PYRAMID
www.pyramid.tech

FX4

FX4 Programmer Manual

Document ID: 2711715845

Version: v3

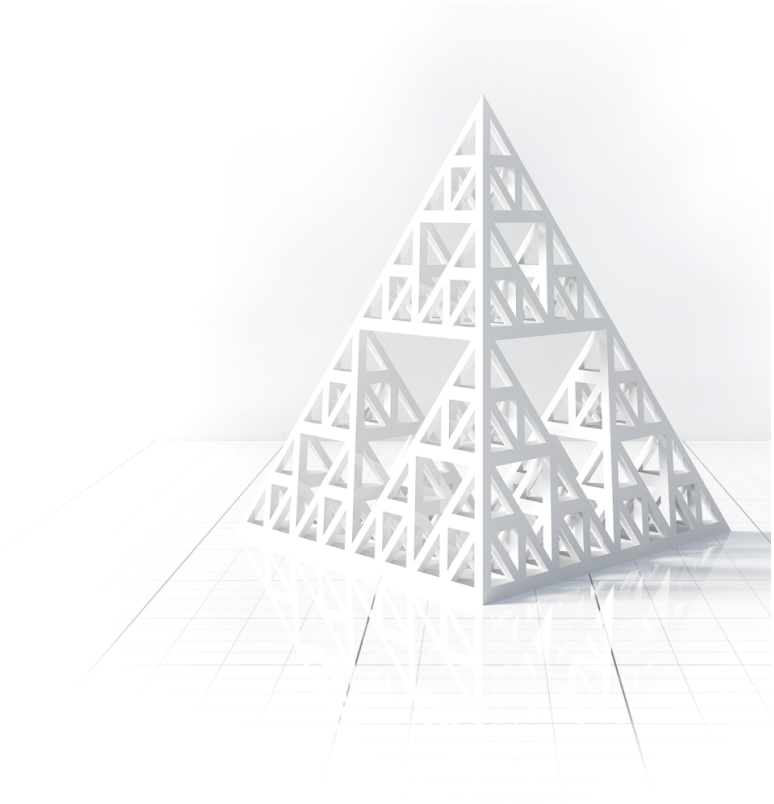


Table of Contents

Document ID: 2711650310

Author	Matthew Nichols
Owner	Project Lead
Purpose	Explain the programming concepts necessary to use the API and extend the product through external applications.
Scope	FX4 related programming concepts.
Intended Audience	Software developers interested in using the product.
Process	https://pyramidtc.atlassian.net/wiki/pages/createpage.action?spaceKey=PQ&title=Standard%20Manual%20Creation%20Process
Training	NOT APPLICABLE

1 Version Control

Version	Description	Saved by	Saved on	Status
v3	Added a simple overview and more examples.	Matthew Nichols	Mar 6, 2025 10:29 PM	APPROVED
v2	Added digital IO interfaces and references back to IGX.	Matthew Nichols	May 3, 2024 7:39 PM	APPROVED
v1	Initial release, still a work in progress.	Matthew Nichols	Feb 21, 2024 11:25 PM	APPROVED

Document Control Not Reviewed

Current document version: v.1

No reviewers assigned.

1.1 Signatures

for most recent document version
Friday, Mar 7, 2025, 10:33 PM UTC
Matthew Nichols signed ; meaning: **Review**

2 References

Document	Document ID	Author	Version
IGX - Programmer Manual	2439249921	Matthew Nichols	1

3 FX4 Programming Overview

The FX4 processor runs on an environment called IGX, which is built on the QNX high-reliability real-time operating system from BlackBerry (QNX Website[1]). IGX provides a flexible and comprehensive application programming interface (API) for users who want to write their own host computer software.

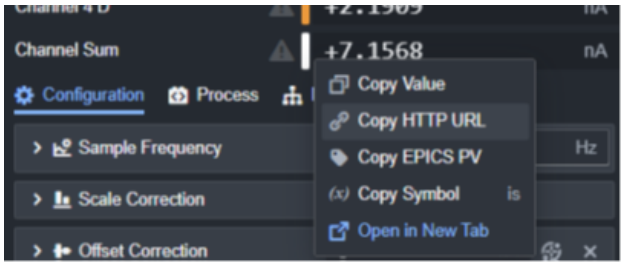
The IGX environment is shared across other Pyramid products, allowing software solutions developed for one product to be easily transferred to others.

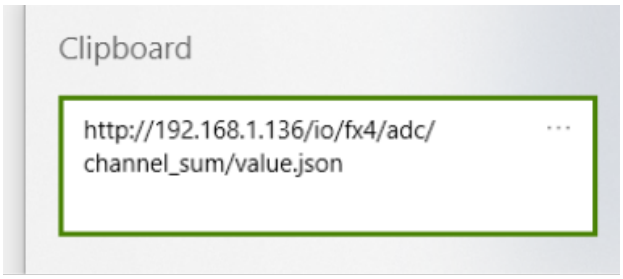
Programmers can refer to the complete documentation for IGX available on the Pyramid website at: IGX | Modern Modular Control System Framework for Web-enabled Applications[2]

This section provides an introduction to testing two of the API methods: HTTP using JSON format and EPICS. For simplicity, Python (Python Website[3]) is used as an example host computer language, which is accessible and easy to use for non-professional programmers.

3.1 Using Python and HTTP

As an example, assume you want to read the sum of the measured currents with Python. You need the URL for that particular IO. The FX4 web GUI provides an easy way to find this: simply right-click in the field and select 'Copy HTTP URL' to copy the string to the clipboard.





Now you can use Python to test connectivity to user software via HTTP and JSON. You may need to import the `requests` and `json` libraries to handle the HTTP requests and data parsing.

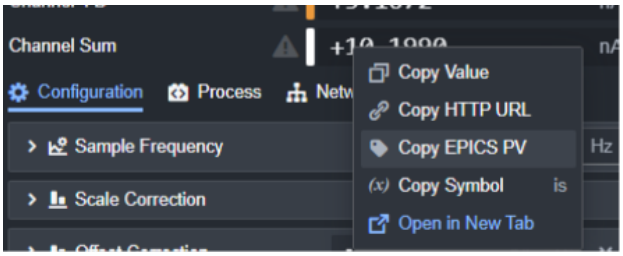
1 Simple Python HTTP Example

3.2 Using EPICS

The process for connecting the FX4 through EPICS (Experimental Physics and Industrial Control System) is similar. EPICS is a set of software tools and applications used to develop and implement distributed control systems, widely used in scientific facilities.

1 https://blackberry.qnx.com/en
2 https://pyramid.tech/products/igx
3 https://www.python.org/

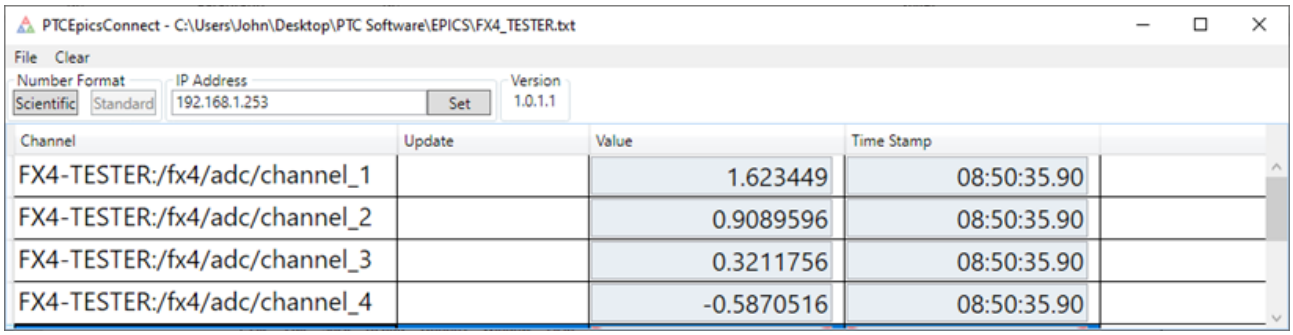
1. Get the **EPICS process variable (PV)** name for the desired IO.
2. Import the EPICS library and read the value.



2 Get EPICS PV Name

3 Simple Python EPICS Example

Additionally, Pyramid created a utility (EPICS Connect[4]) that allows you to monitor EPICS process variables in real-time. This tool is helpful to confirm if the EPICS PV name is correct and the FX4 is serving the PV correctly on your network.



4 PTC EPICS Connect

4 https://ptcusa.com/products/ptcepicsconnect

4 FX4 Programming API

The concepts and methods described in this manual build on the concepts established in the IGX - Programmer Manual. Please see that document for explanation and examples of how basic IGX programming and interfaces work. This manual will only cover the device-specific IO and functionality that is unique to the FX4.

4.1 Analog Input IO

These IO relate to configuring and collecting data on the analog current inputs of the FX4. The units of the channel inputs are based on the user configurable setting called “Sample Units”, valid options include pA, nA, uA, mA, and A.

All 4 channels use the same interface IO and are independently controlled. Replace `channel_x` with `channel_1`, `channel_2`, `channel_3`, or `channel_4` respectively.

IO Path	Description
`/fx4/adc/channel_x`	**READONLY** **NUMBER** Measured current input.
`/fx4/adc/channel_x/scalar`	**NUMBER** Simple unitless scalar applied to the channel, 1 by default.
`/fx4/adc/channel_x/zero_offset`	**NUMBER** Current offset in nA for the channel.

The following IO are not channel independent and are applied to all channels simultaneously.

IO Path	Description
`/fx4/channel_sum`	**READONLY** **NUMBER** Sum of the current input channels.
`/fx4/adc_unit`	**STRING** Sets the current user units for each channel and sum. Options: “pa”, “na”, “ua”, “ma”, “a”
`/fx4/range`	**STRING** Sets the current input range. See GUI for how each range code corresponds to the maximum current input limits and BW. Options: “0”, “1”, “2”, “3”, “4”, “5”, “6”, “7”
`/fx4/adc/sample_frequency`	**NUMBER** The frequency in Hz that sample data will be averaged to. This controls the signal-to-noise and data rate for all channels.
`/fx4/adc/conversion_frequency`	**NUMBER** The frequency in Hz that the ADC will convert analog to digital values at. By default, this is 100kHz, and you will only rarely need to change this value.
`/fx4/adc/offset_correction`	**READONLY** **NUMBER** Sum of all channel's current offsets.

4.2 Analog Output IO

These IO relate to the configuration of the general-purpose analog outputs of the FX4 found under the analog inputs on the front panel. All 4 channels use the same interface IO and are independently controlled. Replace `channel_x` with `channel_1`, `channel_2`, `channel_3`, or `channel_4` respectively.

IO Path	Description
`/fx4/dac/channel_x`	NUMBER Command voltage output. This value can only be written to when output mode is set to manual.
`/fx4/dac/channel_x/readback`	READONLY NUMBER Measured voltage output. This is most helpful when using expression output mode.
`/fx4/dac/channel_x/output_mode`	STRING Sets the output mode for the channel. Options: “manual”, “expression”, “process_control”
`/fx4/dac/channel_x/slew_control_enable`	BOOL Enables or disables slew rate limiting.
`/fx4/dac/channel_x/slew_rate`	NUMBER Slew rate in V/s for the channel.
`/fx4/dac/channel_x/upper_limit`	NUMBER The maximum allowed command voltage for the channel. Applies to all operation modes.
`/fx4/dac/channel_x/lower_limit`	NUMBER The minimum allowed command voltage for the channel. Applies to all operation modes.
`/fx4/dac/channel_x/output_expression`	STRING Sets the expression string used by the channel when it is in the expression output mode.
`/fx4/dac/channel_x/reset_button`	BUTTON Resets the command voltage to 0.

4.3 Digital Input and Outputs

These IO relate to controlling the various general purpose digital inputs and outputs found on the FX4.

IO Path	Description
`/fx4/fr1`	READONLY BOOL Fiber receiver 1.
`/fx4/ft1`	BOOL Fiber transmitter 1.
`/fx4/fr2`	READONLY BOOL Fiber receiver 2.
`/fx4/ft2`	BOOL Fiber transmitter 2.
`/fx4/fr3`	READONLY BOOL Fiber receiver 3.
`/fx4/ft3`	BOOL Fiber transmitter 3.
`/fx4/digital_expansion/d1`	BOOL D1 bidirectional digital expansion IO.
`/fx4/digital_expansion/d2`	BOOL D2 bidirectional digital expansion IO.
`/fx4/digital_expansion/d3`	BOOL D3 bidirectional digital expansion IO.
`/fx4/digital_expansion/d4`	BOOL D4 bidirectional digital expansion IO.

4.3.1 Digital IO Configuration

All digitals have child IO for configuring their behavior including an operating mode which controls how that digital will operate. Each digital will have a different set of available options. See the GUI for details on what options are available for what IO.

Child IO Path	Description
`.../mode`	**STRING** Operation mode for the digital. Options: “input“, “output”, “pwm”, “timer”, “encoder”, “capture”, “uart_rx”, “uart_tx”, “can_rx”, “can_tx”, “pru_input”, or “pru_output”
`.../process_signal`	**STRING** The process control signal name, if there is one.
`.../pull_mode`	**STRING** Pull up/down mode for a digital input. Options: “up“, “down”, or “disable”

4.4 Relay Control

Both relays are independently controlled and share the same type of interface. Replace `relay_x` with `relay_a` or `relay_b` respectively.

IO Path	Description
`/fx4/relay_x/permit/user_command`	**BOOL** Commands the relay open or closed. A true command will try to close the relay if the interlocks are granted, and false command will always open the relay.
`/fx4/relay_x/state`	**READONLY** **STRING** The current state of the relay. Locked relays are open but cannot be closed due to an interlock. States: “opened”, “closed”, or “locked”
`/fx4/relay_x/automatically_close`	**BOOL** When set to true, the relay will automatically close when the interlocks are granted. False by default.
`/fx4/relay_x/cycle_count`	**READONLY** **NUMBER** The number of relay cycles since the last reset. Useful for tracking relay lifetime.

4.5 High Voltage Module

See the IGX - Programmer Manual for details on the FX4 high voltage interface. The component parent path is `/fx4/high_votlage`.

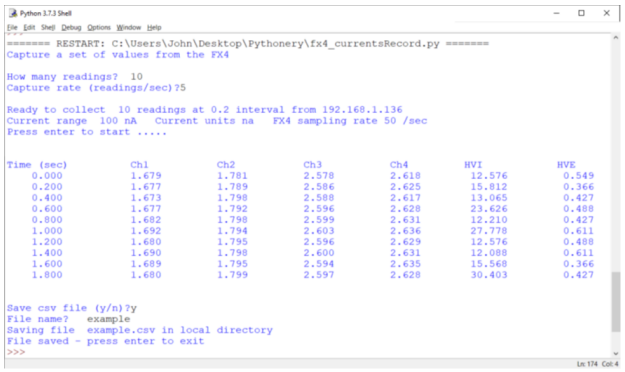
4.6 Dose Controller

See the IGX - Programmer Manual for details on the FX4 dose controller interface. The component parent path is `/fx4/dose_controller`.

5 FX4 Python Examples

5.1 Data Logger using HTTP

This example demonstrates how to capture a number of readings and save them to a CSV file. By choosing a long delay between readings, you can perform long-term data logging even if the FX4 sampling rate is set higher. This allows you to continuously collect and store measurements over extended periods without overwhelming the system, ensuring that data is captured at intervals suitable for your analysis. The delay between readings helps regulate the pace at which data is logged, allowing for efficient storage and reducing the risk of missing data points while still benefiting from high-speed sampling for real-time measurements.



FX4 Data Record						
Tue Jun 25 18:36:02 2024						
Current range	100 nA					
Units	na					
FX4 sampling	50	per sec				
Time	Ch1	Ch2	Ch3	Ch4	HVI	HVE
0	1.678955	1.780889	2.577962	2.618431	12.57631	0.549451
0.2	1.67705	1.788596	2.585997	2.624714	15.81197	0.3663
0.4	1.672937	1.797814	2.58761	2.616807	13.06471	0.42735
0.6	1.676953	1.791853	2.596328	2.628122	23.62637	0.488401
0.8	1.682062	1.797711	2.598939	2.630938	12.21001	0.42735
1	1.692272	1.793881	2.602689	2.636235	27.77778	0.610501
1.2	1.680042	1.795453	2.595704	2.628523	12.57631	0.488401
1.4	1.689656	1.79755	2.60047	2.63107	12.08791	0.610501
1.6	1.689145	1.79451	2.594081	2.634942	15.56777	0.3663
1.8	1.680262	1.799178	2.59707	2.628149	30.40293	0.42735

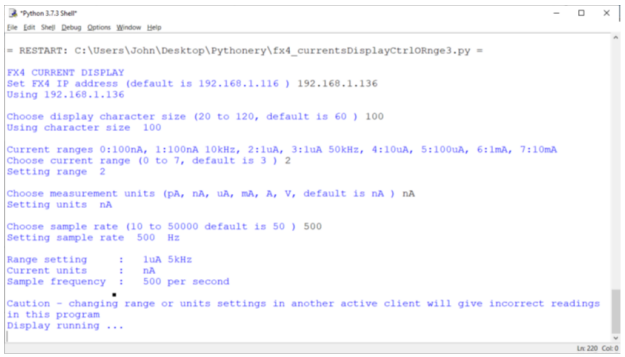
```
# Example Python script to display multiple readings from the FX4
# Write values to console and give option to save to csv

# Import libraries

import requests      # Call to html server
import time          # Timestamps
import csv           # csv file handling
import winsound      # Beep for start and end data collection

# IP address of FX4.  Change as needed.
ipaddr = "192.168.1.136"
fx4_ip = "http://"+ipaddr

# Create a session
sessionFX4 = requests.Session()

# List files for results.  fx4_data is a list of lists.

fx4_data = []
fx4_data_entry = []

# Prompt for number of readings and interval between readings
print ("Capture a set of values from the FX4\n")
num = input("How many readings?  ")
if num=="":
    num=1
else:
    num = eval(num)
rate = input("Capture rate (readings/sec)?")
if rate=="":
    rate=1
else:
```

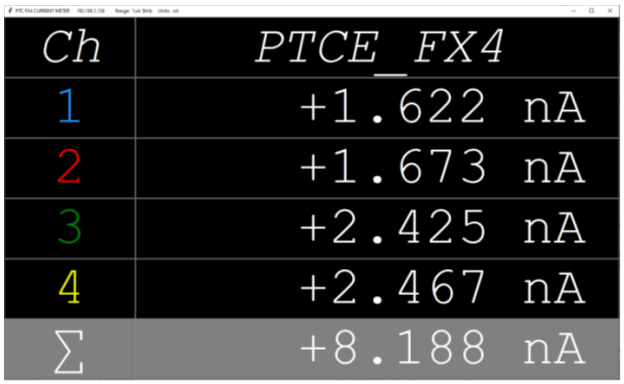
```
    rate = eval(rate)
rate = max(rate,0.1)    # Restrict min rate
rate = min(rate,1000)   # Restrict max rate
interval=1/rate
print()

# Define range settings
i_ranges = ['100 nA','100 nA fast','1 uA','1 uA fast','10 uA','100 uA','1 mA','10 mA']

# Get range setting
fx4_range = (sessionFX4.get(fx4_ip+"/io/fx4/range/value.json")).json()
fx4_range = int(fx4_range[-1])
fx4_range = i_ranges[fx4_range]

# Get units setting
fx4_units = (sessionFX4.get(fx4_ip+"/io/fx4/adc_unit/value.json")).json()
fx4_units = fx4_units[-2:]

# Get sample rate setting (averaging)
fx4_srate = (sessionFX4.get(fx4_ip+"/io/fx4/adc/sample_frequency/value.json")).json()

# Print header lines to console
print ("Ready to collect ", num , "readings at", interval ,"interval from",ipaddr)
print ("Current range ",fx4_range,"  Current units",fx4_units,"  FX4 sampling rate",fx4
_srate,"/sec")
print ("Press enter to start .....","\n")
input()

# Start alert
winsound.Beep(256,500)
winsound.Beep(512,500)
print ("Time (sec)          Ch1           Ch2           Ch3           Ch4
HVI            HVE")

ctr = 0

# Collect values
while num > 0:

#-----------------------------------------------------------------

# Get data from FX4 html server
    vala = (sessionFX4.get(fx4_ip+"/io/fx4/adc/channel_1/value.json")).json()
    valb = (sessionFX4.get(fx4_ip+"/io/fx4/adc/channel_2/value.json")).json()
    valc = (sessionFX4.get(fx4_ip+"/io/fx4/adc/channel_3/value.json")).json()
    vald = (sessionFX4.get(fx4_ip+"/io/fx4/adc/channel_4/value.json")).json()
    vale = (sessionFX4.get(fx4_ip+"/io/fx4/high_voltage/monitor_voltage_internal/
value.json")).json()
    valf = (sessionFX4.get(fx4_ip+"/io/fx4/high_voltage/monitor_voltage_external/
value.json")).json()

#   Increment time counter
    tctr = (ctr)*interval

#   Print to console
    print ("{:9.3f}    {:12.3f}  {:12.3f}  {:12.3f}  {:12.3f}  {:12.3f}  {:12.3f}".form
at(tctr,vala,valb,valc,vald,vale,valf))
```

```
#   Add data to table
    fx4_data_entry = [1, 2, 3, 4, 5, 6, 7]
    fx4_data_entry = [tctr, vala, valb, valc, vald, vale, valf]
    fx4_data [len(fx4_data):] = [fx4_data_entry]

#   Step counters and wait to get next reading before looping
    num = num - 1
    ctr = ctr + 1
    time.sleep(1/rate)

#-----------------------------------------------------------------------

#  Finished alert
winsound.Beep(512,200)
time.sleep(.05)
winsound.Beep(256,500)

#-----------------------------------------------------------------------

#  Prompt for csv file save
time.sleep(1)
print("\n")
response = input("Save csv file (y/n)?")

if (response == 'y'):
    saveName = input("File name?   ")
    if (len(saveName) == 0):
        saveName = "fx4_data"
    saveName = saveName + ".csv"
    saveTime = time.ctime(time.time())
    print("Saving file ", saveName, "in local directory")
    with open(saveName, 'a', newline='') as csvFile:
        writer = csv.writer(csvFile)
        writer.writerow(["FX4 Data Record"])
        writer.writerow([saveTime])
        writer.writerow(["Current range",fx4_range])
        writer.writerow(["Units",fx4_units])
        writer.writerow(["FX4 sampling",fx4_srate,"per sec"])
        writer.writerow(["Time", "Ch1", "Ch2", "Ch3", "Ch4", "HVI", "HVE"])
        writer.writerows(fx4_data)

    csvFile.close()
    input("File saved - press enter to exit")

time.sleep(.5)
```

5.2 Simple Python GUI

The second example uses the **Tkinter** GUI tool, which is built for Python, to create a display of the measured currents. This interface allows you to visualize the current readings in a user-friendly graphical format. The display can be resized to make it large enough to read from across a room, making it ideal for scenarios where real-time monitoring is needed in larger spaces. Tkinter provides an easy way to create interactive interfaces, and by integrating it with the FX4, you can quickly build a visual display of the measured currents that can be customized to fit your specific needs.






```
# Display big FX4 current readings using Tk GUI
# Update reading continuously

import sys
import requests # Call to html server - library has to be downloaded
import json         # For JSON format decoding
import time     # Timestamps
import tkinter      # Graphic GUI
from tkinter import *

# Defaults
dftIPaddr = "192.168.1.116"
dftCharSize = "60"
dftRange = '3'
dftUnits = 'nA'
dftSampleRate = 50 # 50 or 60 according to line frequency

# Initial prompts in console
print ("\nFX4 CURRENT DISPLAY"), print

# Create a session
sessionFX4 = requests.Session()

# Request and check IP address for device
print ("Set FX4 IP address (default is",dftIPaddr,") ",end="")
ipaddr = input()
if ipaddr == "":
    ipaddr = dftIPaddr
try:
    testIP = sessionFX4.get("http://"+ipaddr+"/io/fx4/range/value.json").json()
except:
    print ("No response from device - network issue or invalid IP address")
    print("Exiting")
    time.sleep(2)
    quit()

print ("Using",ipaddr,"\n")

# Set up urls for required parameters
fx4HostName_url = "http://"+ipaddr+"/io/net/hostname/value.json"
fx4range_url = "http://"+ipaddr+"/io/fx4/range/value.json"
fx4units_url = "http://"+ipaddr+"/io/fx4/adc_unit/value.json"
fx4sampleRate_url = "http://"+ipaddr+"/io/fx4/adc/sample_frequency/value.json"
fx4ch1_url = "http://"+ipaddr+"/io/fx4/adc/channel_1/value.json"
fx4ch2_url = "http://"+ipaddr+"/io/fx4/adc/channel_2/value.json"
```

```
fx4ch3_url = "http://"+ipaddr+"/io/fx4/adc/channel_3/value.json"
fx4ch4_url = "http://"+ipaddr+"/io/fx4/adc/channel_4/value.json"
fx4ch1sc_url = "http://"+ipaddr+"/io/fx4/adc/channel_1/scalar/value.json"
fx4ch2sc_url = "http://"+ipaddr+"/io/fx4/adc/channel_2/scalar/value.json"
fx4ch3sc_url = "http://"+ipaddr+"/io/fx4/adc/channel_3/scalar/value.json"
fx4ch4sc_url = "http://"+ipaddr+"/io/fx4/adc/channel_4/scalar/value.json"

# Get host name
hostName = (sessionFX4.get(fx4HostName_url)).json()

#Get scaling values
fx4ch1sc = (sessionFX4.get(fx4ch1sc_url)).json()
fx4ch2sc = (sessionFX4.get(fx4ch2sc_url)).json()
fx4ch3sc = (sessionFX4.get(fx4ch3sc_url)).json()
fx4ch4sc = (sessionFX4.get(fx4ch4sc_url)).json()

# Request character size for display
print ("Choose display character size (20 to 120, default is",dftCharSize,") ",end="")
charSize = input()
if charSize == "":
    charSize = dftCharSize
charSize = int(float(charSize))
if charSize < 20:
    charSize = 20
if charSize > 120:
    charSize = 120
charSize = str(charSize)
print ("Using character size ",charSize,"\n")

# Request acquisition parameters
# Current range
print ("Current ranges 0:100nA, 1:100nA 10kHz, 2:1uA, 3:1uA 50kHz, 4:10uA, 5:100uA,
6:1mA, 7:10mA")
print ("Choose current range (0 to 7, default is",dftRange,") ",end="")
fx4range = input()
if fx4range == "":
    fx4range = dftRange
fx4range = int(float(fx4range))
if fx4range < 0:
    fx4range = 0
if fx4range > 7:
    fx4range = 7
print ("Setting range ",fx4range,"\n")
fx4range = str(fx4range)
fx4rangeJSON = json.dumps(fx4range)
sessionFX4.put(fx4range_url,fx4rangeJSON).json()
#
# Display units
print ("Choose measurement units (pA, nA, uA, mA, A, V, default is",dftUnits,") ",end="
")
fx4units = input()
i_unitsF = {'pA':'pa', 'nA':'na', 'uA':'ua', 'mA':'ma', 'A':'a', 'V':'v'}
try:
    fx4unitsJSON = json.dumps(i_unitsF[fx4units])
    sessionFX4.put(fx4units_url,fx4unitsJSON).json()
    print("Setting units ", fx4units,"\n")
except:
    print("Invalid - setting units nA","\n")
```

```
    fx4units = 'nA'
    fx4unitsJSON = json.dumps(i_unitsF[fx4units])
    sessionFX4.put(fx4units_url,fx4unitsJSON).json()

# Sample rate
print ("Choose sample rate (10 to 50000 default is",dftSampleRate,") ",end="")
fx4sampleRate = input()
if fx4sampleRate == "":
    fx4sampleRate = dftSampleRate
fx4sampleRate = int(float(fx4sampleRate))
if fx4sampleRate < 10:
    fx4sampleRate = 10
if fx4sampleRate > 50000:
    fx4sampleRate = 50000
print ("Setting sample rate ",fx4sampleRate," Hz","\n")
fx4sampleRateJSON = json.dumps(fx4sampleRate)
sessionFX4.put(fx4sampleRate_url,fx4sampleRateJSON).json()

# Readback settings
# Get range setting
i_ranges = ['100nA 1kHz','100nA 10kHz','1uA 5kHz','1uA 50kHz','10uA 50kHz','100uA
50kHz','1mA 50kHz','10mA 50kHz']
fx4rangeO = int(sessionFX4.get(fx4range_url).json())
fx4range = i_ranges[fx4rangeO]
#
# Get measurement units setting and correct capitalisation
i_unitsB = {'pa':'pA', 'na':'nA', 'ua':'uA', 'ma':'mA', 'a':'A', 'v':'V'}
fx4units = (sessionFX4.get(fx4units_url)).json()
fx4units = i_unitsB[fx4units]
#
# Set overrange condition
oRangeLimit = 1.19
oRangeR = {0:1e-7, 1:1e-7, 2:1e-6, 3:1e-6, 4:1e-5, 5:1e-4, 6:1e-3, 7:1e-2}
oRangeU = {'pA':1e12, 'nA':1e9, 'uA':1e6, 'mA':1e3, 'A':1e0, 'V':10}
oRange = oRangeR[fx4rangeO]*oRangeU[fx4units]
oRange = abs(oRange)*oRangeLimit
if fx4units == 'V': oRange = 10 * oRangeLimit

# Get averaging setting
fx4sampleRate = (sessionFX4.get(fx4sampleRate_url)).json()

# Format according to units
if fx4units == "mA":
    UnitFmt = "{:+10.4f}"
elif fx4units == "uA":
    UnitFmt = "{:+10.4f}"
elif fx4units == "nA":
    UnitFmt = "{:+10.3f}"
elif fx4units == "pA":
    UnitFmt = "{:+10.2f}"
else: UnitFmt = "{:+10.5e}"


# Print acquisition settings to console
print ("Range setting     :  ",fx4range)
print ("Current units     :  ",fx4units)
print ("Sample frequency  :  ",fx4sampleRate, "per second","\n")
```

```
print ("Caution - changing range or units settings in another active client will give
incorrect readings in this program")

print ("Display running ... ")


# Run Tk graphic display
# Set up display window
fx4_window = Tk()
fx4_window_Title = "PTC FX4 CURRENT METER   " + "   "+ ipaddr +"         Range:  " +
 fx4range +"     Units:  " + fx4units
fx4_window.wm_title(fx4_window_Title)
fx4_window.minsize(300,150)
fontTitle = "Courier " + charSize + " italic"
fontValues = "Courier " + charSize
padXY = 2

# Set up labels
headerGUI0 = Label(fx4_window,text = "Ch", height=1,width=4,justify="left",bd=8,font=fo
ntTitle,bg="black",fg='white',relief="groove",borderwidth=2,padx=padXY,pady=padXY)
headerGUI0.grid(row=0, column=0)
headerGUI1 = Label(fx4_window,text = hostName, height=1,width=15,justify="left",bd=8,fo
nt=fontTitle,bg="black",fg='white',relief="groove",borderwidth=2,padx=padXY,pady=padXY)
headerGUI1.grid(row=0, column=1)
ch1_GUI = Label(fx4_window,text = "1", height=1,width=4,justify="center",bd=8,font=font
Values,bg="black",fg='DodgerBlue',relief="groove",borderwidth=2,padx=padXY,pady=padXY)
ch1_GUI.grid(row=1, column=0)
ch2_GUI = Label(fx4_window,text = "2", height=1,width=4,justify="center",bd=8,font=font
Values,bg="black",fg='red2',relief="groove",borderwidth=2,padx=padXY,pady=padXY)
ch2_GUI.grid(row=2, column=0)
ch3_GUI = Label(fx4_window,text = "3", height=1,width=4,justify="center",bd=8,font=font
Values,bg="black",fg='green',relief="groove",borderwidth=2,padx=padXY,pady=padXY)
ch3_GUI.grid(row=3, column=0)
ch4_GUI = Label(fx4_window,text = "4", height=1,width=4,justify="center",bd=8,font=font
Values,bg="black",fg='yellow2',relief="groove",borderwidth=2,padx=padXY,pady=padXY)
ch4_GUI.grid(row=4, column=0)
sum_GUI = Label(fx4_window,text = "∑", height=1,width=4,justify="center",bd=8,font=font
Values,bg="gray",fg='white',relief="groove",borderwidth=2,padx=padXY,pady=padXY)
sum_GUI.grid(row=5, column=0)


# Get first current readings
ch1_val = (sessionFX4.get(fx4ch1_url)).json()
chSum = ch1_val
ch1_val = str(UnitFmt.format(ch1_val))
ch2_val = (sessionFX4.get(fx4ch2_url)).json()
chSum = chSum + ch2_val
ch2_val = str(UnitFmt.format(ch2_val))
ch3_val = (sessionFX4.get(fx4ch3_url)).json()
chSum = chSum + ch3_val
ch3_val = str(UnitFmt.format(ch3_val))
ch4_val = (sessionFX4.get(fx4ch4_url)).json()
chSum = chSum + ch4_val
ch4_val = str(UnitFmt.format(ch4_val))
chSum = str(UnitFmt.format(chSum))


# Loop to get repeat readings
```

```
def draw():
    global headerGUI1
    Label(fx4_window,text = hostName, height=1,width=15,justify="left",bd=8,font=fontTi
tle,bg="black",fg='white',relief="groove",borderwidth=2,padx=padXY,pady=padXY)
    headerGUI1.grid(row=0, column=1)
    global ch1_valGUI
    ch1_valGUI = Label(fx4_window,text = ch1_val +" " + fx4units, height=1,width=15,jus
tify="right",bd=8,font=fontValues,bg="black",fg='green2',relief="groove",borderwidth=2,
padx=padXY,pady=padXY)
    ch1_valGUI.grid(row=1, column=1)
    global ch2_valGUI
    ch2_valGUI = Label(fx4_window,text = ch2_val + " " + fx4units, height=1,width=15,ju
stify="right",bd=8,font=fontValues,bg="black",fg='green2',relief="groove",borderwidth=2
,padx=padXY,pady=padXY)
    ch2_valGUI.grid(row=2, column=1)
    global ch3_valGUI
    ch3_valGUI = Label(fx4_window,text = ch3_val +" " + fx4units, height=1,width=15,jus
tify="right",bd=8,font=fontValues,bg="black",fg='green2',relief="groove",borderwidth=2,
padx=padXY,pady=padXY)
    ch3_valGUI.grid(row=3, column=1)
    global ch4_valGUI
    ch4_valGUI = Label(fx4_window,text = ch4_val + " " + fx4units, height=1,width=15,ju
stify="right",bd=8,font=fontValues,bg="black",fg='green2',relief="groove",borderwidth=2
,padx=padXY,pady=padXY)
    ch4_valGUI.grid(row=4, column=1)
    global chSumGUI
    chSumGUI = Label(fx4_window,text = chSum + " " + fx4units, height=1,width=15,justif
y="right",bd=8,font=fontValues,bg="gray",fg='white',relief="groove",borderwidth=2,padx=
padXY,pady=padXY)
    chSumGUI.grid(row=5, column=1)

def updater():
# Get channel and sum readings and format
    ch1_val = (sessionFX4.get(fx4ch1_url)).json()
    if abs(ch1_val) > (oRange * fx4ch1sc):
        fgColour = 'orange'
    else:
        fgColour = 'white'
    chSum = ch1_val
    ch1_val = str(UnitFmt.format(ch1_val))
    ch1_valGUI.configure(text= ch1_val + " " + fx4units,fg = fgColour)

    ch2_val = (sessionFX4.get(fx4ch2_url)).json()
    if abs(ch2_val) > (oRange * fx4ch2sc):
        fgColour = 'orange'
    else:
        fgColour = 'white'
    chSum = chSum + ch2_val
    ch2_val = str(UnitFmt.format(ch2_val))
    ch2_valGUI.configure(text= ch2_val + " " + fx4units,fg=fgColour)

    ch3_val = (sessionFX4.get(fx4ch3_url)).json()
    if abs(ch3_val) > (oRange * fx4ch3sc):
        fgColour = 'orange'
    else:
        fgColour = 'white'
    chSum = chSum + ch3_val
    ch3_val = str(UnitFmt.format(ch3_val))
```

```
    ch3_valGUI.configure(text= ch3_val + " " + fx4units,fg=fgColour)

    ch4_val = (sessionFX4.get(fx4ch4_url)).json()
    if abs(ch4_val) > (oRange * fx4ch4sc):
        fgColour = 'orange'
    else:
        fgColour = 'white'
    chSum = chSum + ch4_val
    ch4_val = str(UnitFmt.format(ch4_val))
    ch4_valGUI.configure(text= ch4_val + " " + fx4units,fg=fgColour)

    chSum = str(UnitFmt.format(chSum))
    chSumGUI.configure(text= chSum + " " + fx4units,fg ='white')

# Update period
    fx4_window.after(20, updater)

# Animate dose display
draw()
updater()

# Run forever
fx4_window.mainloop()
```

5.3 Simple WebSockets Example

This example demonstrates the WebSockets interface, which is the preferred method for reading data from the FX4 when maximum bandwidth is required. WebSockets provide a real-time, full-duplex communication channel, allowing for faster and more efficient data transfer compared to other methods.

The example reads a series of samples, reports the average time per sample and maximum latency, and saves the data to a CSV file for later analysis. This setup allows for efficient real-time monitoring and easy data storage for post-processing.

The specific performance that can be achieved with WebSockets depends on the reliability of your Ethernet interface and the relative priority of your application. For optimal results, ensure that your network is stable and that the FX4's data transmission is prioritized if necessary.

	A	B	C	D	E	F
1	timestamp	elapsed	/fx4/adc/chan	/fx4/adc/chan	/fx4/adc/chan	/fx4/adc/channel
2	0	0	1.73E-10	-9.47E-11	-3.18E-11	-8.07E-11
3	0.007	0.007	2.09E-10	-9.47E-11	-3.18E-11	-8.07E-11
4	0.008	0.001	1.73E-10	-1.31E-10	-3.18E-11	-8.07E-11
5	0.01	0.002	1.73E-10	-1.31E-10	-6.84E-11	-1.17E-10
6	0.013	0.003	1.73E-10	-9.47E-11	-3.18E-11	-8.07E-11
7	0.015	0.002	2.09E-10	-1.31E-10	-3.18E-11	-8.07E-11
8	0.017	0.002	1.73E-10	-9.47E-11	-3.18E-11	-8.07E-11
9	0.019	0.002	2.09E-10	-9.47E-11	-3.18E-11	-8.07E-11
10	0.02	0.001	2.09E-10	-9.47E-11	-6.84E-11	-8.07E-11
11	0.023	0.003	1.73E-10	-9.47E-11	-3.18E-11	-8.07E-11
12	0.024	0.001	2.09E-10	-9.47E-11	-6.84E-11	-1.17E-10
13	0.026	0.002	1.73E-10	-9.47E-11	-3.18E-11	-8.07E-11
14	0.028	0.002	1.73E-10	-1.31E-10	-6.84E-11	-1.17E-10
15	0.029	0.001	1.36E-10	-9.47E-11	-3.18E-11	-8.07E-11
16	0.03	0.001	1.36E-10	-9.47E-11	-6.84E-11	-8.07E-11
17	0.033	0.003	2.09E-10	-1.31E-10	-3.18E-11	-8.07E-11
18	0.034	0.001	2.09E-10	-9.47E-11	-3.18E-11	-8.07E-11

```python
# Example Python script to display multiple readings from the FX4
# Write values to console and give option to save to csv

import websocket
import json
import time
import csv
import sys
import os

# Acquisition specific data
subs = [
    "/fx4/adc/channel_1/value",
    "/fx4/adc/channel_2/value",
    "/fx4/adc/channel_3/value",
    "/fx4/adc/channel_4/value",
]
ip = "192.168.100.111"
target_count = 1000
filename = os.path.join(os.getcwd(), "fx4_data.csv")

# Create the websocket and subscribe. Use "always_update" to ensure that updates will
always return all fields
# to faciliate parsing.
ws = websocket.create_connection("ws://" + ip)
ws.send(json.dumps({"event": "config", "data": {
    "always_update": True,
}}))
ws.send(json.dumps({"event": "subscribe", "data": {io: False for io in subs}}))

# Collect values
data = []
count = 0
t1 = time.time()
last_time = -1
max_elapsed_time = 0
```

```
start_time = 0
while count < target_count:
    # The get message data is empty, the data returned is set by the subscribe message.
    ws.send(json.dumps({"event": "get"}))

    # No need to wait after sending, you can just start receiving right away.
    response = json.loads(ws.recv())
    update_data = response["data"]

    # Process data if time has elapsed.
    if len(update_data) > 0:
        # Compute the elapsed time and if not zero process the record
        elapsed_time = update_data[subs[0]][0][1] - last_time
        if(elapsed_time != 0):
            # Record the maximum time unless it is sample 0. If sample 0 record the
start time.
            if(count > 0):
                if(max_elapsed_time < elapsed_time):
                    max_elapsed_time = elapsed_time
                    sample = count
            else:
                start_time = update_data[subs[0]][0][1]
                elapsed_time = 0
            last_time = update_data[subs[0]][0][1]
            # Process the list
            channels = []
            channels.append((update_data[subs[0]][0][1]-start_time)/1e9)
            channels.append(elapsed_time/1e9)
            for io in subs:
                channels.append(update_data[io][0][0]/1e9)
            data.append(channels)
            count += 1

# Wait for the last message to be sent.
print( f"Average sample time : {(1000*(time.time()-t1)/target_count):.1f} ms",  )
print( f"Maximum latency time: {(max_elapsed_time/1e6):.1f} ms on sample", sample )

with open(filename, 'w', newline='') as csvFile:
   writer = csv.writer(csvFile)
   writer.writerow(["timestamp"] + ["elapsed"] + subs)
   writer.writerows(data)
csvFile.close()


ws.close()
```