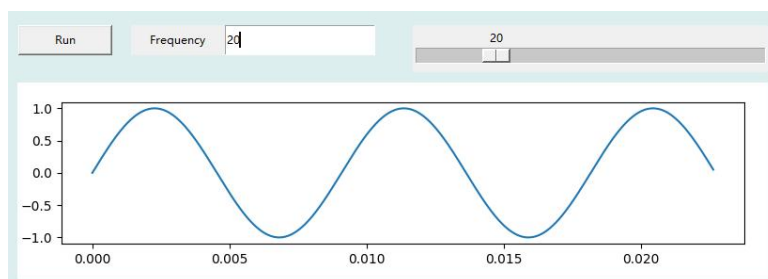


虚拟仪器控件直连和数据流驱动原理

1、直连编程问题的提出

DRVI 的内核是 Python 自带的 GUI 库 Tkinter 或 PyQt 等第三方 GUI 库。Tkinter GUI 控件采用的是回调函数工作机制，用户通过 GUI 界面的键盘或鼠标等操作触发某一 Tkinter 控件后，则调用该控件绑定的自定义回调函数来完成相应功能。程序控制流程很清晰，与一般的带 GUI 界面的 Matlab、C++ 等的编程原理相似。

回调函数工作机制编程简单，但对多控件联动操作时，其编程效率有点低。例如，下面是一个简单的正弦波信号发生器程序，其信号频率即可以用文本框输入，也可以用滑块输入，也就是说数字型文本框和滑块的数值应该是绑定和联动的。



下面是实现上述功能和这两个控件联动的样例代码。我们需要在各自的回调函数中编程调用另一控件的 set 方法来改变使其属性值，使它们的值同步改变。

```
import tkinter as tk
#定义回调函数
def entry1Func(e):
    scale1.set(str(e.widget.get()))

def scalFunc(value):
    entry1.delete(0,"end")
    entry1.insert(0,float(value))

# 创建主窗口（Evidence 10,11）
win = tk.Tk()
win.geometry("1100x740")
# Insert a Entry
entry1_var=tk.StringVar(win,'10')
entry1=tk.Entry(win,textvariable=entry1_var)
entry1.place(x=200,y=20,width=100,height=32)
entry1.bind("<Return>",entry1Func)
# Insert a scale
svar1 = tk.DoubleVar(win,10)
scale1 = tk.Scale(win,variable=svar1,from_=0, to=100, orient=tk.HORIZONTAL,command=scalFunc)
scale1.place(x=320,y=20,width=500,height=50)
# 运行主循环
win.mainloop()
```

如果能够将数字型文本框的输出端绑定到滑块控件的输入端，同时把滑块控件的输出端绑定到数字型文本框的输入端，则这两个回调函数就可以不需要；也就是可以用一个控件的输出数据流去直接驱动和修改另一个控件的属性。这就是我们提出的 DRVI 虚拟仪器控件直连和数据流驱动思想。

下面是 DRVI 控件实现相同功能的代码：

```
import tkinter as tk
import drvi.drviControlls as dr
#==Main Window=====
win= tk.Tk()
win.geometry('1204x716')
```

```

#==Layout of controls=====
mEntry1=dr.DREntry(win,180,114,100,30,'#fffff','#000000','100',12)
mHScale2=dr.DRHSale(win,359,101,200,50,'#002266','#000000',0,100,50,11)
mEntry1.addCallBackSingle(mHScale2.setValueSingle)
mHScale2.addCallBackSingle(mEntry1.setValueSingle)
#==Main Loop=====
win.mainloop()

```

由于直接用控件的回调函数对另一控件的属性进行设定，这样就把外部的 2 个回调函数给省掉了，程序更加简明、高效。但省代码并不是我们的主要目的，根本好处是这样处理后的控件可以实现直连编程，也就是可以用控件装配的图形化编程方式来实现 Python 框架程序设计。

为实现控件直连，必须满足一个前提条件，就是上一控件的输出函数参数个数和类型必须和下一控件的输入函数一致。因此，我们按照信号分析类程序的特点，对 Tkinter 和 Matplotlib 基础上生成的 DRV1 虚拟仪器控件间直连传递的参数进行了标准化处理，并分为如下几个常用的类别：

String: 字符串，用于在控件间交换一段文字信息。

Single: 数字，用于在控件间交换一个浮点数字。

2DArray: 2 个数组，用于在控件间交换二个一维的浮点数组，如二维曲线的 x 坐标和 y 坐标数据。

2DXArray: 数字+一维数组，用于在控件间传递一个浮点数字和一个一维数组，如一条二维曲线中的采样间隔和采样信号数组值。在传递等间距信号，如采样信号波形、FFT 频谱等时，可只传递一个一维数组，其数据传输效率比 2DArray 高。

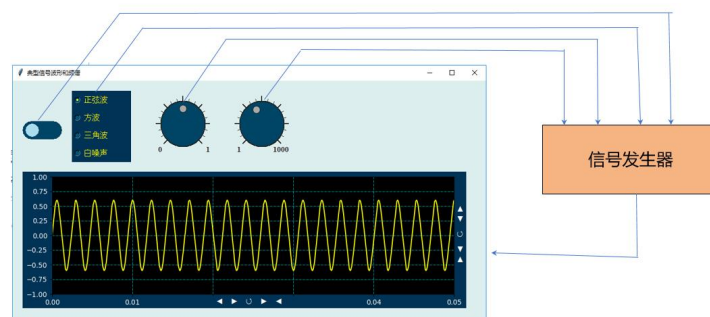
3DArray: 3 个数组，用于在控件间交换三个数组对象。如谱阵图、语谱图数据，这时 x、y 为一维数组，z 通常二维的阵图数据。

Image: 图象，用于在控件间传递一幅图象的数据块。

其它：用户也可以自定义其它组合形式的参数传递组合，只要输出方和输入方的参数个数和类型一致即可实现控件 set 属性函数和 get 属性函数的直连和驱动。

做了参数传递的标准化处理后，只要控件间 get/set 方法的参数个数和类型一致，就可以通过直接将上一控件的输出方法(get)，作为输入传递给下一控件的属性设定方法(set)。这样可以删除因控件取值、赋值所需的回调函数；同时，通过控件的直连和实现数据流驱动，程序框架也变的更易于实现装配式图形化编程。

例如，下面是用 DRV1 虚拟仪器控件设计的一个简单的信号发生器程序。通过一个按钮控件、一个单选按钮组、两个旋钮控件、一个曲线控件和一个数字信号发生器控件间的直连关系绑定，不用编写任何控件以外的回调函数，就可以构成一个数据流驱动的信号发生器应用程序。



下面是用 DRV1 库控件直连实现的信号发生器 Python 样例程序代码。程序内容就是简单的由控件布局 and 控件绑定两部分组成，无需额外的流程控制代码。

```

import tkinter as tk
import drvi.drviDSP as dsp
import drvi.drviControlls as dr
#==创建主窗口=====
win= tk.Tk()
win.geometry('1100x640')
win.config(bg="#ddeeee")
win.wm_title('典型信号波形和频谱')
#==控件插入和布局=====
mSwitch1=dr.DRSwitch(win,20,80,80,36,'#004466','#aaddee')
mVButType=dr.DRRadioBut(win,120,20,120,140,txt='正弦波,方波,三角波,白噪声')
mKnobA=dr.DRKnob(win,280,20,130,130,'#004466','#222222','#aaaaaa','0,1',0,1,0.5,11)
mKnobFre=dr.DRKnob(win,440,20,130,130,'#004466','#222222','#aaaaaa','1,1000',1,1000,50,11)
mSignal=dsp.DRGenerator(0,44100,4096,0.6,100,0)
mPlotWave=dr.DRPlot(win,20,180,900,270,ylow=-1,yhigh=1)
#==控件直连关系绑定=====
mSwitch1.addCallBackSingle(mSignal.setValueSingle)
mSignal.addCallBack2D(mPlotWave.setValue2DFast)
mVButType.addCallBackSingle(mSignal.setVSType)
mKnobA.addCallBackSingle(mSignal.setVSAmp)
mKnobFre.addCallBackSingle(mSignal.setVSFre)
#==Main Loop=====
win.mainloop()

```

绑定 1：实现开关按钮与信号发生器模块定时器开关的绑定，打开开关则信号发生器连续产生信号并输出。

```
mSwitch1.addCallBackSingle(mSignal.setValueSingle)
```

绑定 2：实现信号发生器输出与曲线控件输入的绑定，曲线波形随信号发生器的运行不断刷新变化。

```
mSignal.addCallBack2D(mPlotWave.setValue2DFast)
```

绑定 3：实现单选按钮组与信号发生器的信号类型属性的绑定，信号发生器的信号类型随用户选择改变。

```
mVButType.addCallBackSingle(mSignal.setVSType)
```

绑定 4：实现旋钮 1 与信号发生器的信号幅值属性的绑定，信号发生器输出的信号幅值随旋钮 1 改变。

```
mKnobA.addCallBackSingle(mSignal.setVSAmp)
```

绑定 5：实现旋钮 2 与信号发生器的信号频率属性的绑定，信号发生器输出的信号频率随旋钮 2 改变。

```
mKnobFre.addCallBackSingle(mSignal.setVSFre)
```

控件的绑定关系图也就是程序的运行控制图。因此，基于 DRVI 直连控件库的图形化编程就只需要实现可视化的控件布局和控件间连线，我们称之为程序的装配；最后再将布局中用到的控件和控件间的绑定关系导出为 Python 代码，就可以快速的为用户实现一个 Python 应用程序框架内，这实现起来也似乎不难。

HaHa，一个 ugly 的仿 Matlab AppDesigner 的图形化辅助编程工具实现了。

2、DRVI 库在 Python Class 结构程序设计中的应用

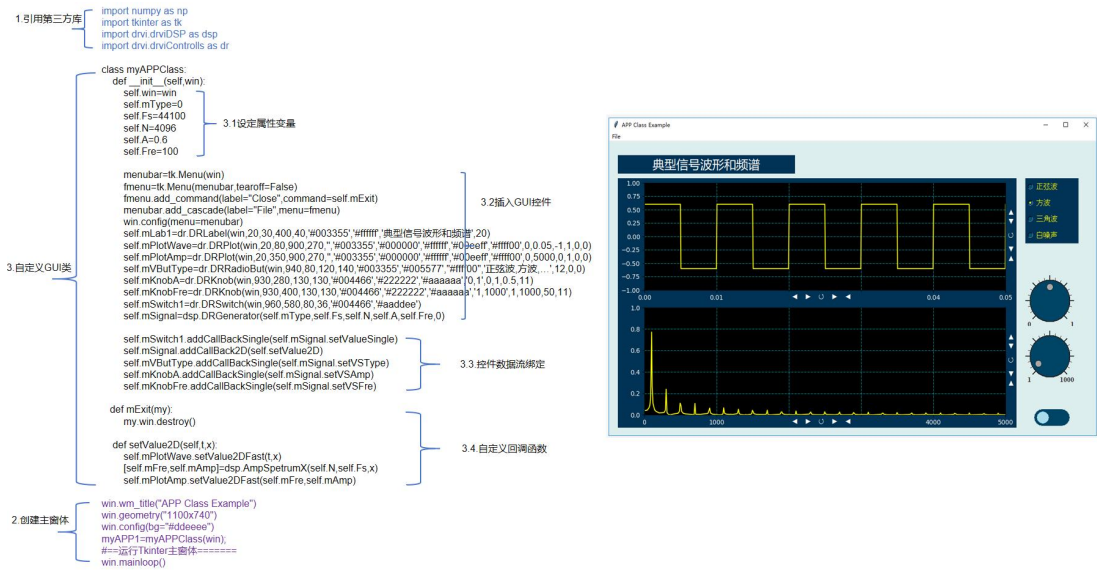
对复杂一些的应用程序，带 Python Class 结构的程序模块化功能会更好一些，同时也更便于实现设计者间的代码共享。毕竟将功能封装为一个类后，只要提供一个接口调用说明，一切就 OK，内部实现可以不用考虑。

Tkinter 支持 Python Class 结构，作为 Tkinter 基础上开发的虚拟仪器风格 GUI 控件包，DRVI 库也就顺势继承了这一特色。将 DRVI 库应用 Python Class 中，在原理实现上并没有区别，只要将程序的主体结构改为 Python Class 就可以了。通常包含如下 5 个设计步骤：

- 1、书写需要引用的 Tkinter 包、DRVI 包和 Numpy 包等程序用到的支持包；
- 2、书写 Tkinter 主窗体程序创建代码；
- 3、用 DRVI 控件布局 and 连线做为主体内容，创建一个 GUI Class；
- 4、在 GUI Class 中定义公共属性变量和完成用户所需功能的自定义回调函数。
- 5、将 GUI Class 加载到 Tkinter 中

通常，1、2、3、5 步，可以用图形化编程方式实现，也就是程序的框架结构可以用图形化编程方式实现；用户只要专注于完成特定功能的自定义回调函数编程即可。不想学 DRPython 图形化编程的话，也可以从安装后的 DRVI 库中找一个相近的样例代码，在其基础上修改实现。这样，用户编程主要集中在第 4 步，也就是自己的特色工作上。

下面是一个完整的由这五部分构成的 Python Class 结构形式的 DRVI GUI 应用程序，其功能就是一个典型信号的波形与频谱分析程序。



下面是 Class 形式书写的典型信号波形与频谱分析示例程序的 Python 代码：

```
import numpy as np
import tkinter as tk
import drvi.drviDSP as dsp
import drvi.drviControls as dr

#====自定义 GUI 类=====
class myAPPClass:
    def __init__(self, win):
        #==用 self 标记定义类中可引用的属性变量
        self.win = win
        self.mType = 0
        self.Fs = 44100
        self.N = 4096
        self.A = 0.6
        self.Fre = 100
```

```

self.Pha=0

# 插入 GUI 控件=====
menubar=tk.Menu(win)
fmenu=tk.Menu(menubar,tearoff=False)
fmenu.add_command(label="Close",command=self.mExit)
menubar.add_cascade(label="File",menu=fmenu)
win.config(menu=menubar)
self.mLab1=dr.DRLabel(win,20,30,400,40,'#003355','#fffff','典型信号波形和频谱',20)
self.mPlotWave=dr.DRPlot(win,20,80,900,270,"'#003355','#000000','#fffff','#00eeff','#ffff00',0,0.05,-1,1,0,0)
self.mPlotAmp=dr.DRPlot(win,20,350,900,270,"'#003355','#000000','#fffff','#00eeff','#ffff00',0,5000,0,1,0,0)
self.mVButType=dr.DRRadioBut(win,940,80,120,140,'#003355','#005577',"#ffff00",'正弦波,方波,三角波,白噪声',12,0,0)
self.mKnobA=dr.DRKnob(win,930,280,130,130,'#004466','#222222','#aaaaaa','0,1',0,1,0.5,11)
self.mKnobFre=dr.DRKnob(win,930,400,130,130,'#004466','#222222','#aaaaaa','1,1000',1,1000,50,11)
self.mSwitch1=dr.DRSwitch(win,960,580,80,36,'#004466','#aaddee')
self.mSignal=dsp.DRGenerator(self.mType,self.Fs,self.N,self.A,self.Fre,self.Pha)
#==控件数据流绑定=====
self.mSwitch1.addCallBackSingle(self.mSignal.setValueSingle)
self.mSignal.addCallBack2D(self.setValue2D)
self.mVButType.addCallBackSingle(self.mSignal.setVSType)
self.mKnobA.addCallBackSingle(self.mSignal.setVSAmp)
self.mKnobFre.addCallBackSingle(self.mSignal.setVSFre)

#==控件回调函数/自定义函数=====
def mExit(my):
    my.win.destroy()

def setValue2D(self,t,x):
    self.mPlotWave.setValue2DFast(t,x)
    [self.mFre,self.mAmp]=dsp.AmpSpectrumX(self.N,self.Fs,x)
    self.mPlotAmp.setValue2DFast(self.mFre,self.mAmp)

#==创建 Tkinter 主窗体=====
win= tk.Tk()
win.wm_title("APP Class Example")
win.geometry("1100x740")
win.config(bg="#ddeeee")
#==创建自定义 GUI 类
myAPP1=myAPPClass(win);
#==运行 Tkinter 主窗体=====
win.mainloop()

```

开发工作完成后，我还将 DRVI 移植到了 pyqt 上，用户可以选择用 Tkinter 做为基础 GUI 库或选择用 PyQt5 做为基础 GUI 库来开发虚拟仪器程序。二者完全相同，只是引用的 GUI 库有所差异。下面是视频文件的音频信号波形和频谱分析程序代码。

```

from PyQt5.QtWidgets import *
from PyQt5.QtGui import *
from PyQt5.QtCore import *
import sys
import numpy as np
import drvi.drviDSP as dsp
import drvi.drviQTControlls as drqt

#===Start of MainClass=====
class drviMain(QMainWindow):
    def __init__(self):
        super(drviMain,self).__init__()
        self.initGUI()

```

```

#==GUI 初始化=====
def initGUI(self):
    self.setWindowTitle("DRVI-MP4 文件波形和频谱 -- By DRPython")
    self.resize(1040,650)
    self.win=QWidget()
    self.win.setStyleSheet('background-color : #ddeeee; ')
    self.setCentralWidget(self.win)

    #添加控件=====
    self.mLab2=drqt.DRLabel(self.win,20,10,300,36,'#005588','#ffffff','MP4 文件波形和频谱',20)
    self.mFile=drqt.DREntry(self.win,600,10,310,36,'#000000','#ffff00','',12)
    self.mPlotWave=drqt.DRPlotFast(self.win,20,60,1000,260,0,0,0.09,-1,1,0,0)
    self.mPlotAmp=drqt.DRPlotFast(self.win,20,310,1000,260,0,0,5000,0,0.25,0,0)
    self.mPlayer=drqt.DRPlay(self.win,920,10,80,36,'#000000','#ffff00',11,'')
    self.mBut=drqt.DRHButtonGroup(self.win,400,10,190,36,'#006655','#ffff00','运行,停止,文件',12)

    #==事件流、数据流绑定=====
    self.mBut.addCallbackSingle(self.mPlayer.setValueSingle)
    self.mPlayer.addCallback2DX(self.setValue2DX)
    self.mPlayer.addCallbackStrFile(self.mFile.setValueString)
    #==初始运行状态=====
    self.TT=0

    #==事件流、数据流响应函数=====
    def setFile(self,v):
        self.mPlayer.setValueSingle(2)

    def setValue2DX(self,dx,xArray):
        Fs=1/dx; N=len(xArray)
        CH=xArray.ndim
        if CH==2:
            xArray=xArray[:,0] #立体声，取一通道数据
        mFre,mAmp=dsp.AmpSpectrum(N,Fs,xArray)
        self.mPlotWave.setValue2DX(dx,xArray)
        self.mPlotAmp.setValue2D(mFre,mAmp)

#=====
app=QApplication(sys.argv)
win=drviMain()
win.show()
sys.exit(app.exec_())

```

程序运行效果如下：

