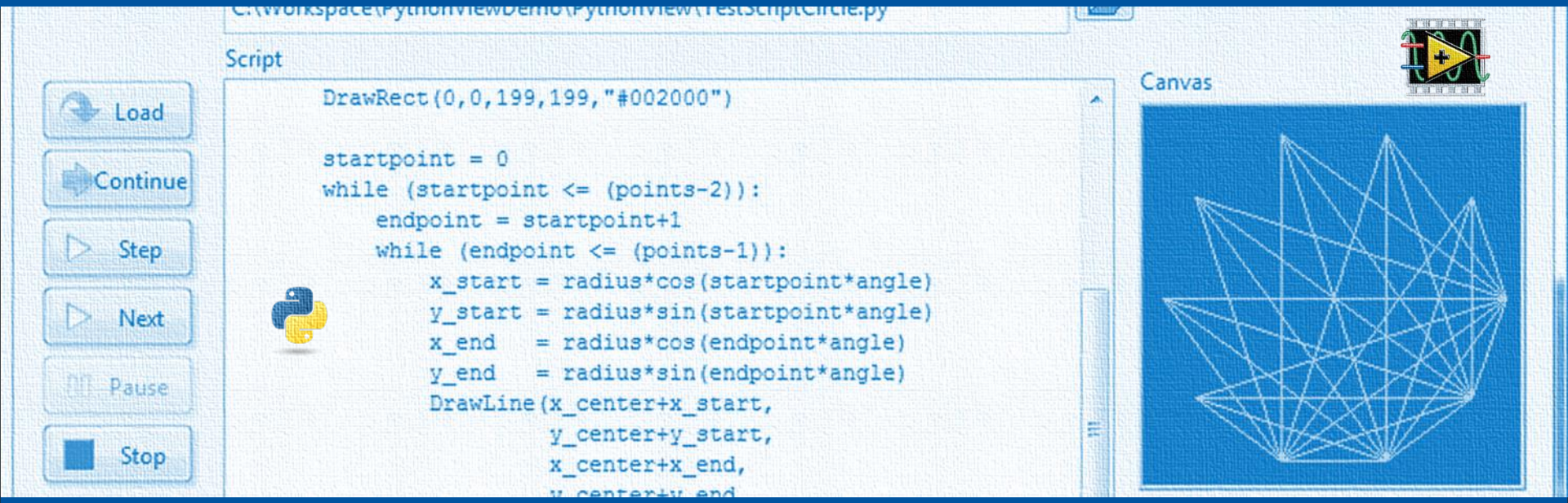
electronics & embedded systems

CO-DEVELOPMENT MANUFACTURING INNOVATION & SUPPORT



LabVIEW Scripting Using Python

NIDays 2015

Aschwin van de Haar

(Consulting Engineer)

aschwin.van.de.haar@3t.nl

your co-development partner

We develop, manufacture, innovate and support, customer specific electronics and embedded software through close cooperation with our customers.

Company profile

3T: leading for more than 30 years

- Founded in 1982, 3T since 1994
- Management buy-out in October 2014
- Turnover: 7.1 Meur (2014), Growing year over year
- ISO 9001:2008 certified , ISO 13485:2003 in progress
- 45-50 employees (mainly MSc/BSc)
- Offices in Enschede (HQ) and Eindhoven

Alliance Partner
National Instruments

National Instruments Alliance Partner since 1988

Contents

- Challenge
- Solution
- Architecture
- Demo
- Results
- Performance
- Conclusion
- Questions

Challenge

“Add Python script engine to existing LabVIEW application.”

- Existing functionality (from LabVIEW application) must be accessible via the Python script
- Execution of the python script must be fully controllable via LabVIEW environment (Start/Step/Pause/Abort/Breakpoints).

Challenge

Available LabVIEW-Python solution:

LabPython

- Runs Python scripts from LabVIEW, but…
- Script execution not controllable by LabVIEW (only run-and-wait-until-done)
- Does not facilitate LabVIEW call backs.

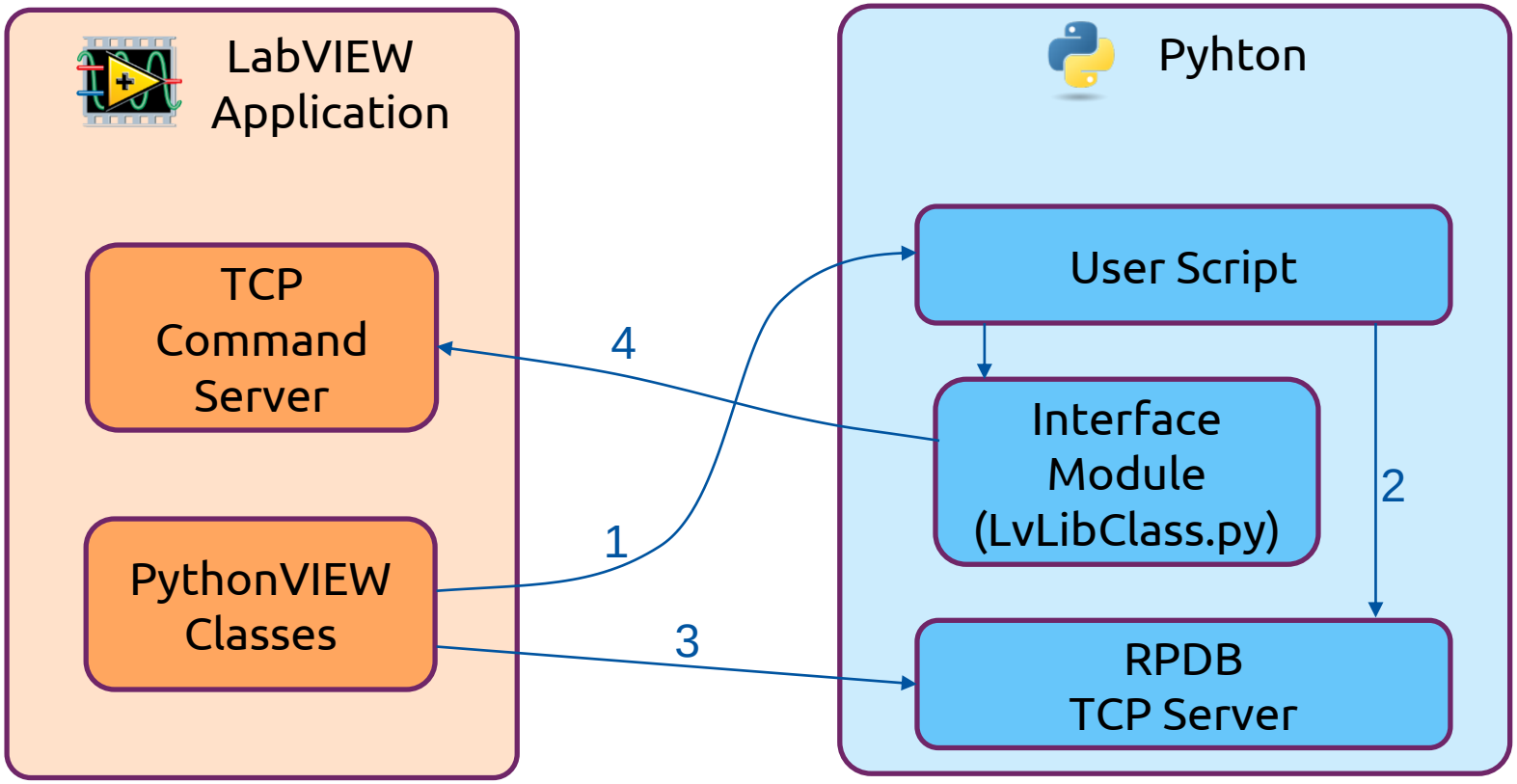
Solution

PythonVIEW

Basic operation:

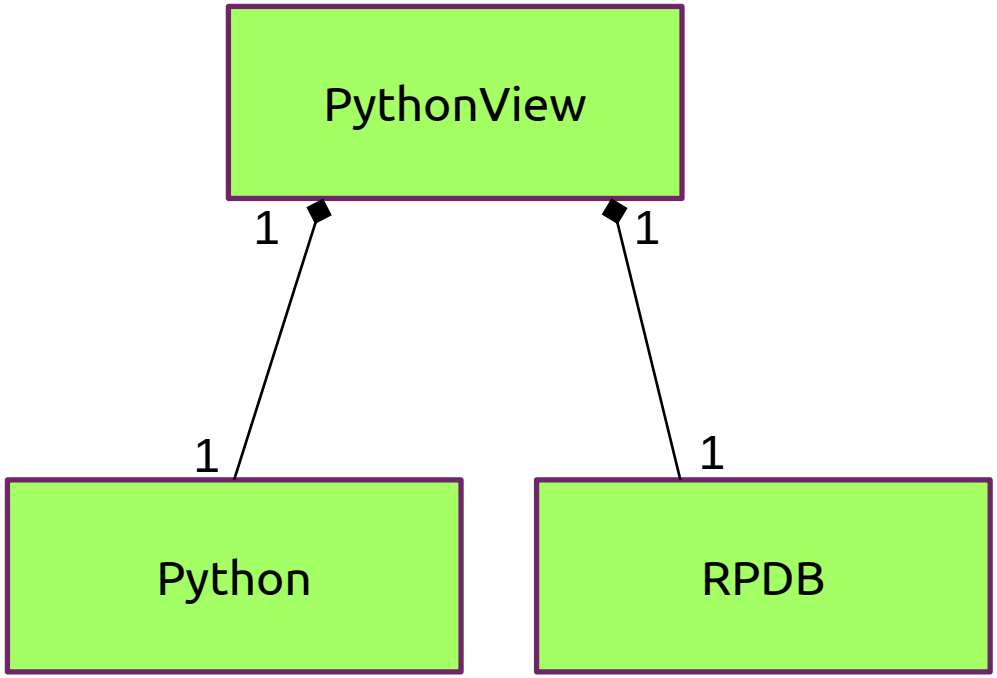
1) LabVIEW application starts a TCP Command Server with functionality to be published to the Python script.
2) LabVIEW starts an user defined Python script. Via the LvLibClass.py module, the user script can access the TCP Command Server.
3) The user script is started using the Remote Python Debugger (RPDB) server. Via this server, the LabVIEW application can control the execution of the user script.

Architecture

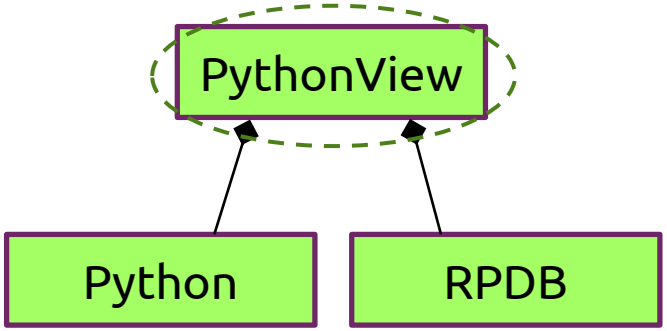


1. Start user script
2. Scipt pauses in debug mode.
3. PythonVIEW controls script execution via debugger
4. Script calls LabVIEW commands Via LvLibClass.py

PythonVIEW Class Diagram



PythonVIEW Class



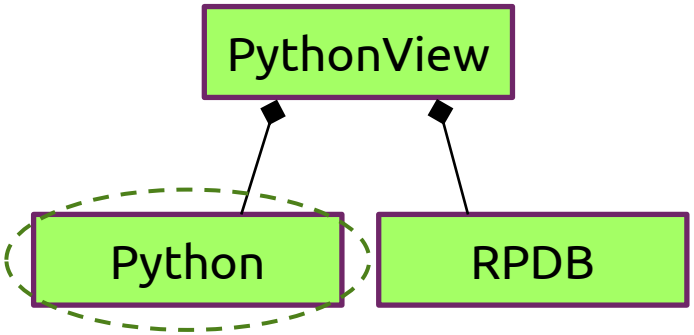
Main purpose:

- State machine for python script management
- Simple operation interface for host application

Available Public Methods:

- LoadScript
- Next (steps over current line)
- Step (step into current line)
- Run/Continue
- Pause
- Stop
- JumpToLine#
- ToggleBreakpoint

Python Class

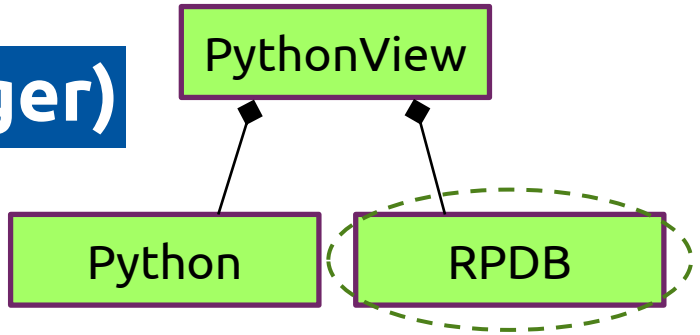


Used by PythonVIEW class only.

Main purposes:

- Launch of Python and user script
- Monitors state the of execution of Python
- Captures Python standard output, in case Python crashes

RPDB (Remote Python DeBugger)



Used by PythonVIEW class only.

Main purposes:

- Manage connection to RPDB Server
- Interface to RPDB commands

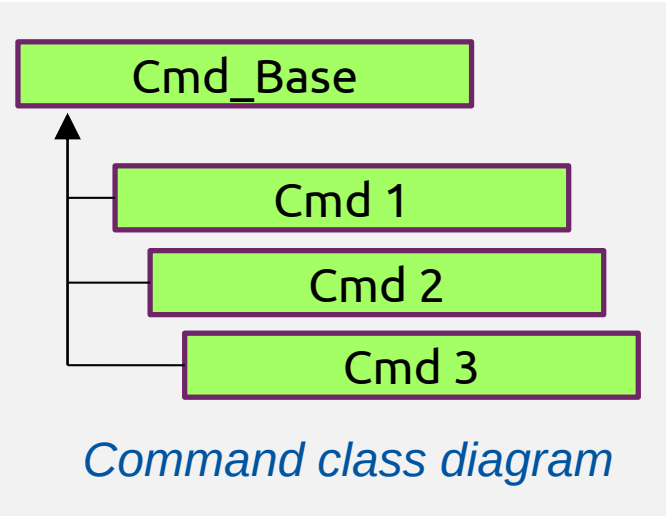
(more on RPDB on later sheets)

TCP Command Server

Serves LabVIEW commands, via a TCP socket. To be called from Python user script

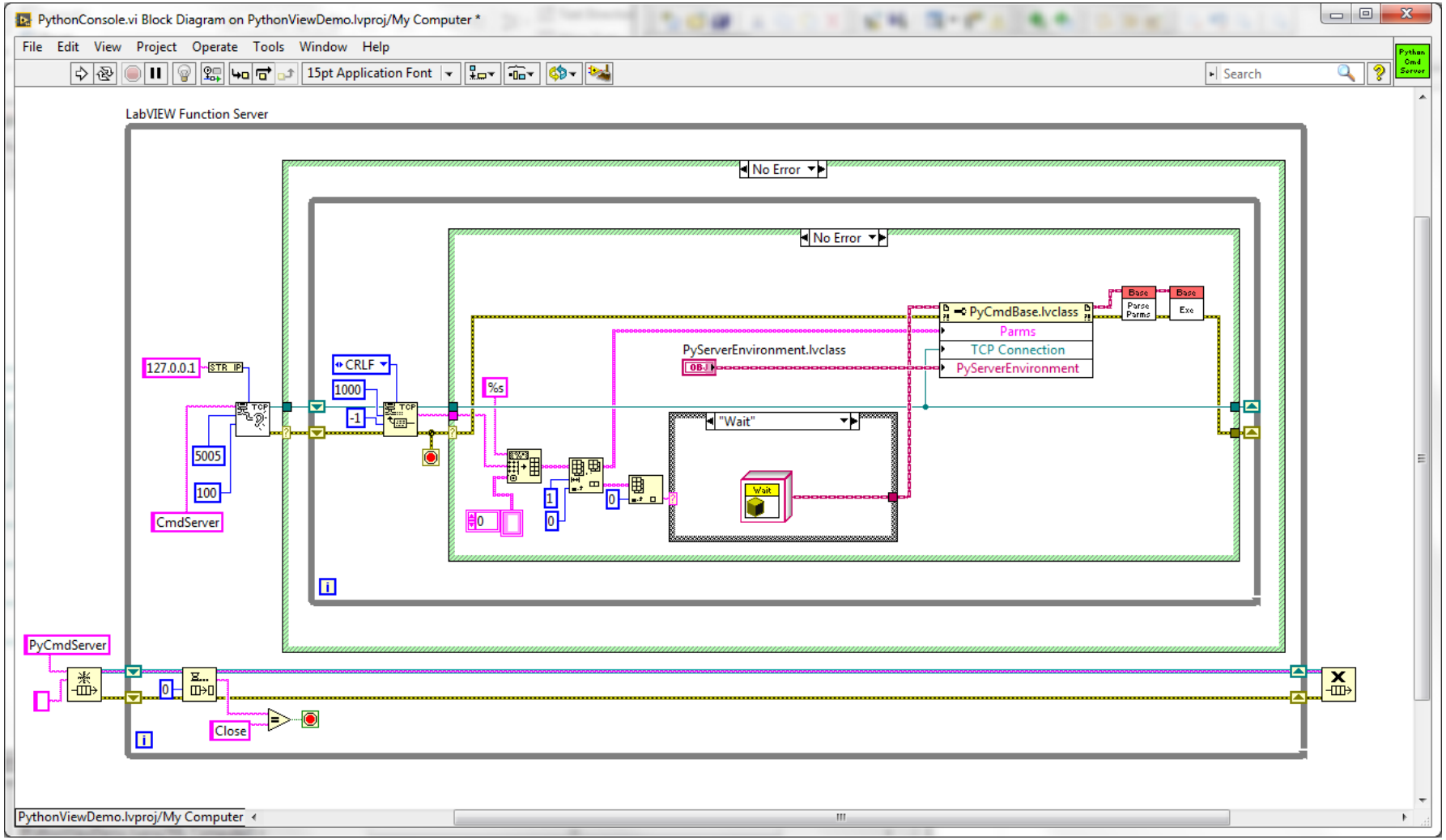
Main purpose:

- Captures incoming calls and performs initial parameter parsing
- Creates corresponding command object
- Passes the parameters to command object for further parsing
- Executes the command
- Sends result back to python script

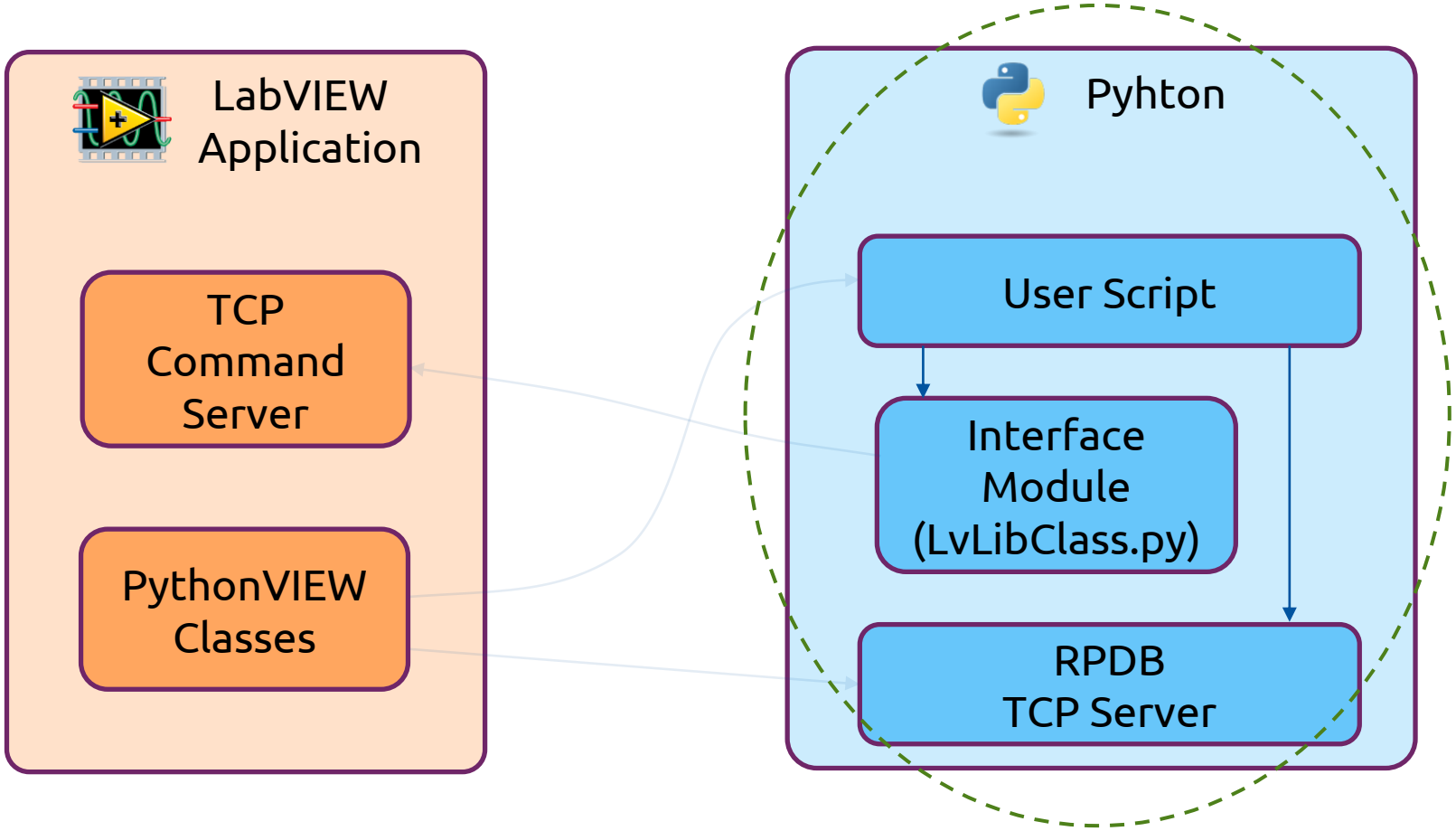


Command class diagram

TCP Command Server



And now for the Python side….



RPDB (Remote Python DeBugger)

Wrapper around PDB module that re-routes stdin and stdout to a TCP socket handler.

Used module: RPDB v0.1.5 (Bertrand Janin)

This module allows for:

- Full control over script execution
- Get current script location

LvLibClass.py

Provides Python interface to commands from LabVIEW Command Server

For example:

```
def Wait(a_TimeInMs):
    astring = "Wait\t%s\r\n" % (a_TimeInMs)
    s.sendall(astring.encode('utf-8'))
    data = s.recv(1024).decode("utf-8")
    if (data!='Ok'):
        raise SyntaxError(data)
```

ScriptLoader.py

Helper script that handles overhead for RPDB and connecting/disconnecting to LabVIEW Command Server.

This keeps the user script clean.

Contents:

```
import rpdb
import LvLibClass
#TestScript is imported via RPDB interface

rpdb.Rpdb () .set_trace ()
LvLibClass.Connect()
TestScript.TestMain()
LvLibClass.Disconnect())
```

Script execution is handed over to the debugger, and thus LabVIEW in our case.

User script

Required Imports: LvLibClass

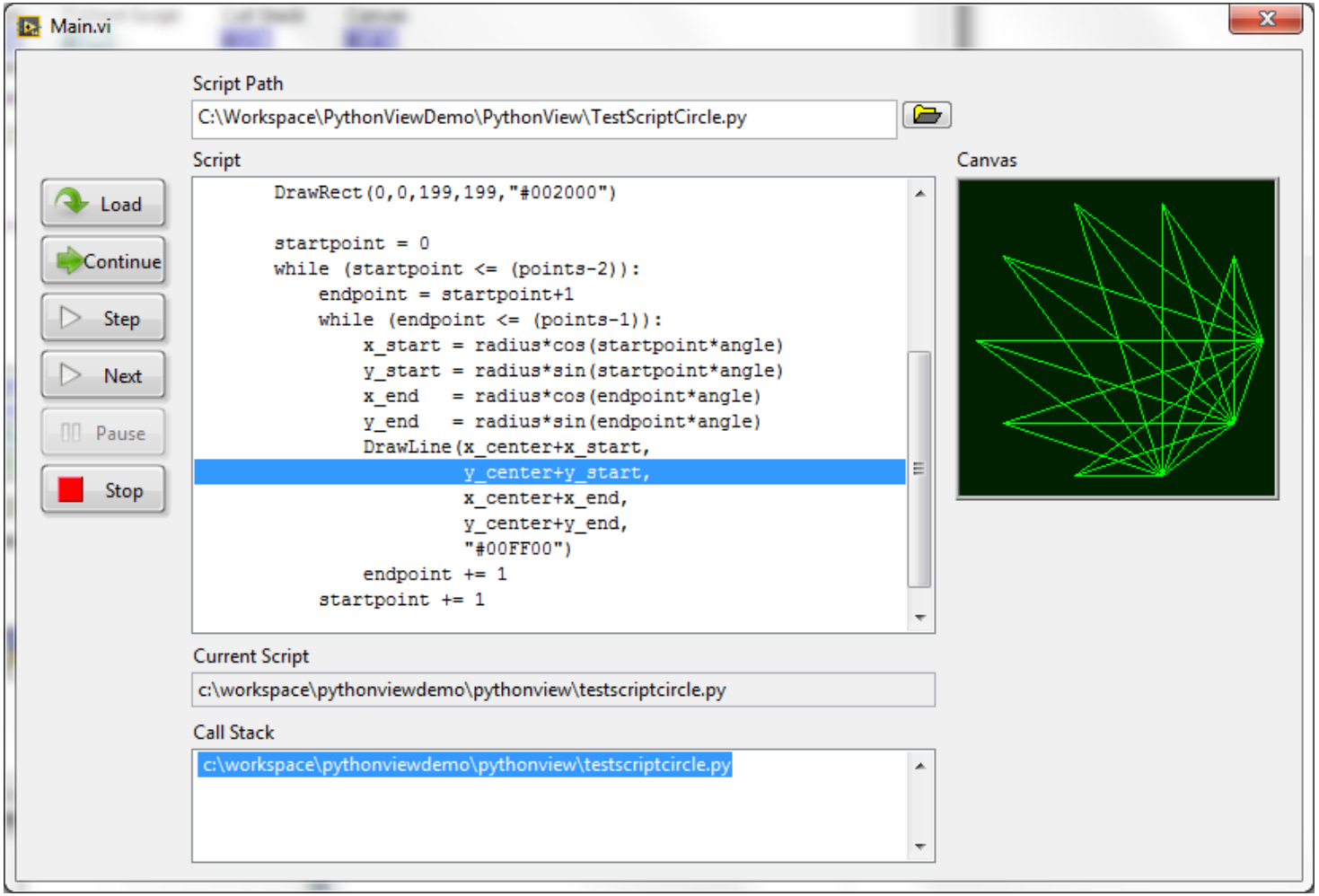
Required Main Function name: TestMain()

Example script:

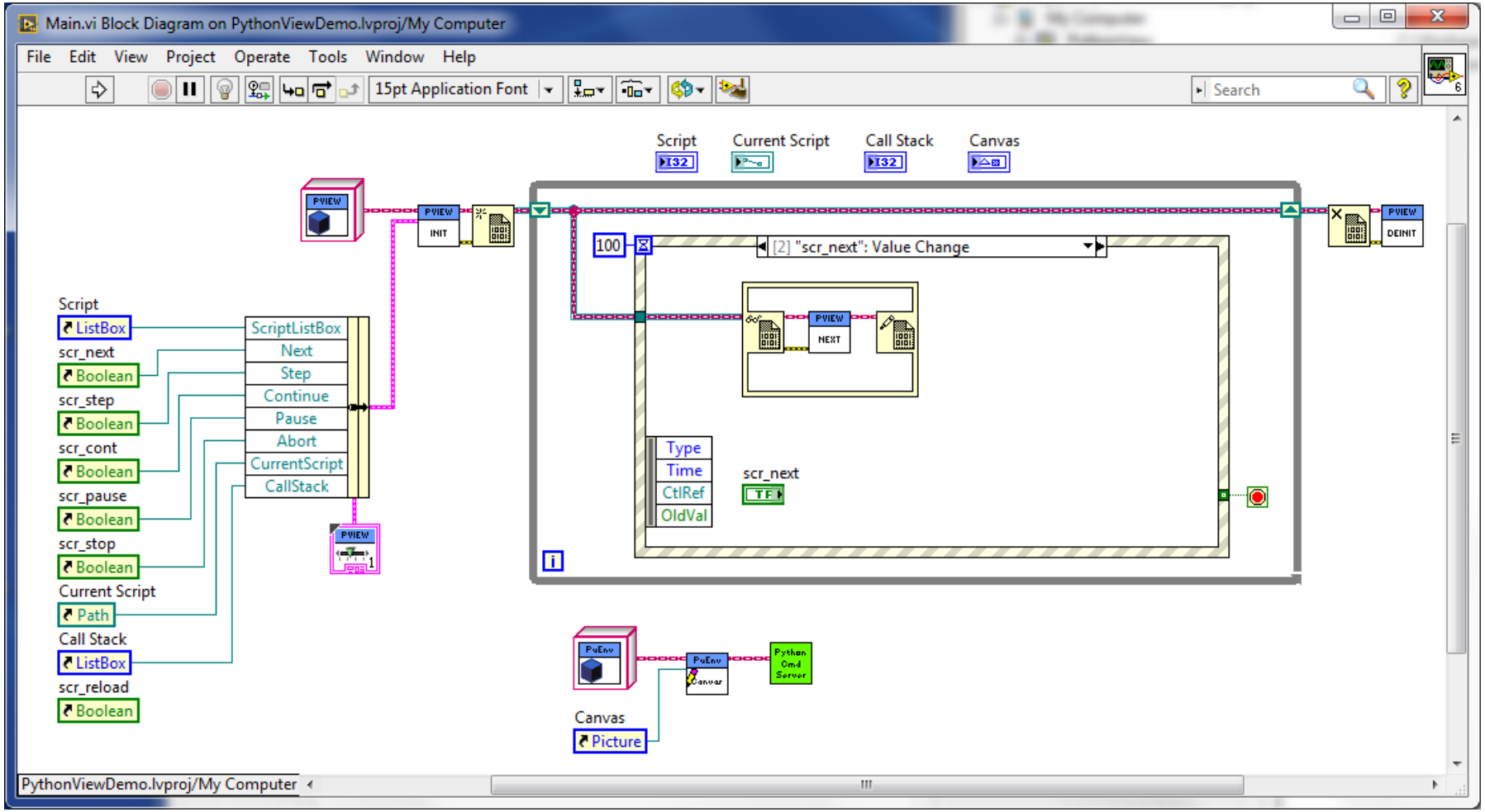
```
from LvLibClass import *

def TestMain():
    Wait(1000)
```

Is not a Python Wait command,
but the LabVIEW Wait command!

Demo


Main.vi
Script Path
C:\Workspace\PythonViewDemo\PythonView\TestScriptCircle.py
Script
Canvas
Load
Continue
Step
Next
Pause
Stop
DrawRect(0,0,199,199,"#002000")
startpoint = 0
while (startpoint <= (points-2)):
endpoint = startpoint+1
while (endpoint <= (points-1)):
x_start = radius*cos(startpoint*angle)
y_start = radius*sin(startpoint*angle)
x_end = radius*cos(endpoint*angle)
y_end = radius*sin(endpoint*angle)
DrawLine(x_center+x_start,
y_center+y_start,
x_center+x_end,
y_center+y_end,
"#00FF00")
endpoint += 1
startpoint += 1
Current Script
c:\workspace\pythonviewdemo\pythonview\testscriptcircle.py
Call Stack
c:\workspace\pythonviewdemo\pythonview\testscriptcircle.py

Demo


Main.vi Block Diagram on PythonViewDemo.lvproj/My Computer
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
Script
Current Script
Call Stack
Canvas
PVIEW
INIT
DEINIT
NEXT
100
[2] "scr_next": Value Change
Type
Time
CtlRef
OldVal
scr_next
TF
Script
ListBox
scr_next
Boolean
scr_step
Boolean
scr_cont
Boolean
scr_pause
Boolean
scr_stop
Boolean
Current Script
Path
Call Stack
ListBox
scr_reload
Boolean
ScriptListBox
Next
Step
Continue
Pause
Abort
CurrentScript
CallStack
PuEnv
Python Cmd Server
Canvas
Picture
PythonViewDemo.lvproj/My Computer

Performance

LabVIEW stepping through code: ~6.2ms per step

Call of LabVIEW server function via python script: 2 ms per call

Conclusion

- With PythonVIEW, Python scripts can be loaded and executed via the LabVIEW environment.
- The Python scripts can call LabVIEW code using the TCP Command server.

Questions?

electronics & embedded systems

3T B.V.
Institutenweg 1
7521 PH Enschede
The Netherlands

Esp 401
5633 AJ Eindhoven
The Netherlands

T. +31 53 4 33 66 33
F. +31 53 4 33 68 69
E. info@3t.nl
W. www.3t.eu