Automation of measurements and macro management of a Vector Network Analyzer through SCPI commands

Pablo Hernández Sánchez

Julio 2025
Informe Técnico IT-CDT 2025-3

Observatorio de Yebes
Apdo. 148, E-19080
Guadalajara, Spain

Table of Contents

1 Introduction

A vector network analyzer (VNA) is an essential instrument in high-frequency and microwave engineering and research. It is primarily used to characterize electronic components, such as filters, amplifiers, antennas, and integrated circuits. Unlike scalar analyzers, a VNA measures both the amplitude and phase of reflected and transmitted signals simultaneously, enabling the complete and accurate determination of S-parameters (scattering parameters). This capability is critical for designing, analyzing, and validating electronic systems and RF circuits.

This project uses the Keysight PNA N5227B analyzer, which supports remote communication via SCPI (Standard Commands for Programmable Instruments) commands. This facilitates automation and integration into complex measurement processes.

The objective of this project is to develop a software application designed to automate and simplify macro management in the Keysight PNA N5227B analyzer through SCPI commands sent via an Ethernet connection. The application allows for the automated loading, deletion, and execution of macros, offering a simple interface that significantly reduces the technical barrier for non-specialized users. This minimizes human errors associated with manual operation and increases productivity in repetitive or highly complex technical measurement environments.

2 Development

2.1 Documentation

All development has been carried out following the official documentation provided by Keysight [Keysight, 2021], where you can find all the available SCPI commands that will enable the automation of any process in the vector analyzer. In addition, this documentation contains multiple macros ready to be imported and used, which can be used as examples to develop other custom macros that are better suited to the needs of each user.

2.2 Limitations

Until now, all measurements have been performed manually, which could be slow, especially in environments where each engineer could create and use a large number of specific macros according to their needs. It is in this context that having and managing a large number of macros can be extremely useful when carrying out measurements.

The problem arises in the number of spaces available in memory for macros, since one of the main limitations of the Keysight PNA N5227B analyzer is that it allows a maximum of 25 macros to be stored simultaneously in memory, which restricts advanced measurement automation, especially when multiple configurations or extensive measurement sequences are required.

It is important to distinguish between the "macros files" stored in the analyzer, which were previously created by engineers, and a macro which is a "macro file" already loaded into the analyzer's memory and is occupying one of the 25 available slots. Only those loaded in memory can be executed in the analyzer.

For this reason, a program is needed to load and remove macros files in the 25 available

slots easily. This allows users to quickly execute the macros they will use during their session.

2.3 Implementation

The main program consists of a Python script that establishes and maintains a TCP/IP connection, loads macro files into the analyzer's memory, and executes the macros by sending the appropriate SCPI commands. All operations are performed directly from a terminal, eliminating the need for a graphical user interface (GUI).

All macros defined and saved in the analyzer must have: a title, the path to the file containing all the sequences to be executed, and arguments, if necessary. These characteristics of the macros currently loaded in memory can be obtained using the following SCPI commands:

```
SYST:SHORtcut<1-25>:PATH?
SYST:SHORtcut<1-25>:TITLe?
SYST:SHORtcut<1-25>:ARGuments?
```

The macro files are loaded into the analyzer's memory using the following syntax:

```
SYST:SHORtcut<1-25>:PATH [Path]
SYST:SHORtcut<1-25>:TITLe [Title]
SYST:SHORtcut<1-25>:ARGuments [Arguments]
```

The following command is used to execute the macros that have been configured:

```
SYSTem:SHORtcut<1-25>:EXECute
```

All these commands are sent to the vector analyzer based on the options selected in the application's main menu, allowing macros to be managed without needing to know the complete syntax.

The Python application can be executed from any computer on which Python and all the necessary requirements are installed, as long as it is on the same local area network (LAN) as the PNA. When invoking the application, the user must supply the IP address of the analyser.

```
    $ ./macro_pna.py -a <ip>
```

Once the application has been started, an intuitive text menu (see listing 1) is displayed, showing all the available features and allowing the user to select them.

Listing 1: Application's main menu

```
Working with the instrument: Keysight PNA N5227b [ID]
Features----------------
 1. List all available macro files
 2. List the 25 macros currently loaded in memory
 3. Load macro files into memory
 4. Run macros
 5. Run custom SCPI scripts
 6. Interactive SCPI terminal
 7. Delete macros
 0. EXIT
```

All macro files must be placed in a specific directory within the analyzer in order to be loaded. These "macros files" are all Visual Basic scripts that can be run by the analyzer. Keysight provides some examples in their documentation to test the Python script and demonstrate its correct usage in the PNA.

The first script generates a new S21 measurement and displays it on the VNA screen.

```
Dim app
Dim scpi
' Create / Get the VNA application.
Set app = CreateObject("AgilentPNA835x.Application")
Set scpi = app.ScpiStringParser
' A comment
'Preset the analyzer
scpi.Execute ("SYST:FPReset")
' Create and turn on window 1
scpi.Execute ("DISPlay:WINDow1:STATE ON")
'Define a measurement name, parameter
scpi.Execute ("CALCulate:PARameter:DEFine:EXT 'MyMeas',S21")
'Associate ("FEED") the measurement name ('MyMeas') to WINDow (1), and give
    the new
TRACe a number (1).
scpi.Execute ("DISPlay:WINDow1:TRACe1:FEED 'MyMeas'")
```

Listing 2: S21 Measurment VisualBasic script

This macro example configures and displays Phase Sweep measurements.

```
   'Assume port 1 is connected to port 3
Set pna = CreateObject("AgilentPNA835x.Application")
Set SCPI = PNA.ScpiStringParser
'Create 3 trace S33, R3/C(amp),R3/C(phase)
SCPI.Parse("SYST:FPR")
SCPI.Parse("DISP:WIND:STATE ON")
SCPI.Parse("CALC:PAR:DEF 'MyMeas1',S33")
SCPI.Parse("DISP:WIND1:TRAC1:FEED 'MyMeas1'")
SCPI.Parse("CALC:PAR:SEL 'MyMeas1'")
SCPI.Parse("CALC:FORM SMIT")
SCPI.Parse("CALC:PAR:DEF 'MyMeas2',R3C,3")
SCPI.Parse("DISP:WIND1:TRAC2:FEED 'MyMeas2'")
SCPI.Parse("CALC:PAR:SEL 'MyMeas2'")
SCPI.Parse("CALC:FORM MLOG")
SCPI.Parse("CALC:PAR:DEF 'MyMeas3',R3C,3")
SCPI.Parse("DISP:WIND1:TRAC3:FEED 'MyMeas3'")
SCPI.Parse("CALC:PAR:SEL 'MyMeas3'")
SCPI.Parse("CALC:FORM PHAS")
SCPI.Parse("SENS:SWE:TYPE PHAS")
'turn on 3 and 1
SCPI.Parse("SOUR:POW1:MODE ON")
SCPI.Parse("SOUR:POW3:MODE ON")
'set port3's control parameter to R3/C
SCPI.Parse("SOUR:PHAS3:PAR 'R3/C'")
'Set port3 to PAR mode
SCPI.Parse("SOUR:PHAS3:PAR:MODE PAR")
SCPI.Parse("SOUR:PHAS3:PAR:PORT 1")
SCPI.Parse("SOUR:PHAS3:POFF:FIX 3")
SCPI.Parse("SOUR:PHAS3:STAR 0")
SCPI.Parse("SOUR:PHAS3:STOP 180")
```

Listing 3: Phase Sweep measurment VisualBasic script

3 Conclusions

This application can be extremely useful, as it greatly facilitates both the daily use of the vector analyzer and the performance of measurements by laboratory technical staff, thereby reducing the learning curve. In addition, it provides engineers with an effective tool for automating and speeding up complex tasks through the creation and management of custom macros, enabling accurate results to be obtained in a faster, more repeatable, and more efficient manner.

A Source Code

```python
import socket

class Command_Ethernet:
    PORT = 5025

    def __init__(self, host, timeout=3):
        self.host = host
        self.socket = socket.socket(socket.AF_INET,
                                    socket.SOCK_STREAM,
                                    socket.IPPROTO_TCP)
        #self.timeout = 0
        self.set_timeout(timeout)

    def connect(self):
        self.socket.connect((self.host, self.PORT))

    def close(self):
        self.socket.close()

    def write(self, cmd):
        self._send(cmd)

    def read(self, num_bytes=1024):
        # self._send('READ?')
        return self._recv(num_bytes)

    def query(self, cmd, buffer_size=1024*1024):
        self.write(cmd)
        return self.read(buffer_size)

    def set_timeout(self, timeout):
        if timeout < 1e-3 or timeout > 3:
            raise ValueError('Timeout must be >= 1e-3 (1ms) and <= 3 (3s)')

        self.timeout = timeout
        self.socket.settimeout(self.timeout)

    def _send(self, value):
        encoded_value = ('%s\n' % value).encode('ascii')
        self.socket.send(encoded_value)

    def _recv(self, byte_num):
        value = self.socket.recv(byte_num)
        return value.decode('ascii')
```

Listing 4: TCP/IP connection management

```python
#!/usr/bin/python3

import os
import argparse
import instrument_ethernet as it
import re

SCRIPTS_DIR = 'scripts'
INSTRUMENT_ID = ""
MACROS_PATH = "C:\\Users\\Instrument\\Desktop\\users\\PabloH\\Macros"

```

```
def clear_screen():
    """Clears the terminal screen."""
    os.system('cls' if os.name == 'nt' else 'clear')

def continue_prompt():
    input("Press Enter to continue...")

def replace_with_array(string, array):
    """Replaces %n in the string with the nth element of the array."""
    def replacement(match):
        index = int(match.group(1))
        return str(array[index]) if 0 <= index < len(array) else match.
            group(0)
    return re.sub(r'%(\d+)', replacement, string)

def process_script(cm, filename, args=[]):
    """Processes a script file and executes the commands on the instrument.
        """
    resp = ""
    try:
        with open(filename, 'r') as f:
            for line in f:
                line = line.strip()
                if not line or line.startswith("#"):
                    continue
                parts = line.split(";", 1)
                if len(parts) != 2:
                    print(f"Invalid line in {filename}: {line}")
                    continue
                command_type, cmd = parts[0].strip().upper(), parts[1].
                    strip()
                if args:
                    cmd = replace_with_array(cmd, args)
                if command_type == "W":
                    cm.write(cmd)
                elif command_type == "R":
                    # The READ command is not standard in PNA, it is
                        omitted.
                    print("'R' command not supported in PNA. Use SCPI query
                         commands ending with '?'.")
                elif command_type == "Q":
                    resp = cm.query(cmd)
                    print(f"Query: {cmd} -> Response: {resp}", end="")
                elif command_type == "C":
                    process_script(cm, f'./{SCRIPTS_DIR}/{cmd}')
                elif command_type in ("DEBUG", "D"):
                    print("Entering interactive terminal mode (type QUIT to
                         exit)...")
                    while True:
                        user_cmd = input("PNA> ").strip()
                        if user_cmd.upper() == "QUIT":
                            print("Exiting terminal mode.")
                            break
                        if user_cmd:
                            resp = cm.query(user_cmd)
                            print(f"Response: {resp}")
                else:
                    print(f"Unknown command type in {filename}: {line}")
    except Exception as e:
        print(f"Error processing file {filename}: {e}")
```

```
66      return resp
67
68  def list_available_macros(cm):
69      """Lists the macros available in the macros directory."""
70      print("Listing available macros")
71      resp = process_script(cm, SCRIPTS_DIR + "/ListFiles.cmd", [MACROS_PATH
            ])
72      print("Files in directory:")
73      macros = resp.replace("\"", "").split(",")
74      if not macros or macros == ['']:
75          print("No available macros found.")
76      else:
77          for idx, macro in enumerate(macros, start=1):
78              if macro.strip():
79                  print(f"{idx}. {macro.strip()}")
80      return macros
81
82  def list_curr_macros(cm):
83      """Lists the macros currently loaded in the instrument."""
84      print("Listing loaded macros")
85      for i in range(1, 26):
86          args = [i]
87          check_macros_file = "CheckMacros.cmd"
88          print(f"\nMacro {i} --------------------")
89          process_script(cm, SCRIPTS_DIR + "/" + check_macros_file, args)
90
91  def change_macro(cm):
92      """Allows swapping/loading a macro in the instrument."""
93      print("Macro management")
94      macro_list = list_available_macros(cm)
95      macro_number = input("Macro number to load: ")
96      if not macro_number.isdigit() or not (1 <= int(macro_number) <= len(
            macro_list)):
97          print("Invalid macro number.")
98          return
99      macro_title = input("Macro title: ")
100     macro_arguments = input("Macro arguments (comma separated): ")
101     new_macro_number = input("Slot number to load the macro (1-25): ")
102     if not new_macro_number.isdigit() or not (1 <= int(new_macro_number) <=
             25):
103         print("Invalid destination macro number.")
104         return
105     process_script(
106         cm,
107         SCRIPTS_DIR + "/SetMacro.cmd",
108         [new_macro_number, f"{MACROS_PATH}\\{macro_list[int(macro_number)
                -1]}", macro_title, macro_arguments]
109     )
110     print(f"Macro '{macro_list[int(macro_number)-1]}' loaded as macro {
            new_macro_number} with title '{macro_title}' and arguments '{
            macro_arguments}'.")
111
112 def exec_macro(cm):
113     """Executes a macro loaded in the instrument."""
114     print("Macro execution")
115     macro_number = input("Macro number to execute: ")
116     if not macro_number.isdigit() or not (1 <= int(macro_number) <= 25):
117         print("Invalid macro number. Must be a number between 1 and 25.")
118         return
119     process_script(cm, SCRIPTS_DIR + "/ExecMacro.cmd", [macro_number])
```

```
120     print(f"Macro {macro_number} executed.")
121
122 def delete_macro(cm):
123     """Deletes one or all macros loaded in the instrument."""
124     print("Macro deletion")
125     macro_number = input("Macro number to delete (1-25) or 0 to delete all:
            ")
126     if not macro_number.isdigit() or not (0 <= int(macro_number) <= 25):
127         print("Invalid macro number. Must be a number between 0 and 25.")
128         return
129     if macro_number == '0':
130         for i in range(1, 26):
131             process_script(cm, SCRIPTS_DIR + "/SetMacro.cmd", [str(i), "",
                    "", ""])
132         print("All macros deleted.")
133     else:
134         process_script(cm, SCRIPTS_DIR + "/SetMacro.cmd", [macro_number, ""
                , "", ""])
135         print(f"Macro {macro_number} deleted.")
136
137 def menu():
138     """Displays the main menu."""
139     clear_screen()
140     print(f"Working with the instrument: {INSTRUMENT_ID}")
141     print("Features----------------")
142     print(" 1. List all available macro files ")
143     print(" 2. List the 25 macros currently loaded")
144     print(" 3. Load macro files into memory")
145     print(" 4. Run macros")
146     print(" 5. Run custom SCPI scripts")
147     print(" 6. Interactive SCPI terminal")
148     print(" 7. Delete macros")
149     print(" 0. EXIT")
150
151 def menu_loop(cm):
152     """Main menu loop."""
153     menu_actions = {
154         1: list_available_macros,
155         2: list_curr_macros,
156         3: change_macro,
157         4: exec_macro,
158         5: lambda cm: process_script(cm, input("File: ")),
159         6: lambda cm: process_script(cm, SCRIPTS_DIR + "/Debug.cmd"),
160         7: delete_macro
161     }
162     while True:
163         menu()
164         try:
165             menu_opt = int(input("\nSelection: "))
166         except ValueError:
167             print("Enter a valid number.")
168             continue
169         if menu_opt == 0:
170             clear_screen()
171             print("Bye :)")
172             break
173         action = menu_actions.get(menu_opt)
174         if action:
175             action(cm)
176             continue_prompt()
```

```
177             else:
178                 print(f"Value '{menu_opt}' not recognized\n")
179
180 if __name__ == "__main__":
181     parser = argparse.ArgumentParser(description="Execute commands on the
            PNA N5227B through scripts.")
182     parser.add_argument("-a", "--address", required=True, help="IP address
            of the PNA N5227B instrument")
183     parser.add_argument("-f", "--files", required=False, nargs="+", help="
            List of script files to execute")
184     args = parser.parse_args()
185
186     cm = it.Command_Ethernet(args.address)
187     try:
188         cm.connect()
189     except Exception as e:
190         print(f"Connection error. Check that the equipment is turned on.
                Details: {e}")
191         exit(1)
192
193     id_file = "ID.cmd"
194     INSTRUMENT_ID = process_script(cm, SCRIPTS_DIR + "/" + id_file)
195
196     # Execute scripts passed by command line before entering the menu
197     if args.files:
198         for script_file in args.files:
199             process_script(cm, script_file)
200
201     menu_loop(cm)
```

Listing 5: Macro management main script

References

Keysight. Vna help file. Technical report, Keysight, 2021.