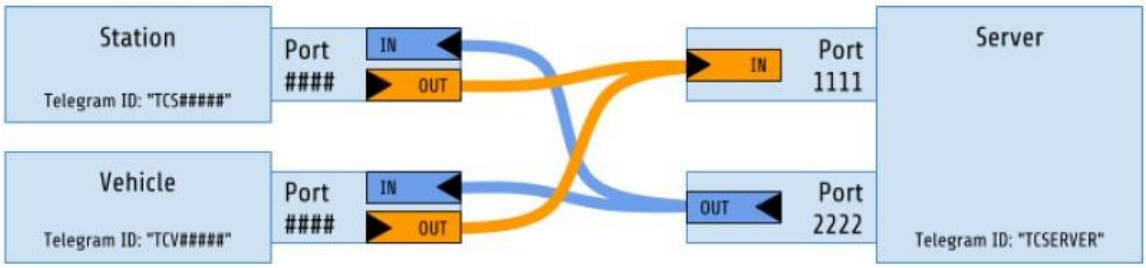
TransportControl Target API

Introduction

All network users connected to TransportControl via UDP are called targets. Targets can be vehicles or stationary devices (stations), e.g., controllers for traffic lights, fire doors, sensors, or buttons. TransportControl monitors targets continuously. Status and error messages can be logged and displayed in different ways. Unlike REST API clients, targets are meant to communicate cyclically with TransportControl - no matter if the telegrams contain new information or not.

Communication Structure

TransportControl's Target API is based on UDP. The network address of the host system of TransportControl is fixed and known to each target. There are no dedicated login telegrams or anything like that. As soon as the first telegram is received by TransportControl, the system will register a new target with the transmitted ID and starts to send back status telegrams immediately. TransportControl extracts and updates the target's network address from the UDP header after each incoming message and will cyclically send back telegrams to this address. Thus, it is possible that a target changes the IP or port number between two telegrams without failure. Targets must use the same port for incoming and outgoing telegrams, as indicated in the following chart:



TransportControl sends telegrams to targets in time with its core loop. Targets should adapt to this timing by synchronizing sending and receiving, e.g., by using a blocking receive method, see code example below for details. In any way, please avoid using an own timer for the target's communication with TransportControl!

Compensation of Packet Loss

Communication between targets and TransportControl is based on continuous monitoring. Telegrams are exchanged quite often but in return they are extremely compact. UDP with the lowest overhead is the most suitable protocol for this in our experience.

Due to the connectionless character of UDP telegrams might get lost. In such a case the next valid incoming telegram would again provide all up-to-date information, that means packet loss can be compensated through redundancy. In addition, time limits for offline periods are defined in the target's program as well as in the system settings of TransportControl. In case of exceeding these thresholds, appropriate actions will be initiated (speed reduction, emergency stop, etc.).

Structure of UDP Telegrams

index	bytes	type	name	value	description
▼	10 + n				standard packet from target to server
0	8	string	telegram ID	ASCII, 8 chars	TCV##### for vehicles TCS##### for stations
8	n	byte[]	payload	array	content and length must be defined via INPUT script
8 + n	2	short	checksum	CRC16-CCITT init: 0xFFFF poly: 0x1021	calculated for previous 8 + n bytes
▼	10 + n				standard packet from server to target
0	8	string	id	ASCII, 8 chars	TCSERVER
8	n	byte[]	payload	array	content and length must be defined via OUTPUT script
8 + n	2	short	checksum	CRC16-CCITT init: 0xFFFF poly: 0x1021	calculated for previous 8 + n bytes
▼	16				update packet from target to server
0	8	string	telegram ID	ASCII, 8 chars	TCV##### for vehicles TCS##### for stations
8	2	short	contentIndex	0 to numberOfParts minus 1	mirror received contentIndex from server
10	2	short	numberOfParts	1 to n	mirror received numberOfParts from server
12	2	short	contentLength	1 to n	number of content bytes from last received packet
14	2	short	checksum	CRC16-CCITT init: 0xFFFF poly: 0x1021	calculated for previous 14 bytes
▼	16 + n				update packet from server to target
0	8	string	telegram ID	ASCII, 8 chars	TCU + first 5 characters of filename
8	2	short	contentIndex	0 to numberOfParts minus 1	index of this part
10	2	short	numberOfParts	1 to n	total number of parts
12	2	short	contentLength	1 to n	number of content bytes in this packet
14	n	byte[]	content	array	file content as byte array
14 + n	2	short	checksum	CRC16-CCITT init: 0xFFFF poly: 0x1021	calculated for previous 14 + n bytes

ATTENTION: The UDP server of TransportControl expects the network byte order (big endian / high byte first). For example, the 2-byte value of the checksum 0x1234 must be transmitted in the telegram from target to server as follows:
index n: 0x12, index n+1: 0x34

Example: Simple Target Integration

Let us assume you want to integrate a simple device as a target for TransportControl. This device will exchange information in form of a single bit and a single byte. The following steps are needed to make things work:

1. WRITE YOUR TARGET'S CODE (HERE: JAVA 8)

Initialization

```
private DatagramSocket socket;
private Thread comThread = new ComThread();

private Target()
{
   try
   {
       // initialize socket and bind to any available port
       socket = new DatagramSocket();

       // stop trying to receive after 1 second
       socket.setSoTimeout(1000);
   }
   catch (Exception ex) {...}

   comThread.start();
}
```

Communication Thread

```
private class ComThread extends Thread
{
   public void run()
   {
       Thread thisThread = Thread.currentThread();
       while (comThread == thisThread)
       {
           sendTelegram();
           receiveTelegram(); // blocking to adept timing from server
       }
   }
}
```

Sending of Telegrams

```
private void sendTelegram()
{
   try
   {
       InetAddress ip = InetAddress.getByName("localhost"); // server IP
       int port = 1111; // server receive port

       byte[] data = new byte[12];
       ByteBuffer buffer = ByteBuffer.wrap(data);

       buffer.put("TCS00001".getBytes());
       buffer.put((byte) 1);
```

```
            buffer.put((byte) 123);

        // calculate and insert checksum
        buffer.putShort(getCRC(data, 0, 10));

        DatagramPacket packet = new DatagramPacket(data, data.length, ip, port);
        socket.send(packet);
    }
    catch (Exception ex) {...}
}
```

Receiving of Telegrams

```
private void receiveTelegram()
{
    try
    {
        byte[] data = new byte[12];
        DatagramPacket packet = new DatagramPacket(data, data.length);
        socket.receive(packet);

        String serverId = new String(data, 0, 8);
        if (serverId.equals("TCSERVER")) // looks like a valid telegram
        {
            ByteBuffer buffer = ByteBuffer.wrap(data);
            buffer.position(10); // index of checksum

            short receivedChecksum = buffer.getShort()
            if (receivedChecksum == getCRC(data, 0, 10))
            {
                // evaluate telegram
                buffer.position(8);
                System.out.println("payload byte 1 from TransportControl: " + (int) buffer.get());
                System.out.println("payload byte 2 from TransportControl: " + (int) buffer.get());

            }
        }
    }
    catch (Exception ex) {...}
}
```

Calculation of the Checksum

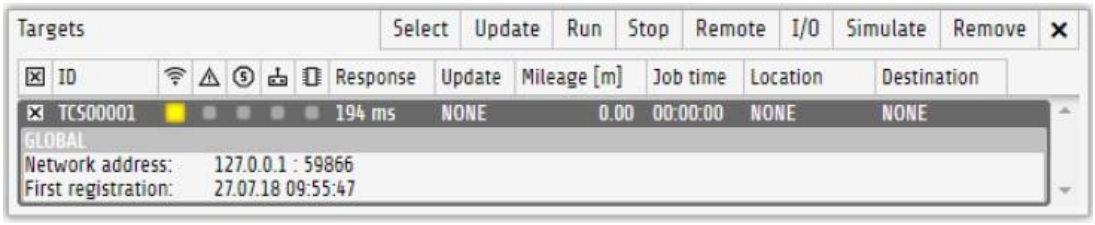
```
private static short getCRC(byte[] data, int start, int length)
{
    int crc = 0xFFFF; // initial value
    int polynomial = 0x1021;

    for (int i = 0; i < length; i++)
    {
        byte b = data[start + i];
        for (int k = 0; k < 8; k++)
        {
            boolean bit = ((b >> (7 - k) & 1) == 1);
            boolean c15 = ((crc >> 15 & 1) == 1);
            crc <<= 1;
```

```
            if (c15 ^ bit) crc ^= polynomial;
        }
    }
    return (short) crc;
}
```

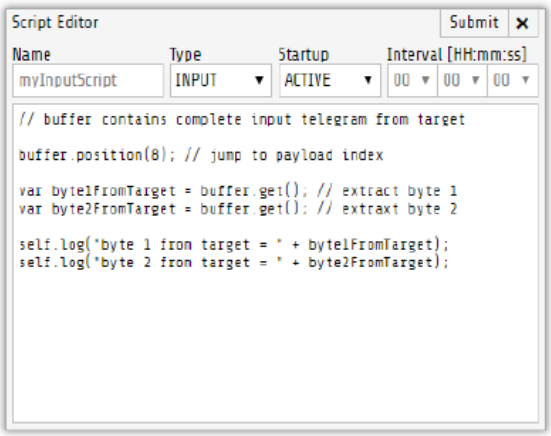
2. CHECK FOR CONNECTION

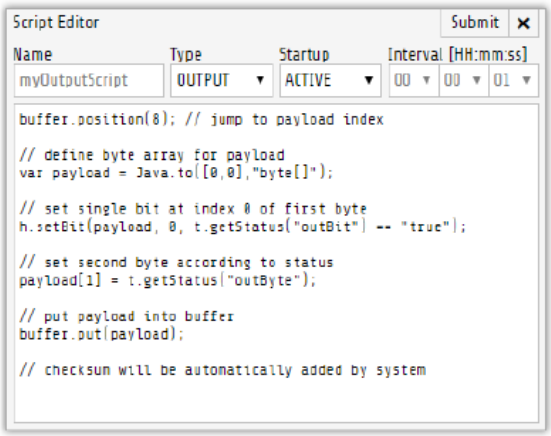
Run your newly created program, then open TransportControl Main UI and click on [Targets]. Your target will show up in the target table if its telegrams are not blocked by a firewall and reach TransportControl. For now, the telegrams do not need to be correct regarding length and checksum, also the return telegrams from TransportControl don't need to be received by your target.



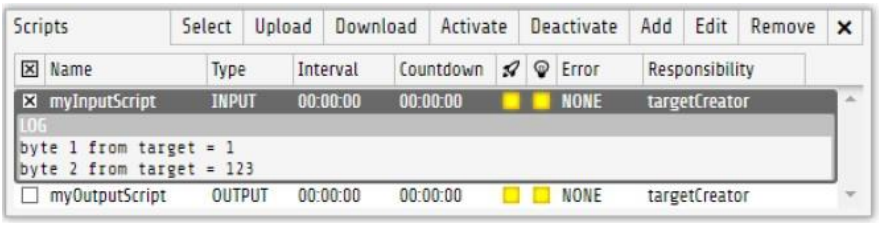
3. CREATE INPUT AND OUTPUT SCRIPTS

In this step you tell TransportControl about the structure of your telegrams. You do this by creating two special scripts, one for incoming and one for outgoing telegrams. With the help of JavaScript as the scripting language of TransportControl, you have the possibility to tune your target's I/Os on the fly in a running system. To keep it simple in the beginning create the scripts like this (INPUT and OUTPUT are named from the perspective of TransportControl):



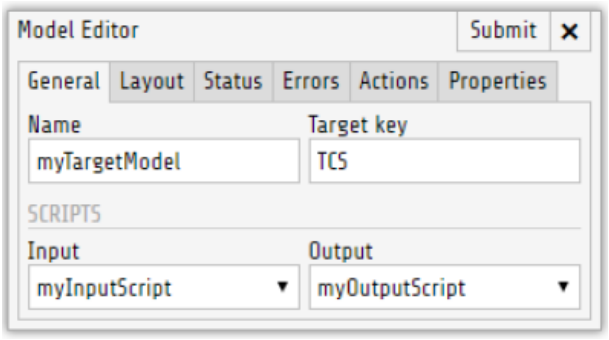


Select both scripts and press [Activate]. The LEDs in the column with the light bulb icon should be light up like in the screenshot below. If you select only the input script, you can already see the logs of the raw input data:



4. CREATE A MODEL

Now that the scripts are ready, TransportControl needs to know to which type of target they belong. Target types are called models. Targets will be linked to a specific model based on their ID. Create a new model by selecting [Models] > [Add]. Set the inputs as in the screenshot below. All targets whose ID starts with "TCS" (yours including) will be linked to this new model.



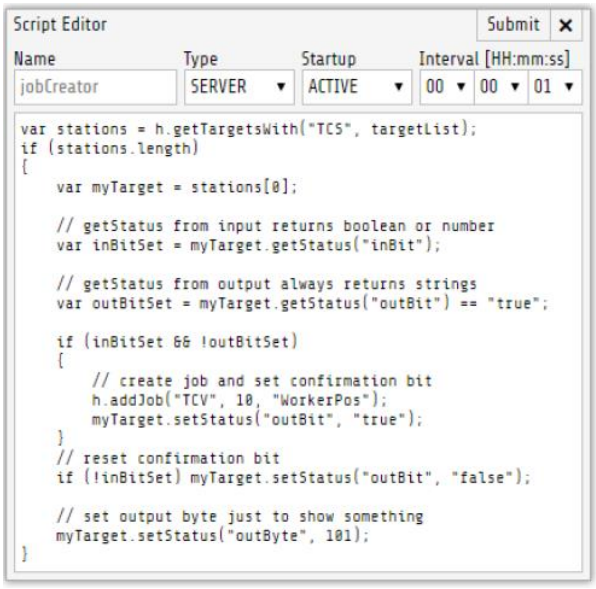
Switch to the [Status] tab. Here you can define what information is hidden inside the payload of the telegrams. The NAME will appear inside the target table and the logs, also it is the key to access the information from within scripts. INDEX means bit index. Each telegram starts with the 8 bytes long ID, so the first bit index of the telegram payload would be 8 * 8 = 64. INDEX -1 means that this status entry cannot be found in the input telegram, instead it is used by TransportControl's output to the target. LENGTH is the bit length of the status entry. The maximum bit length of a status entry is 64. ALERT defines how this status entry contributes to the target's alert state. Targets with an alert state will indicate this in the target table, in addition vehicles will blink red on the map. COLOR refers to the LED color of the status entry inside the target table, it does not affect the alert color which is always red. You can find further details regarding ALERT and COLOR listed below. Fill the status table as in the screenshot:

ALERT entry	Matching LENGTH	Description	COLOR entry	description
-1	n	status value will not affect target alert	green yellow red blue	status LED will light up in the specified color if status value is greater than 0
0	1	target is alerting if status value is 0		
1	1	target is alerting if status value is 1		
n	8	target is alerting if status value is below n	auto	status LED color depends on status value

Why are script and model creation separated? Because multiple models could share the same I/O scripts. For example, if an automation company integrates a forklift and a truck that both use the same telegram structure, they would need two models but only one pair of scripts.

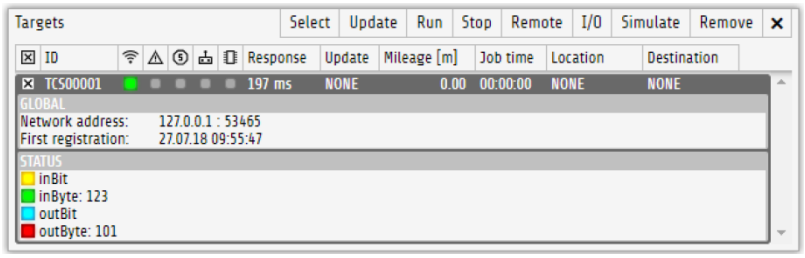
5. USE YOUR TARGET'S INFORMATION

Next, we want to work with the information of the target. Maybe the single bit inside the input telegram reflects a push button that is pressed by a worker to call an AGV. If the call was successful (a job was created), TransportControl should confirm by returning a bit to your target, which then can clear its bit. This should serve as a simple handshake pattern. You can realize all this by writing a script of type SERVER as follows:

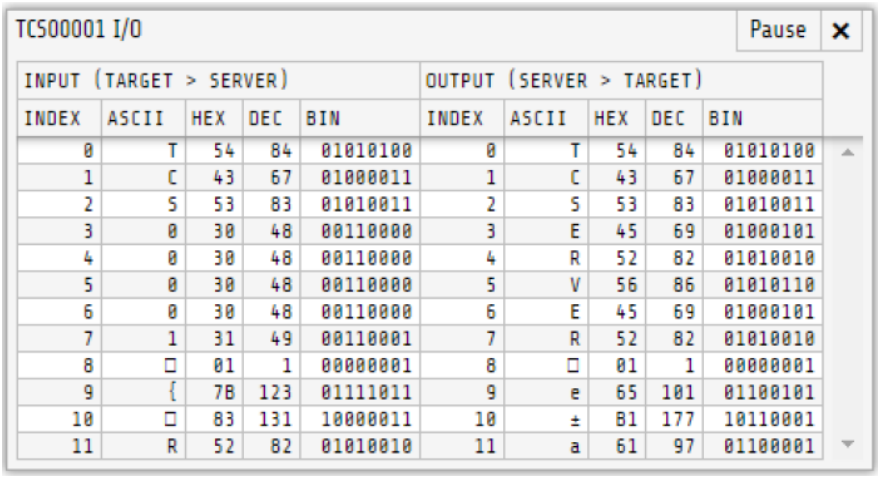


6. VALIDATION AND TROUBLESHOOTING

Let us switch back to the target table. If everything works as expected, you can now see the status (both input and output) of the selected target correctly. Please note the green LED in the connection status column (WiFi symbol). Before it was yellow to indicate that the input telegram could not be validated. But after the script and model creation TransportControl knows how long the target's telegram will be and therefore also at which position the checksum is transmitted?



If something goes wrong, the I/O box is worth a look. Click on the [I/O] button at the top of the target table while your target is selected. Now you can see the raw data of both telegrams. You will recognize the three parts of each telegram: ID (8 bytes), payload (2 bytes) and checksum (2 bytes). Check if everything is listed as intended and adjust the target's code if necessary.



INPUT (TARGET > SERVER)					OUTPUT (SERVER > TARGET)				
INDEX	ASCII	HEX	DEC	BIN	INDEX	ASCII	HEX	DEC	BIN
0	T	54	84	01010100	0	T	54	84	01010100
1	C	43	67	01000011	1	C	43	67	01000011
2	S	53	83	01010011	2	S	53	83	01010011
3	0	30	48	00110000	3	E	45	69	01000101
4	0	30	48	00110000	4	R	52	82	01010010
5	0	30	48	00110000	5	V	56	86	01010110
6	0	30	48	00110000	6	E	45	69	01000101
7	1	31	49	00110001	7	R	52	82	01010010
8	□	01	1	00000001	8	□	01	1	00000001
9	{	7B	123	01111011	9	e	65	101	01100101
10	□	83	131	10000011	10	±	B1	177	10110001
11	R	52	82	01010010	11	a	61	97	01100001