Aim TTi
Measurably better value

TELONIC
TEST INSTRUMENTS & POWER SUPPLIES
ISO 9001 QUALITY MANAGEMENT
UKAS MANAGEMENT SYSTEMS 0015
ESTABLISHED OVER 50 YEARS SINCE 1966
01189 786911 TELONIC.CO.UK

Sending Lists and Sequences via LAN

Application Note

Products:

SMU4000 Series:
SMU4001
SMU4201

This application note describes how to transfer a list or sequence to an SMU4000 series instrument via a LAN connection.

Introduction

When communicating with the SMU4000 series SMU via LAN the standard method of transferring data in the form of lists and/or sequences is not available, the following method is required to transfer data.

Equipment Required

- One SMU4000 series SMU connected to a PC via LAN

Method (Sequence File)

Transferring a sequence file to the SMU via LAN is done in three stages:

1. Send a start command to the SMU
   MEMory:DATA:STARt <File Number>,<File Type>,<Length>\n
   The three arguments are defined as:
   <File Number> = 0 (not used for sending sequences)
   <File Type> = 0 (0= sequence, 1 = list)
   <Length> = total number of bytes in the sequence file.
2. Transfer the file data in chunks of 1200 bytes. Send the following command multiple times until all the data has been transferred:
   MEMory:DATA:TRANsfer <Data Start Index>,<Number Of Bytes>,<Data Block>
   The three arguments are defined as:
   <Data Start Index> = Zero based position of the first byte of data being transferred to the SMU in this command.
   <Number Of Bytes> = Number of bytes of data being transferred to the SMU in this command.
   <Data Block> = Binary data being transferred.
   So as not to overload the SMU with too much data too quickly send a *OPC?\n command after each transfer command so you know that the SMU is ready for the next block of data.
   Note: Don't send \n at the end of the command.
3. Send a complete command telling the SMU to load the sequence file into memory.
   Send:
   MEMory:DATA:COMPlete\n
   The SMU will check that the file you have sent is valid and then load it.

Method (List File)

Transferring a list to the SMU via LAN is done as follows:

1. Build a binary array of data 4bytes per list point. Encode the float data as per IEEE 745 and store it in an array.
2. Send a start command to the SMU
   MEMory:DATA:STARt <File Number>,<File Type>,<Length>\n
   The three arguments are defined as:
   <File Number> = list number to be stored in the SMU The filename will be formatted as LIST<File Number>.CSV. It must be between 0 and 99.
   <File Type> = 1 (0= sequence, 1 = list)
   <Length> = total number of bytes in the list binary array.
3. Transfer the file data in chunks of 1200 bytes. Send the following command multiple times until all the data has been transferred:
   MEMory:DATA:TRANsfer <Data Start Index>,<Number Of Bytes>,<Data Block>
   The three arguments are defined as:
   <Data Start Index> = Zero based position of the first byte of data being transferred to the SMU in this command.
   <Number Of Bytes> = Number of bytes of data being transferred to the SMU in this command.
   <Data Block> = Binary data being transferred.
   So as not to overload the SMU with too much data too quickly send a *OPC?\n command after each transfer command so you know that the SMU is ready for the next block of data.
   Note: Don't send \n at the end of the command.
4. Send a complete command telling the SMU to save the list file. Send:
   MEMory:DATA:COMPlete\n
   The SMU will check that the file you have sent is valid and then save it.

C# Example

This is a C# method that can be used to transfer sequences and lists to and SMU

```
using System;
using System.Net.Sockets;
using System.Text;
using System.Threading;

namespace SMU.Model
{
  internal class ApplicationNoteExamples
  {
    /// <summary>
    /// Send binary list or sequence. For use with LAN, it will ensure that all data transfers are less than one packet long.
    /// </summary>
    /// <param name="socket">Ethernet TCP socket</param>
    /// <param name="data">Binary list or sequence data</param>
    /// <param name="list">True = list, False = sequence</param>
    /// <param name="fileNumber">List number, will create a list called LIST<fileNumber>.CSV. Not used for sequences</param>
```

```
        /// <param name="overwrite">Not used</param>
        /// <param name="progress">ProgressReporter object to allow the GUI progress to be updated during the transfer. null is allowed</param>
        /// <param name="length">Optional length of data to be sent. Used to test that the SMU will raise an execution error on complete if it didn't receive all the data.</param>
        /// <returns>True = Succeeded in transferring the data to the SMU. False = Failed to send the data to the SMU.</returns>
        private bool ChunkedBinaryTransfer(Socket socket, byte[] data, bool list, int fileNumber)
        {
            int maxElementsCommand = 1200;
            int listInt = 0;
            if (list == true)
            {
                listInt = 1;
            }
            //Build Start Command
            //fileNumber = list file number and will create a list called LIST<fileNumber>.CSV
            //list = 1 if a list, 0 = sequence
            //length = total number of bytes being sent
            string cmd = $"MEMory:DATA:STARt {fileNumber},{listInt},{data.Length}\n";

            //Send the start command
            socket.Send(Encoding.UTF8.GetBytes(cmd));
            socket.NoDelay = true;

            //Send the chunks of data
            int sentElementsIndex = 0;
            while (sentElementsIndex < data.Length)
            {
                int sendNoElements = maxElementsCommand;
                if ((data.Length - sentElementsIndex) < maxElementsCommand)
                {
                    sendNoElements = data.Length - sentElementsIndex;
                }

                //Build the tranfer command header
                cmd = $"MEMory:DATA:TRANsfer {sentElementsIndex},{sendNoElements},";

                //send the command header
                socket.Send(Encoding.UTF8.GetBytes(cmd));

                //send the command data
                socket.Send(data, sentElementsIndex, sendNoElements, SocketFlags.None);

                //Update the sentElementsIndex
                sentElementsIndex += sendNoElements;

                //Wait for OPC command to return
                socket.Send(Encoding.UTF8.GetBytes("*OPC?\n"));
                Thread.Sleep(5);
                var receivedBytesFull = new byte[1000];
                var receivedLength = socket.Receive(receivedBytesFull);
                var receivedBytes = new byte[receivedLength];
                Array.Copy(receivedBytesFull, receivedBytes, receivedLength);
                string result = Encoding.UTF8.GetString(receivedBytes);
                if(result.Trim() != "1")
                {
                    return false;
```

```
            }
        }
        socket.NoDelay = false;
        //Build the complete command
        cmd = "MEMory:DATA:COMPlete\n";

        //Send the complete command, this will cause the list file to be written or the sequence to be
loaded into memory.
        socket.Send(Encoding.UTF8.GetBytes(cmd));

        return true;
    }
  }
}
```