 Bytes...
10485760 Bytes received.
Length of data: 9
data[0:9]: [81, 231, 32, 233, 161, 24, 71, 140, 245]
Received packets: 82
Lost packets: 0
```

Figure 14: Python TCP stream receiver.

Where we can see, that all TCP stream packets are received successfully. The same example can be used to communicate between any combination of Python ("tcpStreamTx.py" and "tcpStreamRx.py") and MATLAB applications.

## 5. Additional Information

- To compile the Cpp code, you need to set the path to your local boost library in ".\Examples\Cpp\CMakeList.txt"