Using LXI Devices in CANoe

Version 1.2
2018-02-20
Application Note AN-IND-1-016

Author Vector Informatik GmbH

Restrictions Public Document

Abstract Introduction on how to create a module-to-module LXI connection with CAPL.

Table of Contents

1.0 Overview

CANoe provides the possibility to control external devices with an LXI interface. The LXI commands can be sent as data packets in TCP messages using CAPL. This application note describes how to communicate with LXI devices in CANoe.

1.1 Use Case / Motivation

Power supplies and measurement devices often have an LXI interface for remote control. To use those devices in an automated test setup, it is necessary to have access to the LXI interface in CANoe. The communication with CANoe and an LXI device can be implemented using manually written CAPL code. The extent and complexity of the code depends on the use case. Typically a few lines of code are sufficient as described in this application note.

2.0 Module-to-module communication with LXI

This application note explains how to establish a module-to-module connection between a PC and an LXI power supply. A kind of a “device driver” is defined in CAPL. This CAPL node may be added to the test setup and provides an interface between the test application in CANoe and the LXI device. System variables are typically used as the interface mechanism in CANoe.

To implement an LXI communication in the “driver” node, a TCP socket must be created. This socket is used to establish a connection between CANoe and the device and for handling the data protocols. The CAPL code below shows a simple connection between CANoe and a LXI power supply.

The output voltage of the power supply can be set to a specific value with the system variable ‘var_Voltage’. The output itself can be activated or deactivated by pushing the key ‘o’. In addition the identification of the power supply can be requested by changing the system variable ‘var_ID’ to the value 1.

Important Note
For this example the power supply CPX400SP from TTi was used. Some of the commands are device-dependent and can be different for other power supplies. Please read the appropriate user manual of the device for the correct commands.

```
variables
{
  dword address_canoe;
  dword address_power_supply;

  dword port = 9221; // used port
  char buffer_id[16] = "*idn?"; // Identification command
  char buffer_send_volt[128];
  char buffer_receive[128];

  int output = 0; // output activate/deactivate

  dword LXI_socket;
}

on preStart
{
  address_canoe = IpGetAddressAsNumber("192.168.100.1"); // IP address PC
  address_power_supply = IpGetAddressAsNumber("169.254.4.226"); // IP
  address power supply
}

on start
{
  LXI_socket = TcpOpen(address_canoe, port); // creation of an IP socket
  TcpConnect(LXI_socket, address_power_supply, port); // connecting the IP
  socket
}

on stopMeasurement
{
  TcpClose(LXI_socket);
}

on sysvar sysvar::LXI_Variables::var_ID // identification request
{
  if(1 == @this)
  {
    TcpSend(LXI_socket, buffer_id, strlen(buffer_id));
    TcpReceive(LXI_socket, buffer_receive, strlen(buffer_receive));
    //  Function to receive data packets
  }
}

on sysvar sysvar::LXI_Variables::var_Voltage // set voltage
{
  snprintf(buffer_send_volt, elcount(buffer_send_volt), "V1 %f", @this);
  // creation of the command string
  TcpSend(LXI_socket, buffer_send_volt, strlen(buffer_send_volt));
  TcpReceive(LXI_socket, buffer_receive, 128);
}

on key 'v' // measured output voltage
{
  TcpSend(LXI_socket, "V1 O?", strlen("V1 O?"));
  TcpReceive(LXI_socket, buffer_receive, 128);
}

on key 'o' // activate/deactivate output
{
  if(0 == output)
  {
    TcpSend(LXI_socket,"OP1 1", strlen("OP1 1"));
    output = 1;
  }

  else
  {
    TcpSend(LXI_socket,"OP1 0", strlen("OP1 0"));
    output = 0;
  }
}
```

```
void OnTcpReceive( dword socket, long result, dword address, dword port,
char buffer[], dword size) // receive function
{
  write("Received: %s", buffer); // output of the received string
  TcpReceive(LXI_socket, buffer_receive, 128);
}
```

2.1 Creating a network node

To create an interface a network node must be inserted into the test setup (“Test” → “Test Setup” → right-click “New Test Environment” → ”Insert Network Node”). The specification (.can file) for this node is the CAPL file describing the interface.

2.2 Including the IP addresses

To initialize the necessary socket for the communication the IP address of the PC and the power supply has to be converted from a string (e.g. ‘192.168.100.1’) into a numerical value that can be used by the CAPL program. For this the CAPL function `IpGetAddressAsNumber` in the `on preStart` event handler is used.

```
on preStart
{
  address_canoe = IpGetAddressAsNumber("192.168.100.1"); // IP address PC
  address_power_supply = IpGetAddressAsNumber("169.254.4.226"); // IP
  address power supply
}
```

2.3 Initiliazing a socket

After the IP addresses are converted the socket must be initiated. This can be done with the CAPL function `TcpOpen`. This function needs the IP address of the PC and a port for the communication. Please remember that it has to be a port that is not used by another program, in this case port 9221. After the socket is created it must be connected to the used device with the function `TcpConnect`. For this the device’s IP address is needed and a port must be selected. Both ports can be the same. The necessary functions are part of the `on start` event handler.

```
on start
{
  LXI_socket = TcpOpen(address_canoe, port); // creation of an IP socket
  TcpConnect(LXI_socket, address_power_supply, port); // connecting the IP
  socket
}
```

2.4 Closing a socket

When the measurement is stopped the socket has to be closed with the function `TcpClose`. This function is part of the `on stopMeasurement` event handler.

```
on stopMeasurement
{
  TcpClose(LXI_socket);
}
```

2.5 Receiving data messages

The function `TcpReceive` is used to receive data into a specified buffer. The function has to be called before the first packet is received, for example when the connection is established. Usually an answer can't be immediately sent by the LXI device. The function `OnTcpReceive` must be used to capture an asynchronous packet and write it into the buffer. Receiving a packet completes the data transfer and a new packet can't be received anymore unless the function `TcpReceive` is called again. The best solution would be to call the receive function again in the function `OnTcpReceive`.

```
void OnTcpReceive( dword socket, long result, dword address, dword port,
char buffer[], dword size) // receive function
{
  write("Received: %s", buffer); // output of the received string
  TcpReceive(LXI_socket, buffer_receive, 128);
}
```

2.6 Sending data messages

The function `TcpSend` is used to send a command to the device. The command is send as data of the TCP package and is written in ASCII (e.g. '`*idn?`'). The function `TcpSend` adds the necessary header to the packet so that the command is only the necessary ASCII code. The available commands for a device are specified in the appropriate manual. They are normally divided into device-dependent and -independent commands. The two event handlers `on sysvar` are used to get the identification of the power supply and to set the output voltage. The two event handlers `on key` are used to request the measured output voltage of the power supply and to switch the output itself on and off.

```
on sysvar sysvar::LXI_Variables::var_ID // identification request
{
  if(1 == @this)
  {
    TcpSend(LXI_socket, buffer_id, strlen(buffer_id));
    TcpReceive(LXI_socket, buffer_receive, strlen(buffer_receive));
    // Function to receive data packets
  }
}

on sysvar sysvar::LXI_Variables::var_Voltage // set voltage
{
  snprintf(buffer_send_volt, elcount(buffer_send_volt), "V1 %f", @this);
  // creation of the command string
  TcpSend(LXI_socket, buffer_send_volt, strlen(buffer_send_volt));
  TcpReceive(LXI_socket, buffer_receive, 128);
}

on key 'v' // measured output voltage
{
  TcpSend(LXI_socket, "V1 O?", strlen("V1 O?"));
  TcpReceive(LXI_socket, buffer_receive, 128);
}

on key 'o' // activate/deactivate output
{
  if(0 == output)
  {
    TcpSend(LXI_socket,"OP1 1", strlen("OP1 1"));
    output = 1;
  }

  else
  {
    TcpSend(LXI_socket,"OP1 0", strlen("OP1 0"));
    output = 0;
  }
}
```

3.0 Contacts

For a full list with all Vector locations and addresses worldwide, please visit http://vector.com/contact/.