What is it?

SCPI stands for **S**tandard-**C**ommands-for-**P**rogrammable-**I**nstruments. SCPI allows the same commands to control equipment from different manufacturers. SCPI commands use the hierarchical command structure that is popular among instrument vendors. Once you start using SCPI commands to program an Oscilloscope you find it easier to start using SCPI commands to program a Digital Multi-Meter. If you are writing software to program and control various instruments you have fewer interfaces that you have to learn.

While SCPI is easy for the user to use, it can be difficult to program. This parser allows you to define a set of descriptive command templates so that the parser can recognize the command and call the appropriate function. This way, the programmer can concentrate on building the instrument.

SCPI.c and *SCPI.h* provide a simple parsing method for SCPI commands. Functions have been provided for the 488.2 mandatory commands and the user can add templates for all of their custom commands. Default functions for commands and queries are provided so that you can define and test the command set prior to implementing the instrument functionality. Example software is provided that runs on both a PC and an embedded NetBurner MOD54415 controller. The example software recognizes the command set for an Agilent U2741A USB Modular 5.5 Digits Digital Multimeter (with permission from Agilent) and the full set of command templates were defined in a single evening. By the end of the evening the server was able to recognize all of the commands for the U2741A.

The example code uses AES encrypted UDP to communicate between the user (the client) and the instrument (the server). In the PC example, *SCPIServer.exe* emulates an Agilent U2741A. *SCPIClient.exe* is a console based client that you talk to it with.

Commands are processed in *SCPIServer.c* by calling "*size_t ProcessCommand (char * szRequest, char * szResponse, size_t cbResponse);*". You pass the command string and it processes the command and the command returns the response (a NULL string is a valid response). The command string may have been acquired using GPIB, RS-232, USB, TCP, UDP or any other source that can read and write ASCII strings.

Command Templates

The commands recognized are defined by templates. For Example the following Agilent commands:

```
CALibration:ADC?
CALibration[:ALL]?
CALibration:COUNt?
CALibration:SECure:CODE <new_code>
CALibration:SECure:STATe {OFF|0|ON|1|RESET},<code>
CALibration:STORe
CALibration:STRing <string>
CALibration:VALue <value>
```

are defined by the following templates:

```
{ ":CALibration:ADC?",                                       DefQry },
{ ":CALibration:ALL?",                                   DefQry },
{ ":CALibration:COUNt?",                                 DefQry },
{ ":CALibration:SECure:CODE <new_code>",                 DefCmd },
{ ":CALibration:SECure:STATe {OFF|0|ON|1|RESET},<code>", DefCmd },
{ ":CALibration:SECure:STATe?",                          DefQry },
{ ":CALibration:STORe",                                  DefCmd },
{ ":CALibration:STRing <string>",                        DefCmd },
{ ":CALibration:STRing?",                                DefQry },
{ ":CALibration:VALue <value>",                          DefCmd },
{ ":CALibration:VALue?",                                 DefQry },
{ ":CALibration?",                                       DefQry },
```

Notice that “CALibration[:ALL]?” requires two templates:

```
“{ ":CALibration:ALL?", DefQry },” and
“{ ":CALibration?",     DefQry },”.
```

You must specify a separate template for each optional format. The last parameter on the template is the function to be invoked in the format. Default functions are provided:

*size_t DefCmd (char * szCommandLine, char * szCommandSyntax, char * szResponse, size_t cbResponse);*
*size_t DefQry (char * szCommandLine, char * szCommandSyntax, char * szResponse, size_t cbResponse);*

DefCmd is the default command function and *DefQry* is the default query, and both are provided for your convenience. After you get the commands working, you can add your own functions to process each command, using the same calling sequence.

The command templates must be in ASCII sorted order. The *VerifyScpiTableSortOrder()* function has been provided to assure that it is.

IEEE488.2 Common Commands

The SCPI Standard states that “If you wish to implement full SCPI compliancy, you will need to support these and react to the commands in the ways required by IEEE488.2.” Therefore, the following mandatory IEEE488.2 Common Commands are recognized and are partially implemented:

Command	**Description**
*CLS (ESR)	Clears the Standard Event Status Register and Clears the Events Queue
*ESE <NR1>	(ESE) Sets the Standard Event Status Enable Register
*ESE?	(ESE) Queries the Standard Event Status Enable Register
*ESR?	(ESR) Queries and Clears the Standard Event Status Register
*IDN?	Return the Device ID as <Manufacturer>, <ModelNumber>, <SerialNumber>, <FirmwareRev>
*OPC	Start Operation Complete Active Mode
*OPC?	Has Operation completed? <0/1>
*RST	Reset the Device
*SRE <NR1>	(SRER) Sets the Service Request Enable Register
*SRE?	(SRER) Queries the Service Request Enable Register
*STB?	(SBR) Queries the Status Byte Register. Bit 6 is reset to 0, the other bits kept unchanged.
*TST?	Test the Device and Return the Result <0/1>
*WAI	Wait Until the Last Operation is Completed

The library maintains global variables that implement the various status registers in support of the IEEE488.2 Common Commands. You, the programmer, can access these registers as part of your program's behavior. There are also a series of "bit" defines in the SCPI.h including the bits in the "Event Status Register" (ESR), the "Status Byte Register" (STB), the "SCPI OPERation Status Register" and the "The SCPI QUESTionable Status Register". Not all bits have already been assigned a function yet. An example is the bits in the "SCPI OPERation Status Register" and the "SCPI OPERation Status Enable Register".

Variable	**Description**
ucESR	The Standard Event Status Register (ESR) - Read with "*ESR?", Cleared with "*CLS"
ucESE	The Event Status Enable Register (ESE) for the ESR - Set with "*ESE <NRf>" and Read with "*ESE?"
ucSTB	The Status Byte Register (STB) - Read with "*STB?"
ucSRE	The Service Request Enable Register (SRE) for the STB - Set with "*SRE <NR1>" and Read with "*SRE?"
usOPER	The SCPI OPERation Status Register
usQUEST	The SCPI QUESTionable Status Register
usQUESTE	The SCPI QUESTionable Status Enable Register
usOPER	The SCPI OPERation Status Register
usOPERE	The SCPI OPERation Status Enable Register

The Example Code

The example code includes a PC Client, a PC Server and a NetBurner Server. They use AES encrypted UDP with IEEE 1174 "Serial Interface for Programmable Instrumentation" emulation to communicate between the client (the user) and the server (the instrument). This allows the user to command an instrument that can be located anywhere that has an internet connection available. Most SCPI instruments use GPIB which is bulky and expensive (just price out a fiber optic GPIB extender to see what I mean[1]). Using the internet frees you from distance limitations and lets you use thinner, more flexible cabling. Encrypting the communications keeps it private and because the communication is encrypted, it is self validating. If the user was able to encrypt the command then he or she is authorized to issue the command.

The PC example code can be compiled using the free *Microsoft Visual Studio Express 2013 for Windows Desktop*[2] compiler. The NetBurner example code can be compiled using the NetBurner NNDK 2.6.3 (the NetBurner code was compiled and verified on both a MOD5282 and a MOD54415). The PC code builds *SCPIServer.exe* and *SCPIClient.exe*. The NetBurner example code builds *SCPI_APP.s19*. The file in the ZIP file was build for a MOD54415 but the project can be recompiled to run on a MOD5282. The PC version implements stacked commands but the NetBurner does not. The code to support stacked commands works in the PC version but has not been moved into *SCPI.c*. At some point, it will be moved over. I decided to release this version so that people could start using it and commenting on it. You have to get into the pool sometime. The same goes for this manual. Expect more and better later. In the meantime, have some fun with what's here. Good Luck!

[1] http://sine.ni.com/nips/cds/view/p/lang/en/nid/1275
[2] http://www.visualstudio.com/downloads/download-visual-studio-vs#d-express-windows-desktop