Application Report
SBAA397–July 2019

Controlling Latte GUI from Python

Ahmad Dweik

1 Trademarks

All trademarks are the property of their respective owners.

2 Introduction

TI Latte is a Python-based application that allows users to interface with several different devices, including the AFE77xx EVM, for testing and evaluation purposes. Within the application, there are multiple executable scripts for the specified EVM device. For the AFE77xx, some of these scripts include “setup.py” and “basicBringup.py” as shown in Figure 1.

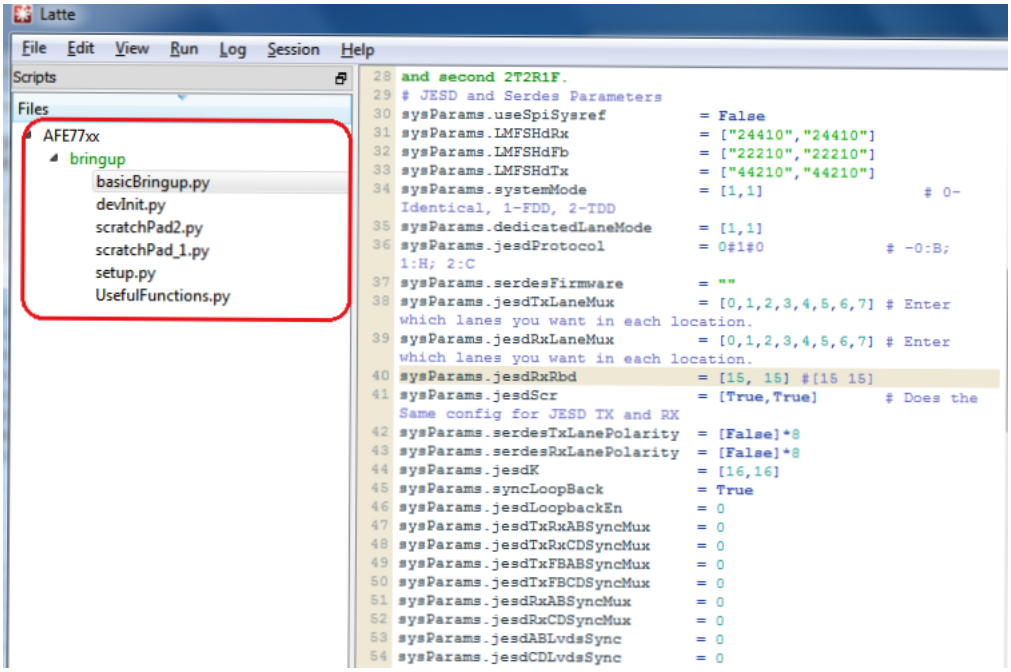

Figure 1. Example Latte Scripts

Although the EVM user interface looks very basic, the Python-based interface can implement various functions like register writes, math calculations, data analysis, and plotting, just to name a few. The tool that facilitates this process is the Socket.py script. The Socket allows communication between two different processes on the same or different machines. Typically, the “client” connects to the “server” to run a command or make a request for information.

The Socket Python code allows the user to code in the Python IDLE while also being able to access all the features in the Latte GUI. The ability to connect the two interfaces is powerful, allowing users to take advantage of the features offered by each platform.

The Socket.py script is useful in situations where certain functions within the Latte GUI do not work properly independently, like communicating with measurement instruments such as spectrum analyzers and signal generators. With the Socket.py script, you are able to write your entire code in the Python IDLE, while being able to bring-up the device under test with Latte commands sent with the Socket.py code.

3 Linking Python to Latte

3.1 Setting up Latte GUI

3.1.1 Creating a server

1. Put Latte in Server Mode (Figure 2).
2. Click on Run >> Create Server.

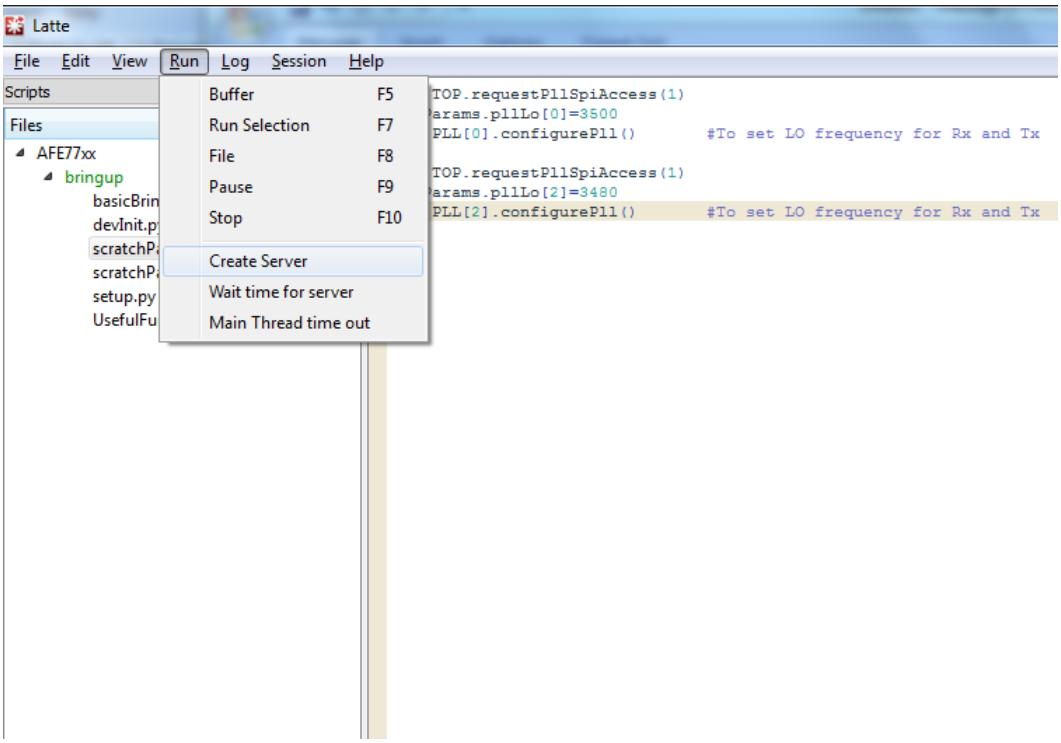

Figure 2. Creating Server in Latte

3. Enter any desired valid port number (1 to 9999) and click "OK".

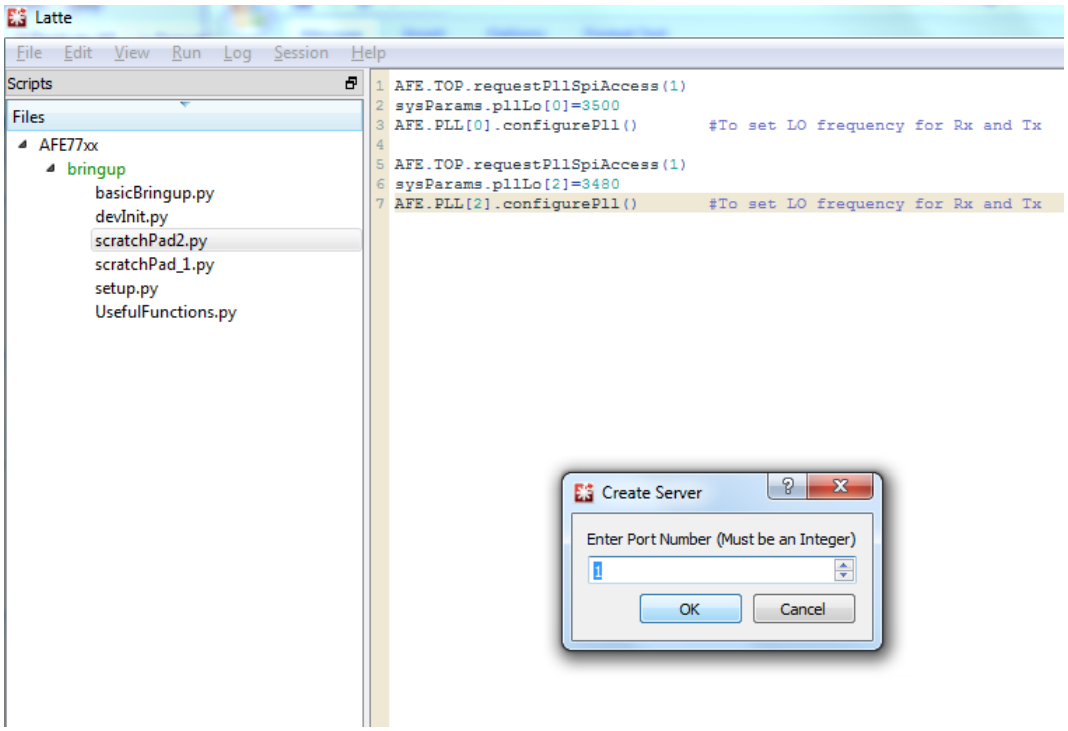

Figure 3. Entering Port Number in Latte

4. The message in Figure 4 appears in the Log section of Latte if the server was created successfully. If the server was not created successfully, locate the "Session" tab on the top menu in latte, clear the log workspace, and try again.

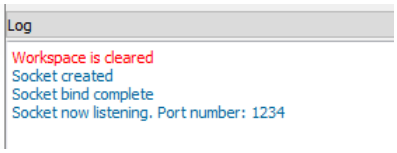

Figure 4. Successful Connection Message

3.2 Setting Up the Client

1. Download the bSocket.py script from AFE webpage or request the script from your local FAE support.
2. Open bSocket.py in a Python editor.
3. Locate variable “port” and set the value to the same port number set in Latte as shown in Figure 5.

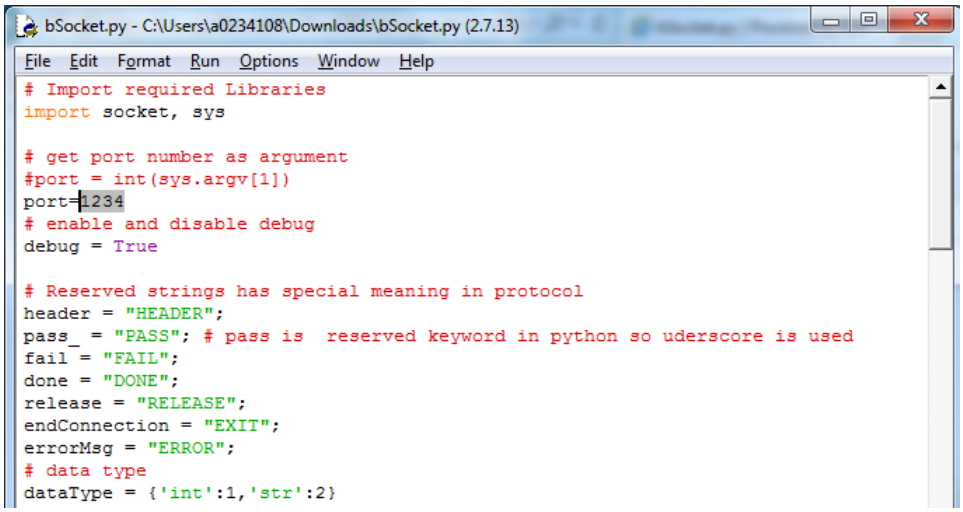

Figure 5. Setting Port Value in the Socket.py Script

4. Locate the “debug” variable and replace the value with “True”. This enables messages to be reported by the socket interface.

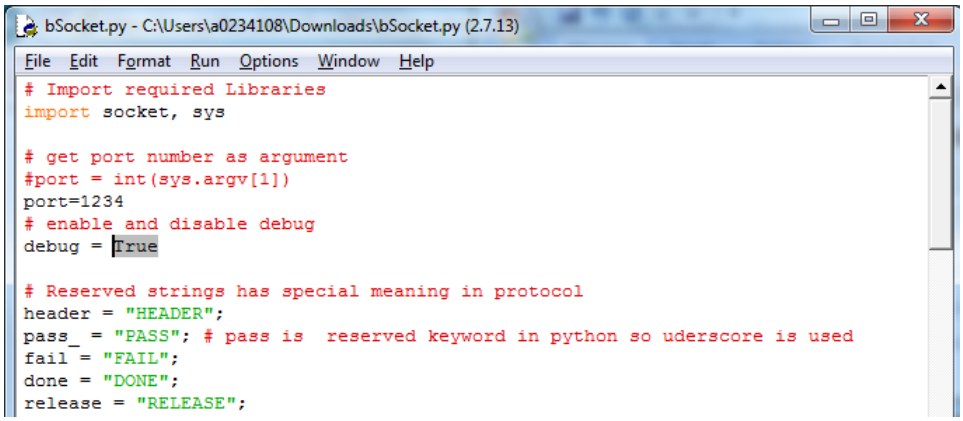

Figure 6. Changing "Debug" Variable to True

5. Save the script, then run the program.
   a. If the client failed to connect with the server, the Python shell window displays the message shown in Figure 7.

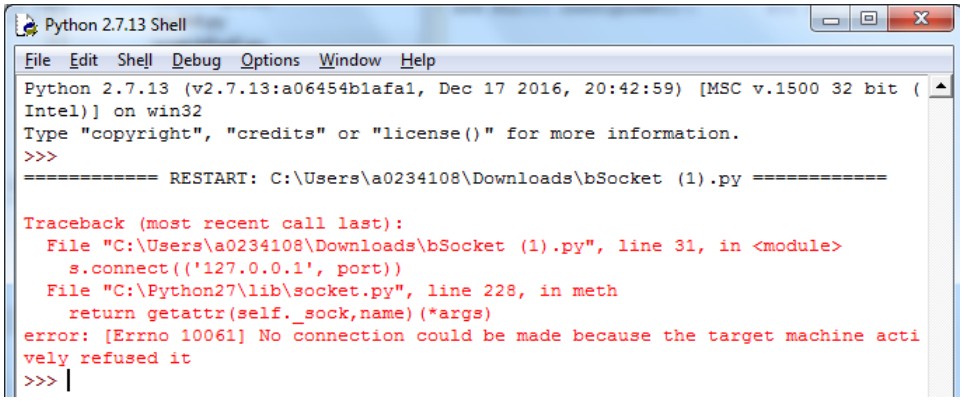

Figure 7. Error Connecting to Server

4 Running a Latte Command

After setting up the server and client successfully, and running the Socket.py script, a prompt “Please enter the Latte command:” is displayed.

1. Type any valid Latte command and click enter to execute the command.
2. Verify that the command is valid in Latte prior to sending it through the Python editor. If there is an error with the command execution, it is displayed in Latte log window.

A list of Latte commands can be found in the following document:

AFE77XX_LatteGUI_Application Note.docx, located under resourceFiles folder in Latte install directory.

An example is shown below:

- A valid command, used to change the LO frequency of the AFE77xx device (Figure 8)

```
Socket created
Socket bind complete
Socket now listening. Port number: 1234
Connected with 127.0.0.1: 50914
pll0: True; LO Frequency: 3500.0
The server disconnected from client. Ready to connect with another client. Port number: 1234
```

Figure 8. Successful Command Executed by Latte

- An invalid command, rejected by Latte (Figure 9)

```
Socket created
Socket bind complete
Socket now listening. Port number: 1234
Connected with 127.0.0.1: 50836
name 'AFE' is not defined
Main Thread takes time more than expected
Disconnecting server from client...
```

Figure 9. Command Message Error

5 Running a Latte Script

1. To run a script using bSocket, run the Socket.py script, then enter the following command: mainWindow.runFile(r"Absolute Path of Python File.py")
   a. Replace "Absolute Path of Python File.py" with the path of the script that needs to be executed.

Figure 10 shows an example.

```
========== RESTART: C:\Users\a0234108\Downloads\bSocket (1).py ==========
Please enter the Latte command: mainWindow.runFile(r"C:\Users\a0234108\Documents\Texas Instruments\Latte\projects\AFE77xx\bringup\scratchPad2.py")
Command Size: 114
Command:  mainWindow.runFile(r"C:\Users\a0234108\Documents\Texas Instruments\Latte\projects\AFE77xx\bringup\scratchPad2.py")
Command type:  <type 'str'>
Encoded command for Latte: HEADER€€€€òmainWindow.runFile(r"C:\Users\a0234108\Documents\Texas Instruments\Latte\projects\AFE77xx\bringup\scratchPad
2.py")€€€€€Ôƒ
cmd sent successfully
Header received succussfully
Number of bytes to receive:  4
Data type: 2
Acknowledge received
Header received succussfully
Number of bytes to receive:  4
Data type: 2
Latte completed task
Header received succussfully
```

Figure 10. Running a Script in Python Shell

6 Controlling Timeout

- Gives ability to control the timeout for execution
- Click run >> Main Thread Timeout.

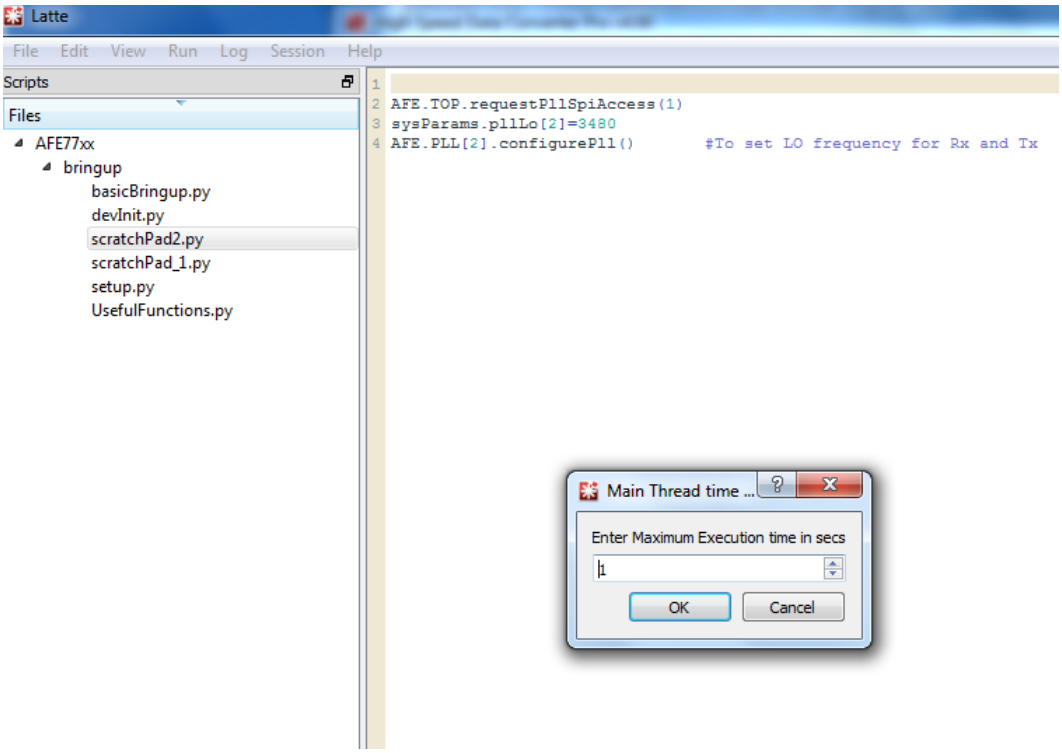

Figure 11. Adjusting the Thread Timeout

- Default timeout is 60 s, but can be adjusted to any desired value less than 1000 s.

7 Application

The bSocket Python script is very useful because of its ability to be implemented in almost any program. By making a few slight adjustments, it is easily reusable for any Python-based code. Being able to control a network of devices with one program is powerful, and saves the user a lot of time. The rest of the application report shows an example.

```
def LatteCommand(command):
        # Import required Libraries
        import socket, sys

        # get port number as argument
        #port = int(sys.argv[1])
        port=ServerNum
        # enable and disable debug
        debug = True

        # Reserved strings has special meaning in protocol
        header = "HEADER";
        pass_ = "PASS"; # pass is  reserved keyword in python so uderscore
        fail = "FAIL";
        done = "DONE";
        release = "RELEASE";
        endConnection = "EXIT";
        errorMsg = "ERROR";
        # data type
        dataType = {'int':1,'str':2}

        # predecided sizes
        cmdSizeInt = 5 # bytes
        checksumInt = 8 # bytes
        dataTypeSize = 1 # byte
        intBits = 16 # 16bit integers
```

Figure 12. Adding the Socket.py Script as a Function in the Main Python Script

1. Socket script is copied into the main program as function “LatteCommand”.
   a. Instead of taking a user input as the Socket.py originally does, you pass the command as a variable to the function as shown in Figure 13.

```
        # Ask user for a command
        cmd = command
        try:
                #encode size of data to send
                lenCmd = len(cmd)
                if(debug):
                        print 'Command Size: {}'.format(lenCmd)
                        print 'Command: ',cmd
                        print 'Command type: ',type(cmd)
                encodedCmdSize = encodeData(lenCmd,cmdSizeInt)
```

Figure 13. Changing the Function from User Input to Variable Assignment

2. In the code, you can assign the variable to a specific command and pass that into the function as shown in Figure 14.

```
with open('SpurAssessment.csv', mode='wb') as f:              #Opening a CSV File
        writer=csv.writer(f)
        writer.writerow(['TX LO' , 'RX LO', 'Fund Pwr' , '+2xF' , '+2xP' , '+4xF' , '+4xP' ,'+6xF' , '+6xP' ,'-2xF' , '-2xP' ,'-4xF' , '-4xP' ,'-6xF' , '-6xP'])


        #This loop tuns through the TX values set from the user
        while(TXStart<=TXStop):
            RX=TXStart-5*DeltaValue
            bSocket(command="AFE.TOP.requestPllSpiAccess(1)") #AFE board commands run through latte using the socket function
            bSocket(command="sysParams.pllLo[0]="+str(TXStart))       #Set LO ; convert to MHz
            bSocket(command="AFE.PLL[0].configurePll()")

            #Nested loop to sweep the RX LO Values
            while (RX<=(TXStart-DeltaValue)):
                LatteCommand(command="AFE.TOP.requestPllSpiAccess(1)")
                LatteCommand(command="sysParams.pllLo[2]="+str(RX))       #Set LO ; convert to MHz
                LatteCommand(command="AFE.PLL[2].configurePll()")
```

Figure 14. Passing Several Commands into the Function

As shown in Figure 14, you can write several commands using this method.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265