**Using NI-XNET with gRPC-device**

**Server Configuration**

**Installation**

1. Install NI-XNET on the server
2. Download and extract the latest release for [NI gRPC Device Server](https://github.com/ni/grpc-device/releases) for you Server’s OS
3. Unzip the tar or zip file on a known location on your server machine

**Running the Server**

1. Run ni\_grpc\_device\_server, the server will print out its listening port when it’s running successfully

Text

Description automatically generated

**Client Configuration**

Original instructions here: <https://github.com/ni/grpc-device/wiki/Creating-a-gRPC-Client>

**Client Installation**

**Install grpcio-tools >pip install grpcio-tools**

**Downloading Client files**

1. Download [ni-grpc-device-client.tar.gz](https://github.com/ni/grpc-device/releases/download/v1.5.1/ni-grpc-device-client.tar.gz) from the grpc-device releases and extract its contents
2. The release folder contains nixnet.proto. This Protobuff file contains all of the function prototypes and data types that will be used to communicate through CAN.

Graphical user interface, application, table

Description automatically generated

**Generate support Python Files**

1. Navigate to ni-grpc-device-client/examples/nixnet to view the shipping nixnet examples
2. The first time running these examples, we need to generate support python files. Generate the files by navigating to the examples folder on the terminal and running:

> python -m grpc\_tools.protoc -I="..\..\proto" --python\_out=. --grpc\_python\_out=. session.proto nidevice.proto nixnet.proto

1. The resulting folder should look like this:

Graphical user interface, text, application

Description automatically generated

**Run Client Examples**

The example clients are documented at the top of the file:

To run this example, install "NI-XNET Driver" on the server machine:

  https://www.ni.com/en-in/support/downloads/drivers/download.ni-xnet.html

For instructions on how to use protoc to generate gRPC client interfaces, see our "Creating a gRPC

Client" wiki page:

  https://github.com/ni/grpc-device/wiki/Creating-a-gRPC-Client

Refer to the NI XNET gRPC Wiki for the latest C Function Reference:

  https://github.com/ni/grpc-device/wiki/NI-XNET-C-Function-Reference

Running from command line:

Server machine's IP address, port number, and interface name can be passed as separate command line

arguments.

  > python can-signal-single-point-output.py <server\_address> <port\_number> <interface\_name>

If they are not passed in as command line arguments, then by default the server address will be

"localhost:31763"

Running the CAN Signal Single Point Output example with parameters, seeing a successful response:

Text

Description automatically generated

The XNET Raw example sends a Raw CAN FD Frame example shows how to send Raw CAN Frames instead of signals.

A Frame Request message is used to configure CAN, CANFD and Automotive Ethernet frames. These message formats are defined on the Proto file:

Text, letter

Description automatically generated

Text

Description automatically generated

These message types are used when sending or receiving frames:

payload\_list = [1] \*8

canfdframe=nixnet\_types.FrameRequest(identifier=0x201, type = nixnet\_types.FRAME\_TYPE\_CAN\_DATA, payload= bytes(payload\_list))

canfdframes=[nixnet\_types.FrameBufferRequest(can= canfdframe)]

Finally, this frame type is used on the WriteRequest function to write to the bus:

  write\_frame\_response = client.WriteFrame(

            nixnet\_types.WriteFrameRequest(session=session, buffer=canfdframes, timeout\_raw= 10.0)

        )

**Additional XNET Function Guide**

To configure other properties, the grpc [wiki](https://github.com/ni/grpc-device/wiki/NI-XNET-General-Functions#nxWriteFrame) has documentation on C and gRPC generated functions. This wiki [article](https://github.com/ni/grpc-device/wiki/gRPC-API-differences-from-C-APIs) also mentions how to build Frames and how to set and get properties.