TEXAS INSTRUMENTS

# Client-Server Model in Asterix(Binary format):

## Objective:

Send commands to Asterix/Latte from external active process and receive the output data from it.

## Introduction:

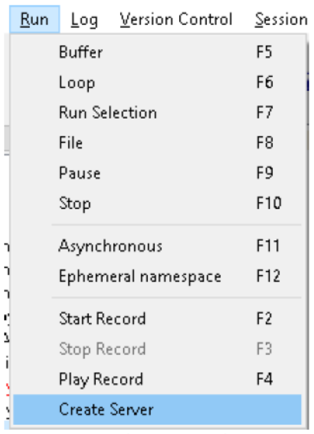
Sockets allow communication between two different processes on the same or different machines. This is used for end-to-end communication. One of the two processes, the *client*, connects to the other process, the *server*, typically to make a request for information. Notice that the client needs to know of the existence and the address of the server, but the server does not need to know the address of (or even the existence of) the client prior to the connection being established.

## Protocol:

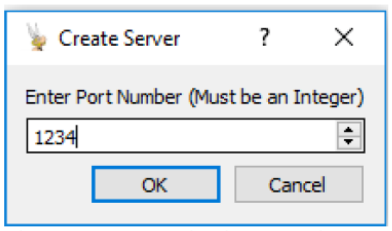
1. Asterix/Latte acts as server with Port number (Assigned by Asterix\Latte user)
2. The external process should be client
3. Asterix/Latte can accept only one client request which comes first and will ignore the others
4. The server needs to release current client to accept other client request (The client can send command **release** to disconnect from server)
5. After receiving and verifying the message, the receiver needs to send **acknowledgment** signal to transmitter. If verification failed then receiver sends **fail** signal. After getting fail signal transmitter needs to resend the message.
6. The server after sending the required data, should send **done** signal to client which indicates that server is ready to accept new command and executed previous command successfully.
7. The server will look for input for duration **1800** secs and disconnect if duration exceeds.
8. If client wants to close the server then it needs to send **exit** command.

## Steps to create server:

1. Select Run  Create Server



2. Give a valid port number (1 to 65535)



3. After successfully creating the server, the below message will be visible in log window

```
Socket created
Socket bind complete
Socket now listening. Port number: 1234
```

## Architecture:

The client needs to initiate the communication after connected to server. The client needs to send commands in below format.

| Header | Data Size | Payload | CheckSum |
|---|---|---|---|

**Header:** It indicates starting of message and it is predefined as **HEADER.**

**Data Size:** It indicates number of bytes the payload contains. It is of size **5 bytes**. It contains the value in binary format and every byte's MSB should be 1. That makes effective available 35 bits. Example: size is 100. The data in binary 10000000_10000000_10000000_10000000_11100100 and ASCII format is '\x80_\x80_\x80_\x80_\xe4'.

**Payload:** This is actual information the client wants to send. It should be in **ASCII** format.

**CheckSum:** It is sum of all the bytes in payload. It should be binary format (same as Data Size). Size of checksum should be **8 bytes**. It has effectively 56 bits for use.

The server can send data in two formats (integer and string). The string should be sent in ASCII format but integer should be in binary for better utilization of space. So, the server needs to send data in following format.

| Header | Data Size | Data Type | Payload | CheckSum |
|---|---|---|---|---|

**Header:** It is same as client header (**HEADER**).

**Data Size:** This is same as client Data Size.

**Data Type:** This contains extra information about the type of data. It is of size **1 byte** and can have value from 1 to 255.

**Payload:** Depending upon the type of data, it is encoded. If string needs to send then ASCII value otherwise integer within given range **(-32768 to 32767)** can be sent in binary format with all bytes' MSB is 1.

**CheckSum:** This is same as client CheckSum.

## Handshaking messages:

**PASS:** This is an acknowledgement message sent by server to client when it received and verified the command successfully.

**FAIL:** This is the message sent by server to client when the received command failed verification. The server will prompt for new command and won't process previous one.

**DONE:** This is the message sent by server to client when execution finished successfully and ready to take next command.

**RELEASE:** This command sent from client to server. After getting this command the server will disconnect from client and look another connection request. After identifying the command, the server will send EXIT message to client and after receiving this message client need to close itself.

**EXIT:** This command sent from client to server. After identifying the command, the server will send EXIT message to client and after receiving this message client need to close itself. And also the server closes itself as well.