Uitleg in het Nederlands

Het in opbouw zijnde

Plantengroeivolgsysteem

heeft tot (uiteindelijk) doel de groei vanaf zaadje tot een jonge plant van enkele jaren oud met grote nauwkeurigheid te volgen. Daartoe moeten lucht- en bodemtemperatuur, licht (PAR-straling en totaalstraling), bodemvochtigheid en CO2-gehalte van de lucht precies gemeten worden. Bovendien ook de luchtdruk en eventuele andere variabelen.
Er zijn meerdere "planteneilandjes" opgezet onder verschillende omstandigheden, bijvoorbeeld op verschillende plaatsen in de buitenlucht, in een serre en in het bijzonder in een grote en meerdere kleine reactoren in de serre. Van alle planteneilandjes kan in ieder geval de buitentemperatuur met een thermokoppel K-type gemeten worden, met als ijkthermometers een zestal zeer nauwkeurige Pt-100 4-draads sensoren.

Aan een dergelijk systeem worden extreem hoge eisen gesteld wat betreft het innemen en verwerken van de resultaten. Bij bijvoorbeeld elektriciteitsuitval moet geprobeerd worden op noodstroom over te schakelen totdat de uitval voorbij is. Alle apparaten moeten zo zijn geprogrammeerd dat ze dan direct verder kunnen doorgaan. Toch gemiste meetwaarden moeten zo nodig via een schatting automatisch worden aangevuld.

Het gebruikte Data Acquisition System is de RIGOL M300 mainframe, met een zes-en-een-halve digit Digitale Voltmeter en vier inschuifkaarten met (vooral) een groot aantal ingangen die onafhankelijk van elkaar kunnen worden geprogrammeerd. De thermokoppels en Pt-100 sensoren kunnen worden aangesloten via BNC-bussen en het "kanaal", de ingang dus, wordt ingesteld op het vooraf ingestelde type. Andere signalen, zoals van drukmeters en CO2-sensoren, kunnen zo ingesteld worden dat het signaal, een lage spanning meestal, worden omgerekend naar de druk en het CO2-gehalte.
De gemeten signalen kunnen met grote snelheid worden ingenomen, de volgorde waarop dit gebeurt staat in de zogenaamde scan list. Deze lijst kan via het toetsenbord van de M300 worden ingegeven of via de computer worden overgebracht.
Per planteneilandje kan een korte scan list worden opgezet die net vóór de meting van de kanalen in het geheugen van de M300 geladen moet zijn. Een dergelijke *measurement configuration* (met de suffix .mfg) behoort dus bij een aantal planten of zaden dat onder gelijke omstandigheden groeit of ontkiemt.

Bij de metingen kan alles misgaan, de M300 heeft dan ook een aantal registers waarin afwijkingen worden opgeslagen. Deze registers kunnen tijdens de metingen of achteraf worden geraadpleegd.
Bijvoorbeeld: indien een kanaal verbonden is met één van de vier Alarm-registers, zal het Alarm-register kunnen worden uitgelezen met het commando SYS:ALARx?

Plant growth following system

M300 Data Acquisition Mainframe
Flexible, modular Data Acquisition system with high performance scanning 6.5 digit DMM.
5 slot mainframes and modules for customized data acquisition solutions including temperature, resistance, current and voltage measurements.

Standard Commands for Programmable Instrumentation (SCPI)
This defined set of commands for controlling instruments uses ASCII characters, providing some basic standardization and consistency to the commands used to control instruments. For example, when you want to measure a DC voltage, the standard SCPI command is “MEASURE:VOLTAGE:DC?”.
en
Interchangeable Virtual Instruments (IVI) Foundation
IviDmmMeasurement.Read
en
Virtual Instrument Software Architecture (VISA) I/O library
en
IVI Shared Components

VISA address string

Scan list/Channel list/Channel
The scan list parameter can be one or more channels. For example, in the CONFigure:CURRent:AC[{<range>|AUTO|MIN|MAX|DEF}[,{<resolution>|MIN|MAX|DEF}],](@<scan_list>) command, the parameter (@<scan_list>) can be (@301:302,215) (representing channel 01 through 02 on the module in Slot3 and channel 15 on the module in Slot2), (@201) (representing channel 01 on the module in Slot2) or (@101:112) (representing channel 01 through 12 on the module in Slot1). This parameter will reset the current scan list.
The channel list parameter can be one or more channels. For example, in the [SENSe:]VOLTage[:DC]:NPLC {<PLCs>|MIN|MAX}[,(@<ch_list>)] command, the parameter (@<ch_list>) can be (@301:302,215) (representing channel 01 through 02 on the module in Slot3 and channel 15 on the module in Slot2), (@201) (representing channel 01 on the module in Slot2) or (@101:112) (representing channel 01 through 12 on the module in Slot1). The current scan list will not be affected by this parameter. The channel parameter can only be a single channel. For example, in the CONFigure:COPY:CH:SLOT (@<channel>),<slot> command, the parameter <channel> can be (@213) (representing channel 13 on the module in Slot2). The current scan list will not be affected by this parameter.

See Resources (ivifoundation.org)

The IVI Foundation has created IVI class

Voorbereiding van de M300 en de computer
Uit de verschillende mogelijkheden wordt een keuze gemaakt die gedurende de daarop volgende metingen gehandhaafd moet blijven. Het betreft dan de taalkeuze, punt of komma, datum. Wat betreft de datum: op de aansturende computer wordt GMT gebruikt, zonder omschakelen naar zomertijd.

Voorbereiding van de meetkanalen
De meetkanalen moeten voorbereid zijn of worden op hun beoogde functie. Bovendien kunnen enkele kanalen uitgebreid worden met de alarm-functie. De kanalen kunnen van te voren ingesteld zijn met de hand en daarna opgeslagen worden in een *measurement configuration file* die daarna aangeroepen kan worden. Bij het aanroepen van een volgende .mfg-file worden de bestaande gegevens gewist.

Tijdverloop van de metingen
Door de lange duur van de metingen moet het aantal metingen per kanaal, meetpunt, beperkt blijven, bijvoorbeeld 100 per dag. Als we 10 planteneilandjes hebben en naast temperatuur ook andere variabelen meten dan loopt het aantal meetwaarden zeer snel op, zeker bij meettijden van 100 dagen of langer.

Ook moet zo'n meetwaarde uitgebreid worden met datum en tijd en het kanaalnummer, zodat alle meetwaarden per meetpunt achteraf chronologisch gesorteerd kunnen worden.
Datum en tijd kunnen in twee versies worden toegevoegd, evenals het de te meten eenheid, kanaalnummer en eventueel het type alarm. De RELative versie lijkt aantrekkelijk omdat vanaf de starttijd de tot dan toe verlopen tijd in s wordt geregistreerd, maar bij elk nieuw experiment wordt de zaak terug gezet op 0.
De ABSolute versie geeft de datum en tijd in absolute waarde weer. Deze moet achteraf terug geconverteerd worden voor de verlopen tijd sinds de allereerste meting.
Tevens kan dus de eenheid van de meetwaarde worden meegestuurd, dus C voor temperatuur, V voor spanning, etc.

De status-registers
Het **Alarm register** bevat het *condition register*, *enable register* (*PSC), *event register* (*CLS)
Het **Operation Status register** het *condition register*, *enable register* (*PSC)
Het **Questionable Status register** het *condition register*, *enable register* (*PSC), *event register* (*CLS)
Het **Standard Event register** het *event register* (*ESR?), *enable register* (*ESE; *PSC)

*CLS clear the event registers, error queues and alarm queues
*OPC set bit 0 of the enable register of the Standard Event register to 1 at the end of the current scan
*OPC? queries whether the current operation is completed
*PSC clearing of the enable register

device id
usb0::6833::3200::MM3A201000013::0::INSTR
USB0::0x1AB1:0x0C80::MM3A201000013::INSTR

*IDN?
SYST:EDIT?
SYST:TYPE?
SYST:SERI?
SYST:DATE 2021,6,15
SYST:TIME 9,19,00.000
SYST:OPEN?
SYST:VERS?

SYST:UTIL:FORM:DECI DOT
SYST:UTIL:DISP:BRIG 8
SYST:UTIL:FORM:SEPA Space
SYST:UTIL:LANG EN
SYST:UTIL:BEEP:STAT OFF
SYST:UTIL:POWE:SWIT:STAT ON
SYST:UTIL:SAVE:STAT ON
SYST:UTIL:SAVE:TIME 10
SYS:IDN:USER:DEF Plantengroeivolgsysteem

TOT:CLE:IMM
ROUT:MON:STAT OFF

planteneilandje en hun *measurement configuration file*:
"Pruim.mfg"
"Peelb.mfg"
"Kweek.mfg"
"React.mfg"
"Kiem.mfg"
"Huish.mfg"
"Licht.mfg"
"CO2.mfg"
"Trig.mfg"

0_STATE0.mfg *SAV/*RCL 0
1_STATE1.mfg *SAV/*RCL 1
2_STATE2.mfg *SAV/*RCL 2
3_STATE3.mfg *SAV/*RCL 3
4_STATE4.mfg *SAV/*RCL 4
5_STATE5.mfg *SAV/*RCL 5

TRIG:COUN 100
TRIG:SOUR TIM
TRIG:TIM 864
dus 100 metingen, 864 s na elkaar, overbruggen 24.00 uur.

OUTP:ALARx:MODE TRAC

FORM:READ:TIME:TYPE ABS
FORM:READ:CHAN ON
FORM:READ:TIME ON
FORM:READ:UNIT ON
FORM:READ:ALAR ON
we krijgen dan 1,23...V op 15 juni 2021 om 8:31 uur, kanaal 102 en geen alarm:
+1.23456789E-00   V,2021,06,15,08,31,12.345,102,0
SYST:TIME:SCAN?
starttijd en datum van de laatste scan.

UNIT:TEMP C
UNIT:ANYS string

aan het einde van een meetsessie kunnen de meetgegevens uit het geheugen worden verwijderd.
OUTP:ALARx:CLE

ABORt
CALCulate:AVERage:AVERage?
CALCulate:AVERage:MAXimum?
CALCulate:AVERage:MINimum?
CALCulate:AVERage:PTPeak?
CALCulate:AVERage:SDEV?
CALCulate:AVERage:CLEar
CALCulate:AVERage:COUNt?
CALCulate:AVERage:MAXimum:TIME?
CALCulate:AVERage:MINimum:TIME?
CALCulate:COMPare:DATA
CALCulate:COMPare:MASK
CALCulate:COMPare:STATe
CALCulate:COMPare:TYPE
CALCulate:LIMit:LOWer
CALCulate:LIMit:UPPer
CALCulate:LIMit:LOWer:STATe
CALCulate:LIMit:UPPer:STATe
CALCulate:SCALe:SQUare
CALCulate:SCALe:GAIN
CALCulate:SCALe:OFFSet
CALCulate:SCALe:CONStant
CALCulate:SCALe:OFFSet:NULL
CALCulate:SCALe:STATe
CALCulate:SCALe:UNIT
CONFigure?
CONFigure:ANYSensor
CONFigure:COPY:CH:CH
CONFigure:COPY:CH:SLOT
CONFigure:COPY:SLOT:SLOT
CONFigure:CURRent:AC
CONFigure:CURRent[:DC]
CONFigure:DIGital:BYTE
CONFigure:DIGital:DWORd
CONFigure:DIGital:WORD
CONFigure:FREQuency
CONFigure:PERiod
CONFigure:FRESistance
CONFigure:RESistance
CONFigure:TEMPerature
CONFigure:TOTalize
CONFigure:VOLTage:AC
CONFigure:VOLTage[:DC]
DATA:LAST?
DATA:POINts?
DATA:POINts:EVENt:THReshold
DATA:REMove?
DIAGnostic:DMM:CYCLes?
DIAGnostic:DMM:CYCLes:CLEar
DIAGnostic:PEEK:SLOT:DATA
DIAGnostic:POKE:SLOT:DATA
DIAGnostic:RELay:CYCLes?
DIAGnostic:RELay:CYCLes:CLEar
DISPlay
DISPlay:TEXT
DISPlay:TEXT:CLEar
FETCh?
FORMat:READing:ALARm
FORMat:READing:CHANnel
FORMat:READing:TIME
FORMat:READing:TIME:TYPE
FORMat:READing:UNIT
*CLS
*ESE
*ESR?
*IDN?
*OPC
*PSC
*RST

1 Paul Theelen, Monarchstraat 19, 5641 GH Eindhoven 040-2814621 l.theelen@on.nl

Dit commando SYS:ALARx is één van de zeer vele SCPI's, Standard Commands for Programmable Instruments, waarmee de M300 aangestuurd kan worden of signalen uitgelezen.

Plantengroei is extreem traag vergeleken met de meeste andere biologische processen. Om een betrouwbaar oordeel te kunnen vormen, moeten we dus over vele dagen aaneensluitend metingen verrichten. Het aantal metingen per dag kan echter beperkt blijven. Daarvoor wordt de dag van 86400 s verdeeld in bijvoorbeeld 100 of 1000 gelijke delen van dus 864 en 86,4 s. De meting starten dus op een triggersignaal; omdat de meting, ook van tientallen kanalen, maar enkele s duren, kan de rest van de tijd het meetinstrument voor andere zaken gebruikt worden.

specifications that define the capabilities for drivers for the following thirteen instrument classes:

Digital multimeter (DMM)	IviDmm
Oscilloscope	IviScope
Arbitrary waveform generator	IviFgen
DC power supply	IviDCPwr
AC power supply	IviACPwr
Switch	IviSwtch
Power meter	IviPwrMeter
Spectrum analyzer	IviSpecAn
RF signal generator	IviRFSigGen
Upconverter	IviUpconverter
Downconverter	IviDownconverter
Digitizer	IviDigitizer
Counter/timer	IviCounter

Write the program. The programs in this series all perform the following steps:
• Initialize the instrument – Initialize is required when using any IVI driver. Initialize establishes a communication link with the instrument and must be called before the program can do anything with the instrument. The examples set reset to true, ID query to false, and simulate to true.
Setting reset to true tells the driver to initially reset the instrument. Setting the ID query to false prevents the driver from verifying that the connected instrument is the one the driver was written for. Finally, setting simulate to true tells the driver that it should not attempt to connect to a physical instrument, but use a simulation of the instrument.
• Configure the instrument – The examples set a range of 1.5 volts and a resolution of 0.001 volts (1 millivolt).
• Access an instrument property – The examples set the trigger delay to 0.01 seconds.
• Set the reading timeout – The examples set the reading timeout to 1000 milliseconds (1 second).
• Take a reading
• Close the instrument – This step is required when using any IVI driver, unless the ADE explicitly does not require it. We close the session to free resources.
Important! Close may be the most commonly missed step when using an IVI driver. Failing to do this could mean that system resources are not freed up and your program may behave unexpectedly on subsequent executions.
• Check the driver for any errors.
• Display the reading.

Using IVI-C with Python
The Environment
Python is an open source programming language developed by the Python Software Foundation. It is interpreted and fully object-oriented with a focus on readability and efficient development. Python is used across a wide variety of applications and features a robust set of third-party modules. This chapter provides detailed instructions on how to call an IVI-C specific driver using Python.

Defining VISA Types as ctypes

*SRE set the enable register of the **Status Byte register** (*PSC)
*STB? query the condition of the **Status Byte register**

Opslag op een externe USB-storage device
De scan list configuratie, systeem configuratie en de meetresultaten kunnen ook worden opgeslagen op een USB-stick die geformateerd moet zijn in FAT of FAT32.
De instructies beginnende met MMEM: maken het bovenstaande mogelijk. Met MMEM:LOG <state> is de opslag aan en uit te schakelen.
Het instrument kan geconfigureerd worden met de .blcfg file. Dit gaat met het commando MMEM:IMP:CONF "<configuration file>".

OUTP:ALAR:CLE:ALL

Meetkanalen

101 referentietemperatuur buiten
102 temperatuur buiten
103 temperatuur bodem bioreactor
104 temperatuur bioreactor uitwendig
105 temperatuur bioreactor inwendig
106 temperatuur bij Peelbak
107
108
109 PAR bioreactor uitwendig
110
111 bij 101
112 bij 102
113 bij 103
114 bij 104
115 bij 105
116 bij 106
117
118
119
120 temperatuur inwendige 19'-kast

201
202
203
204
205
206
207
208
209 aansturing HP current source
210
211
212

301 0.1Ω stroommeting
302
303
304 CO2 binnen bioreactor
305 CO2 buiten bioreactor
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324

401 temperatuur bodem bioreactor
402 temperatuur voorontkiemingsbox
403
404 temperatuur grote Peelbak
405 temperatuur Peelbakje -1-
406 temperatuur Peelbakje -2-
407 temperatuur Peelbakje -3-
408 temperatuur Peelbakje -4-
409
410
411
412
413
414

*SAV
*RCL
*SRE
*STB?
*TRG
*WAI
INITiate
INPut:IMPedance:AUTO
~~INSTrument:DMM~~
~~INSTrument:DMM:INSTalled?~~
~~LXI:IDENtify[:STATE]~~
~~LXI:RESet~~
~~LXI:RESTart~~
MEASure:ANYSensor?
MEASure:CURR:AC?
MEASure:CURR[:DC]?
MEASure:DIGital:BYTE?
MEASure:DIGital:DWORd?
MEASure:DIGital:WORD?
MEASure:FREQuency?
MEASure:PERiod?
MEASure:FRESistance?
MEASure:RESistance?
MEASure:TEMPerature?
MEASure:TOTalize?
MEASure:VOLTage:AC?
MEASure:VOLTage[:DC]?
MEMory:NSTates?
MEMory:SAVE:SYSTem
MEMory:NAME:SYSTem?
MEMory:RECall:SYSTem
MEMory:SAVE:CONFig
MEMory:NAME:CONFig?
MEMory:RECall:CONFig
MEMory:SAVE:MIRRor
MEMory:NAME:MIRRor?
MEMory:RECall:MIRRor
MEMory:SAVE:DATA
MEMory:NAME:DATA?
MEMory:RECall:DATA
MEMory:STATe:DELete
MEMory:STATe:NAME
MEMory:STATe:RECall
MEMory:STATe:VALid?
MMEMory:EXPort?
MMEMory:FORMat:READing:CSEParator
MMEMory:FORMat:READing:RLIMit
MMEMory:IMPort:CATalog?
MMEMory:IMPort:CONFig?
MMEMory:LOG[:ENABle]
OUTPut:ALARm<n>:CLEar
OUTPut:ALARm:CLEar:ALL
OUTPut:ALARm<n>:ENABle?
OUTPut:ALARm[<n>]:MODE
OUTPut:ALARm[<n>]:SLOPe
OUTPut:ALARm<n>:SOURceR?
READ?
~~ROUTe:CHANnel:ADVance:SOURce~~
~~ROUTe:CHANnel:ADVance:EDGE~~
ROUTe:CHANnel:DELay
ROUTe:CHANnel:DELay:AUTO
ROUTe:CHANnel:FWIRe
~~ROUTe:CLOSe~~
ROUTe:CLOSe:EXCLusive
ROUTe:DONE?
ROUTe:MONitor[:CHAN]
ROUTe:MONitor:DATA?
ROUTe:MONitor:DATA:FULL?
ROUTe:MONitor:STATe
ROUTe:OPEn

```
To help with translating function calls from
the tktds1k2k driver, you can first create
ctype aliases for commonly used VISA types.
1. Create a new Python source file named
visatype.py.
2. Import the ctypes library.
3. Declare type aliases for primitive types
defined in visatype.h in equivalent ctypes.
# -*- coding: utf-8 -*-
import ctypes
ViChar=ctypes.c_char
ViInt8=ctypes.c_int8
ViInt16=ctypes.c_int16
ViUInt16=ctypes.c_uint16
ViInt32=ctypes.c_int32
ViUInt32=ctypes.c_uint32
ViInt64=ctypes.c_int64
ViString=ctypes.c_char_p
ViReal32=ctypes.c_float
ViReal64=ctypes.c_double
# Types that are based on other visatypes
ViBoolean=ViUInt16
VI_TRUE=ViBoolean(True)
VI_FALSE=ViBoolean(False)
ViStatus=ViInt32
ViSession=ViUInt32
ViAttr=ViUInt32
ViConstString=ViString
ViRsrc=ViString

Load the tktds1k2k DLL into Memory
To load the tktds1k2k DLL into memory for use
in Python, first import ctypes and the
visatype library that you recently created.
import ctypes
from visatype import *
Now you can call the ctypes.cdll.LoadLibrary
method with a path to the location on disk
where the IVI driver C DLL is installed. This
example is designed for use with 64-bit Python
so it uses the path to the 64-bit driver C
DLL.
#load tktds1k2k DLL into memory
tkDLL=ctypes.cdll.LoadLibrary(r'C:\Program
Files\IVIFoundation\IVI\Bin\tktds1k2k_64.dll')
Python stores a reference to the C DLL in the
specified variable, tkDLL. The path string
literal is prefixed with ‘r’ to indicate this
is a raw string, treating the backslashes as
literal characters. You are now able to call
functions from the tktds1k2k driver as methods
on the reference to the driver DLL.

Initialize the Instrument
Before calling the initialization and
configuration methods of the driver, begin
by declaring the variables you will need to
pass into the function calls such as the
session handle, the resource name, the option
string, and any other pieces of data relevant
to your application.
1. Create a session variable and set it to an
instance of a ViSession object.
session=ViSession()
2. Define the resource name, option string,
and channel string. To allocate string
variables for passing into functions, call
ctypes the method create_string_buffer(). If
using string literals, you need to set the
OS encoding as well.
```

```
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464 accuspanning
```

```
ROUTe:SCAN
ROUTe:SCAN:SIZE?
[SENSe:]ANYSensor:FREQuency:RANGe:LOWer
[SENSe:]ANYSensor:FRESistance:APERture
[SENSe:]ANYSensor:RESistance:APERture
[SENSe:]ANYSensor:FRESistance:NPLC
[SENSe:]ANYSensor:RESistance:NPLC
[SENSe:]ANYSensor:FRESistance:OCOMpensated
[SENSe:]ANYSensor:RESistance:OCOMpensated
[SENSe:]ANYSensor:VOLTage:APERture
[SENSe:]ANYSensor:VOLTage:NPLC
[SENSe:]ANYSensor:CURRent:APERture
[SENSe:]ANYSensor:CURRent:NPLC
[SENSe:]ANYSensor:SEGMent
[SENSe:]ANYSensor:SEGMent:CLEar
[SENSe:]ANYSensor:TYPE
[SENSe:]CURRent:AC:BANDwidth
[SENSe:]CURRent:AC:RANGe
[SENSe:]CURRent[:DC]:RANGe
[SENSe:]CURRent:AC:RANGe:AUTO
[SENSe:]CURRent[:DC]:RANGe:AUTO
[SENSe:]CURRent:AC:RESolution
[SENSe:]CURRent[:DC]:APERture
[SENSe:]CURRent[:DC]:NPLC
[SENSe:]CURRent[:DC]:RESolution
[SENSe:]DIGital:DATA[:BYTE]?
[SENSe:]DIGital:DATA:WORD?
[SENSe:]DIGital:DATA:DWORd?
[SENSe:]DIGital:TYPE
[SENSe:]DIGital:LEVel
[SENSe:]DIGital:THReshold
[SENSe:]FREQuency:APERture
[SENSe:]PERiod:APERture
[SENSe:]FREQuency:RANGe:LOWer
[SENSe:]PERiod:RANGe:LOWer
[SENSe:]FREQuency:VOLTage:RANGe
[SENSe:]PERiod:VOLTage:RANGe
[SENSe:]FREQuency:VOLTage:RANGe:AUTO
[SENSe:]PERiod:VOLTage:RANGe:AUTO
[SENSe:]FRESistance:APERture
[SENSe:]RESistance:APERture
[SENSe:]FRESistance:NPLC
[SENSe:]RESistance:NPLC
[SENSe:]FRESistance:OCOMpensated
[SENSe:]RESistance:OCOMpensated
[SENSe:]FRESistance:RANGe
[SENSe:]RESistance:RANGe
[SENSe:]FRESistance:RANGe:AUTO
[SENSe:]RESistance:RANGe:AUTO
[SENSe:]FRESistance:RESolution
[SENSe:]RESistance:RESolution
[SENSe:]FUNCtion
[SENSe:]TEMPerature:APERture
[SENSe:]TEMPerature:NPLC
[SENSe:]TEMPerature:RJUNction?
[SENSe:]TEMP:TRANsducer:FRTD:OCOMpensated
[SENSe:]TEMP:TRANsducer:RTD:OCOMpensated
[SENSe:]TEMPerature:TRANsducer:FRTD:RESistance
[:REFerence]
[SENSe:]TEMPerature:TRANsducer:RTD:RESistance[
:REFerence]
[SENSe:]TEMPerature:TRANsducer:FRTD:TYPE
[SENSe:]TEMPerature:TRANsducer:RTD:TYPE
[SENSe:]TEMPerature:TRANsducer:TCouple:CHECk
[SENSe:]TEMPerature:TRANsducer:TCouple:RJUNcti
on
[SENSe:]TEMPerature:TRANsducer:TCouple:RJUNcti
on:TYPE
[SENSe:]TEMPerature:TRANsducer:TCouple:TYPE
[SENSe:]TEMPerature:TRANsducer:THERmistor:TYPE
```

```
resourceName=ctypes.create_string_buffer('1001
C'.encode('windows-1251'))
optionString=ctypes.create_string_buffer('Simu
late=1,RangeCheck=1,QueryInstrStatus=0,Cache=1
'.encode('windows-1251'))
channel=ctypes.create_string_buffer('CH1'.enco
de('windows-1251'))
3. Make a call to the initialization function
in the tktds1k2k DLL. Because the session is
returned as an output parameter from
tktds1k2k_InitWithOptions, you need to pass a
pointer to the session variable into the
method. ctypes provides the method
pointer() to pass variables to C DLLs as
pointers.
tkDLL.tktds1k2k_InitWithOptions(resourceName,
VI_TRUE,VI_TRUE,optionString,ctypes.pointer(se
ssion))

Configure the Instrument
Now that the instrument is initialized, the
configuration methods can be called
utilizing the session variable as a reference
to the current instrument session.
1. For ease of use and readability, define the
constants needed for the configuration
functions being used. Set the constants to
their matching values in the tktds1k2k header
file, wrapped in constructors of the type
aliases defined in visatype.py.
# constants to be used by tktds1k2k driver
TKTDS1K2K_VAL_NORMAL=ViInt32(0)
TKTDS1K2K_VAL_DC=ViInt32(1)
TKTDS1K2K_VAL_EDGE_TRIGGER=ViInt32(1)
TKTDS1K2K_VAL_EDGE_TRIGGER=ViInt32(1)
TKTDS1K2K_VAL_MATH_FFT_CH1=ViInt32(6)
TKTDS1K2K_VAL_POSITIVE=ViInt32(1)
2. Make calls to the necessary configuration
functions. The first parameter is always the
session handle variable. Any additional static
data should be wrapped in constructors for the
matching types in visatype.py to ensure data
is being properly passed to the functions.
tkDLL.tktds1k2k_ConfigureAcquisitionType(sessi
on,TKTDS1K2K_VAL_NORMAL)
tkDLL.tktds1k2k_ConfigureChannel(session,
channel,ViReal64(1.0),ViReal64(0),
TKTDS1K2K_VAL_DC,ViReal64(1.0),VI_TRUE)
tkDLL.tktds1k2k_ConfigureChanCharacteristics(s
ession,channel,ViReal64(1.0e6),iReal64(20.0e6)
)
tkDLL.tktds1k2k_ConfigureAcquisitionRecord(ses
sion,ViReal64(0.01),ViInt32(2500),ViReal64(-
0.005))
tkDLL.tktds1k2k_ConfigureTrigger(session,
TKTDS1K2K_VAL_EDGE_TRIGGER,ViReal64(500e-9))
tkDLL.tktds1k2k_ConfigureMathChannel(session,
TKTDS1K2K_VAL_MATH_FFT_CH1)
tkDLL.tktds1k2k_ConfigureMathFFT(session,ViInt
32(50),ViInt3
2(1),ViInt32(0),ViReal64(1))
tkDLL.tktds1k2k_ConfigureEdgeTriggerSource(ses
sion,channel,ViReal64(0.4),TKTDS1K2K_VAL_POSIT
IVE)
tkDLL.tktds1k2k_ConfigureTriggerCoupling(sessi
on,TKTDS1K2K_VAL_DC)

Acquire the Measurement Waveform
To handle arrays of data using ctypes, you
must first know the size of the array you want
```

```
[SENSe:]TEMPerature:TRANsducer:TYPE
[SENSe:]TOTalize:CLEar:IMMediate
[SENSe:]TOTalize:DATA?
[SENSe:]TOTalize:SLOPe
[SENSe:]TOTalize:STARt[:IMMediate]
[SENSe:]TOTalize:STARt:DEFault
[SENSe:]TOTalize:STOP[:IMMediate]
[SENSe:]TOTalize:STOP:DEFault
[SENSe:]TOTalize:TYPE
[SENSe:]TOTalize:THReshold
[SENSe:]VOLTage:AC:RANGe
[SENSe:]VOLTage[:DC]:RANGe
[SENSe:]VOLTage:AC:RANGe:AUTO
[SENSe:]VOLTage[:DC]:RANGe:AUTO
[SENSe:]VOLTage:AC:BANDwidth
[SENSe:]VOLTage:AC:RESolution
[SENSe:]VOLTage[:DC]:APERture
[SENSe:]VOLTage[:DC]:NPLC
[SENSe:]VOLTage[:DC]:RESolution
[SENSe:]ZERO:AUTO
SOURce:DIGital:DATA[:BYTE]
SOURce:DIGital:DATA:DWORd
SOURce:DIGital:DATA:WORD
SOURce:DIGital:STATe?
SOURce:VOLTage
STATus:ALARm:CONDition?
STATus:ALARm:ENABle
STATus:ALARm[:EVENt]?
STATus:OPERation:CONDition?
STATus:OPERation:ENABle
STATus:OPERation[:EVENt]?
STATus:PRESet
STATus:QUEStionable:CONDition?
STATus:QUEStionable:ENABle
STATus:QUEStionable[:EVENt]?
SYSTem:ALARm?
SYSTem:ANALog:OUTPut:SWITch
SYSTem:COMMunicate:GPIB:ADDRess
SYSTem:COMMunicate:LAN:AUTOip
SYSTem:COMMunicate:LAN:CONTrol?
SYSTem:COMMunicate:LAN:DHCP
SYSTem:COMMunicate:LAN:DNS
SYSTem:COMMunicate:LAN:GATEway
SYSTem:COMMunicate:LAN:IPADdress
SYSTem:COMMunicate:LAN:MAC?
SYSTem:COMMunicate:LAN:MANUip
SYSTem:COMMunicate:LAN:TELNet:PROMpt
SYSTem:COMMunicate:LAN:TELNet:WMESsage
SYSTem:COMMunicate:LAN:SMASk
SYSTem:COMMunicate:LAN:UPDate
SYSTem:COMMunicate:RS232:BAUD
SYSTem:COMMunicate:RS232:FLOWcontrol
SYSTem:COMMunicate:RS232:PARIty
SYSTem:COMMunicate:RS232:PRINt:STATe
SYSTem:CPON
SYSTem:CTYPe:DEFine
SYSTem:CTYPe:DEFault
SYSTem:CTYPe?
SYSTem:DATE
SYSTem:EDITion?
SYSTem:ERRor?
SYSTem:IDN:USER:DEFine
SYSTem:IDN:DEFault
SYSTem:LFRequency?
SYSTem:LOCal
SYSTem:OPENtimes?
SYSTem:PRESet
SYSTem:REMote
SYSTem:RWLock
SYSTem:SECurity[:IMMediate]
```

	to allocate.
	1. Create variables to store the outputs of the waveform read function.
	actualRecordLength=ViInt32()
	actualPts=ViInt32()
	initialX=ViReal64()
	incrementX=ViReal64()
	2. Acquire the actual number of data points from the read function.
	tkDLL.tktds1k2k_ActualRecordLength(session,ctypes.pointer(actualRecordLength))
	3. To create an array for use with ctypes, you must first create a new type
	on the fly by using the base type, the multiplication operator, and the
	size of the array. The constructor can then be called to allocate a ctypes array variable in Python.
	waveform=(ViReal64 * actualRecordLength.value)()
	4. Finally, call the waveform read function, passing in the output variables as pointers.
	tkDLL.tktds1k2k_ReadWaveform(session,channel,actualRecordLength,ViReal64(10000.0),ctypes.pointer(waveform),ctypes.pointer(actualPts),ctypes.pointer(initialX),ctypes.pointer(incrementX))
	
	Display the Acquired Waveform
	This example uses Matplotlib, a popular third-party scientific graphing solution,
	to display the acquired waveform.
	1. Import the pyplot class as an object.
	import matplotlib.pyplot as plt
	2. Configure the plot, pass in the acquired waveform, and display the plot.
	if actualRecordLength.value !=0:
	plt.figure(0).canvas.set_window_title('Acquired Waveform')
	plt.plot(list(waveform))
	plt.show()
	You should see the acquired waveform in a pop-up window:
	
	Add Error Handling
	To add Pythonic error handling, you need to create a function that takes the return value of the tktds1k2k DLL functions and throws an exception in the case of an error. This allows you to use a try/except block to handle errors.
	1. Define a new exception class that extends the Python Exception base class and minimally stores an error code.
	class ErrorCodeException(Exception):
	def __init__(self,error_code):
	self.code=error_code
	2. Now that you have an exception defined, build an error check function that checks the return value of the DLL calls and throws an instance of your exception class when an error occurs. Any non-zero code indicates an error occurred.
	def checkErr(error_code):
	if error_code != 0:
	raise ErrorCodeException(error_code)
	3. Wrap all calls to the tktds1k2k DLL inside of the errCheck function:
	checkErr(tkDLL.tktds1k2k_InitWithOptions(resourceName,VI_TRUE,VI_TRUE,optionString,ctypes.po
				SYSTem:SERIal?
				SYSTem:TIME
				SYSTem:TIME:SCAN?
				SYSTem:TYPE?
				SYSTem:UTIlity:BEEPer:STATe
				~~SYSTem:UTIlity:CARDoperation~~
				SYSTem:UTIlity:CONFigure:POWEron
				SYSTem:UTIlity:DISPlay:BRIGht
				SYSTem:UTIlity:FORMat:DECImal
				SYSTem:UTIlity:FORMat:SEPArate
				SYSTem:UTIlity:LANGuage
				SYSTem:UTIlity:POWEr:SWITch:STATe
				SYSTem:UTIlity:SAVEr:STATe
				SYSTem:UTIlity:SAVEr:TIME
				SYSTem:VERSion
				TRIGger:ABSolute
				TRIGger:COUNt
				TRIGger:EDGE
				TRIGger:SOURce
				TRIGger:TIMer
				UNIT:ANYSensor
				UNIT:TEMPeratur

```
inter(session)))
checkErr(tkDLL.tktds1k2k_ConfigureAcquisitionT
ype(session,TKTDS1K2K_VAL_NORMAL))
...
4. Add the main body of your application, from
instrument initialization to the acquisition
and displaying of data, to a try/except block.
try:
checkErr(tkDLL.tktds1k2k_InitWithOptions(resou
rceName,VI_TRUE,VI_TRUE,optionString,ctypes.po
inter(session)))
checkErr(tkDLL.tktds1k2k_ConfigureAcquisitionT
pe(session,TKTDS1K2K_VAL_NORMAL))
...
plt.plot(list(waveform))
plt.show()
except ErrorCodeException as err:
5. In the except portion, call the function to
get an error message from the error code
stored in the exception and display the
exception to the user.
except ErrorCodeException as err:
error_message=ctypes.create_string_buffer('\00
0'.encode('windows-1251'),256)
tkDLL.tktds1k2k_error_message(session,ViStatus
(err.code),error_message)
print('Error '+str(err.code)+':' +
error_message.value.decode('windows-1251'))

Close the Instrument Session
Now that you've finished with the main body of
the application, call the close method on the
session variable, checking first that the
session is not null.
if session.value != 0:
tkDLL.tktds1k2k_close(session)

Load the IviSope interface from the
TekSeriesScope IVI-COM Driver
GetModule('IviScopeTypeLib.dll')
from comtypes.gen import IviScopeLib

Initialize the Instrument driver
Before calling the initialization and
configuration methods of the driver, lets
create a driver object first.
1. Create a driver object.
ivi_scope=CreateObject('TekSeriesScope.TekSeri
esScope',interface=IviScopeLib.IIviScope)
2. Make a call to the initialization function
now. Pass the VISA resource string and other
required parameters.
ivi_scope.Initialize('USB::0x0699::0x0522::C01
1595::INSTR',False,False,'')

Configure the Instrument
Now that the instrument is initialized, the
configuration methods can be called using the
ivi_scope driver object.
1. Access the driver's Identity interface and
get the instrument model. This shows how to
access any driver property.
scope_model=ivi_scope.Identity.InstrumentModel
2. Access the driver's Acquisition interface
and get the current record length.
rec_len=ivi_scope.Acquisition.RecordLength
3. Configure the Horizontal parameters, if
required.
ivi_scope.Acquisition.ConfigureRecord(TimePerR
ecord=.1,MinNumPts=10000,AcquisitionStartTime=
```

```
0)
4. Access to the Channel interface and
configure its parameters.
ch1=ivi_scope.Channels.Item('Channel1')
ch1.Configure(Range=.1,Offset=0.1,
Coupling=IviScopeLib.IviScopeVerticalCouplingD
C,ProbeAttenuation=1,Enabled=True)

Acquire the Measurement Waveform
To get the waveform data from any channel of
the Oscilloscope, we have to call some of the
IVI Scope class defined functions like
ReadWaveform or FetchWaveform. These functions
are available in IviMeasurement interface of
Measurements repeated capabilities group.
1. Get the access to the respective
Measurement interface for the Channel.
meas1 =ivi_scope.Measurements.Item('Channel1')
2. Call the FetchWaveform function on the
above Measurement interface. FetchWaveform
returns a tuple that contains WaveformArray as
an array of double, InitialX and XIncrement.
waveform=meas1.FetchWaveform()
#print ("Waveform Array:",waveform[0])
# waveform[0] contains the Waveform data,
which we will plot in the next section
# waveform[1] and waveform[2] contains
the InitialX and XIncrement respectively
print ("InitialX :",waveform[1])
print ("XIncrement :",waveform[2])

Display the Acquired Waveform
This example uses Matplotlib, a popular third-
party scientific graphing solution, to display
the acquired waveform.
1. Import the pyplot class as an object.
import matplotlib.pyplot as plt
2. Configure the plot, pass in the acquired
waveform, and display the plot.
plt.figure(0).canvas.set_window_title('Acquire
d Waveform')
plt.plot(waveform[0])
plt.show()
You should see the acquired waveform in a pop-
up window:

Add Error Handling
For the comtypes, the error handling is very
much simplified using the “try/except”
statements.
If the COM method call fails, a COMError
exception is raised, containing the HRESULT
value and the error message details also.
The following example shows, how to handle
errors in your Python code for the IVI COM
driver calls.
from comtypes.client import
CreateObject,GetModule
import comtypes
GetModule('IviScopeTypeLib.dll')
from comtypes.gen import IviScopeLib
ivi_scope=CreateObject('TekSeriesScope.TekSeri
esScope',interface=IviScopeLib.IIviScope)
try:
ivi_scope.Initialize('USB::0x0699::0x0522::C01
1595::INSTR',False,False,'')
except comtypes.COMError as ce:
#Get the Error details as a tuples
com_error=ce.args
#Get the HRESULT value
```

	hresult=com_error[0]
	#Get the Error messages details
	error_messages=com_error[2]
	
	Close the Instrument driver connection
	Now that you've finished with the main body of the application, call the Close method on the driver object.
	ivi_scope.Close()			