



Calnex Paragon-neo Calnex Paragon-100G

REMOTE CONTROL GUIDE

Contents

Contents.....	2
Introduction	3
Physical Connection to the Paragon Instrument	3
Overview	3
Generating Remote Control Scripts using the Script Recorder	3
Using Remote Control from Tcl	4
Prerequisites	4
Location of the Module	4
Using the Tcl Module.....	5
Using Remote Control from Python.....	5
Prerequisites	5
Location of the Module	5
Using the Python Module	5
Using Remote Control from Perl	5
Prerequisites	5
Location of the Module	5
Using the Perl Module	5
Content of the Paragon Modules.....	6
Scripting Conventions	6
Compatibility Mode.....	6
On-Line Command Reference	6
Command Reference Concepts	8
Commands and Parameters	8
Settings	8
Script Command Sequencing.....	9
Simple Script Examples	9
TCL	10
Python	10
Perl.....	11
Calnex Analyser Tool (CAT) Overview.....	11
CAT Files.....	11
CAT Metrics.....	12
CAT Metric / Measurement Analysis	14
Retrieving Measurement Results from CAT	14
Ports, Measurements, Metrics and Extended Id.....	14
Example: Enabling a Measurement and Retrieving a Result	15
Accessing Table Data	16
Example: Retrieving Table Data	16
Hints and Tips	17

Introduction

Many users of Paragon have a requirement to automate the testing of their devices. To support this, remote control functionality is built-in to the Calnex Paragon instruments as a standard feature.

This document details the remote control commands supported by Paragon-100G and Paragon-neo. These instruments may also be controlled using Paragon-X style commands – please see the Remote Control Guide installed with your Paragon-X software for more details

Physical Connection to the Paragon Instrument

Remote control commands are sent directly to the Paragon-100G / Paragon-neo from your script. The computer executing your script can exist anywhere in the network as long as it can access the instrument.

In other words, if the instrument can be reached via a web browser, it can be controlled via a script.

Overview

Testing a device using a Paragon instrument (and associated remote control) involves 3 main components:

- Hardware configuration and capture control
- Metrics analysis and visualisation (using the Calnex Analyser Tool - CAT)
- PTP and ToD Message analysis (using the Calnex Field Verifier – PFV)

The CAT and PFV allow the in-depth analysis of captured data, both on previously captured data (which does not require access to hardware) and in real-time whilst running a live capture on an instrument.

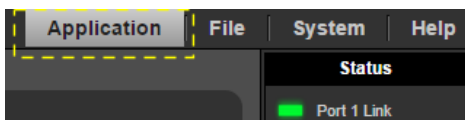
This document describes how to use the remote control functionality for all of the above components.

Tcl, Perl and Python are supported.

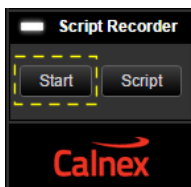
Generating Remote Control Scripts using the Script Recorder

The Paragon-100G and Paragon-neo can record user operations and convert these into scripted commands. This makes script creation very simple – record keypresses in the browser and then use the recorded script as part of your test program. The script recorder can log commands for the CAT and PFV as well as instrument control.

To use script recorder, open a web browser and enter the URL for your instrument. Then select **Application** from the menu bar:

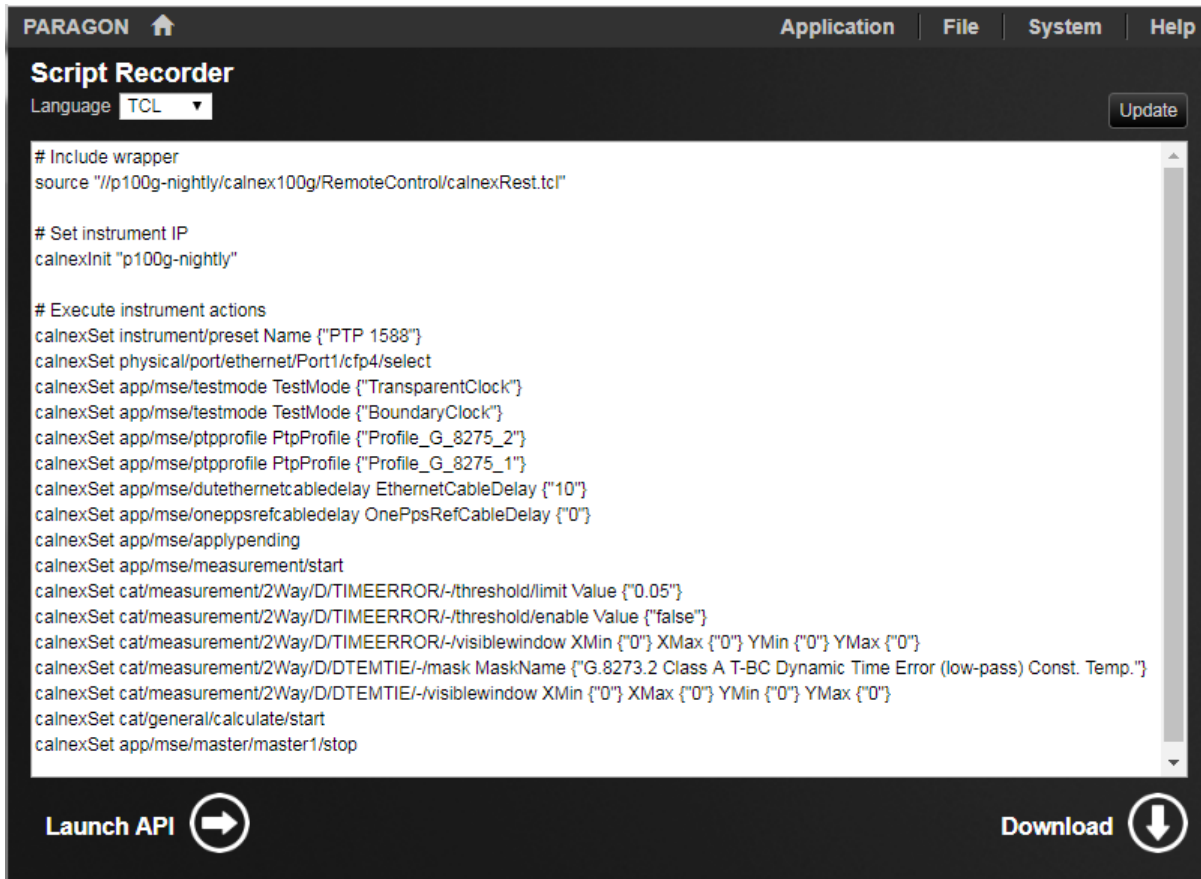



To start recording operations, select **Start** from the **Script Recorder** panel in the bottom left-hand corner of the browser window:



Now configure the Paragon and perform the operations you would like scripted

When you have completed your desired operations, click **Script**. You will now see a new page with your recorded script:



PARAGON  Application | File | System | Help



Script Recorder

Language **TCL** Update

```
# Include wrapper
source //p100g-nightly/calnex100g/RemoteControl/calnexRest.tcl"

# Set instrument IP
calnexInit "p100g-nightly"

# Execute instrument actions
calnexSet instrument/preset Name {"PTP 1588"}
calnexSet physical/port/ethernet/Port1/cfp4/select
calnexSet app/mse/testmode TestMode {"TransparentClock"}
calnexSet app/mse/testmode TestMode {"BoundaryClock"}
calnexSet app/mse/ptpprofile PtpProfile {"Profile_G_8275_2"}
calnexSet app/mse/ptpprofile PtpProfile {"Profile_G_8275_1"}
calnexSet app/mse/dutethernetcabledelay EthernetCableDelay {"10"}
calnexSet app/mse/oneppsrefcabledelay OnePpsRefCableDelay {"0"}
calnexSet app/mse/applypending
calnexSet app/mse/measurement/start
calnexSet cat/measurement/2Way/D/TIMEERROR/-/threshold/limit Value {"0.05"}
calnexSet cat/measurement/2Way/D/TIMEERROR/-/threshold/enable Value {"false"}
calnexSet cat/measurement/2Way/D/TIMEERROR/-/visiblewindow XMin {"0"} XMax {"0"} YMin {"0"} YMax {"0"}
calnexSet cat/measurement/2Way/D/DTEMTIE/-/mask MaskName {"G.8273.2 Class A T-BC Dynamic Time Error (low-pass) Const. Temp."}
calnexSet cat/measurement/2Way/D/DTEMTIE/-/visiblewindow XMin {"0"} XMax {"0"} YMin {"0"} YMax {"0"}
calnexSet cat/general/calculate/start
calnexSet app/mse/master/master1/stop
```

Launch API  **Download** 

The default script language is Tcl. However, you can change this to Perl or Python from the **Language** pull-down. This is possible at any time even after the script has been recorded.

Clicking **Update** in the top-right hand corner of the recorded script window will update the script with any actions that have been recorded since the script window was last refreshed.

Click **Stop** on the main instrument page under **Script Recorder** to stop recording.

The recorded script can be copied from the script window or downloaded to your local PC.

Selecting **Launch API** will open a page through which live interaction can be made with the API. The same page can also be used as a reference for API services, as described in section “On-Line Command Reference”.

Using Remote Control from Tcl

A Tcl module is provided for remote control functionality. This module has been verified using ActiveState Tcl, version 8.5. It is recommended that this is the version you use.

The Tcl module provides a simple bridge between Tcl and the network interface protocol used to talk to the Paragon application.

Prerequisites

- ActiveTCL version 8.5
- “REST” package

On Windows this is obtained by executing shell command “teacup update”.

Note: depending on internet connection bandwidth, this step may take several minutes.

Location of the Module

The Paragon Tcl module (`calnexRest.tcl`) is located in:

```
//<instrumentIpAddress>/calnex100G/RemoteControl/
```



Using the Tcl Module

The Paragon Tcl module must be referenced using the Tcl `source` command prior to running Tcl commands or scripting i.e.

```
source "//<instrumentIpAddress>/calnex100G/RemoteControl/calnexRest.tcl"
```

Using Remote Control from Python

A Python module is provided for remote control functionality. This module has been verified using ActiveState Python, version 3.4. It is recommended that the Python interpreter installed is the same version or newer otherwise Python functionality may not work correctly.

The Python module provides a simple bridge between Python and the network interface protocol used to talk to the Paragon application.

Prerequisites

- Python v3.4 or later
- "requests" package:
On Windows this is obtained by executing shell command:
`[Python install directory]\Scripts\pip install requests`

Location of the Module

The Paragon Tcl module (`calnexRest.py`) is located in:

```
//<instrumentIpAddress>/calnex100G/RemoteControl/
```

Using the Python Module

The Python module must be imported before it can be used i.e.

```
import sys
sys.path.append(r'\\<instrumentIpAddress>\calnex100G\RemoteControl')
from calnexRest import calnexInit, calnexGet, calnexSet, calnexCreate, calnexDel,
calnexGetVal
```

Using Remote Control from Perl

A Perl module is provided for remote control functionality. This module has been verified using ActivePerl 5.20. It is recommended that the Perl interpreter installed is the same version or newer otherwise Perl functionality may not work correctly.

The Perl module provides a simple bridge between Perl and the network interface protocol used to talk to the Paragon application.

Prerequisites

- ActivePerl 5.20
- "REST-client" package:
On Windows this is obtained by executing shell command:
`[Perl install directory]\bin\ppm install REST-client`

Location of the Module

The Paragon Perl module (`calnexRest.pm`) is located in:

```
//<instrumentIpAddress>/calnex100G/RemoteControl/
```

Using the Perl Module

The Python module must be referenced before it can be used i.e.

```
use lib '//<instrumentIpAddress>/calnex100G/RemoteControl';
use calnexRest;
```

Content of the Paragon Modules

Each module provides the following functions:

- `calnexInit`: must be called before any other function. The parameter is the IP address of the instrument.
- `calnexSet`: sets a value for a specified setting
- `calnexGetVal`: returns the value of a single specified setting
- `calnexGet`: can return a single value or a set of values



Note: The wrapper functions and documentation for Paragon-100G and Paragon-neo have been revised. The information in this document is relevant only for Paragon-100G versions later than 06.03 and Paragon-neo versions later than 00.05.

Scripting Conventions

The native Paragon-100G and Paragon-neo API is an HTTP RESTful API. Regardless of the script language selected, all commands ultimately will be translated to RESTful API calls.

The Paragon script modules define specific remote control functions for the Paragon products. Each function maps to one of more associated HTTP RESTful commands. For example:

```
calnexSet [command]
```

...maps to RESTful:

```
PUT [command]
```

Therefore, with minimal rework, a script generated in one language can be modified to run in another language. The remote control commands themselves do not need to change, only the 'header' section of the scripts differ.

Compatibility Mode

Scripts generated for Paragon-X products can be run on P100G, although **only commands pertaining to features common to both products are supported**.

For a Paragon-X script to run with P100G, its `connect` command must be modified to specify only the IP address of the P100G instrument. E.g. the following:

```
connect 192.168.254.1 localhost 9990 9000
```

...should be modified to:

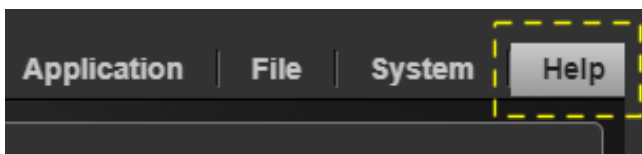
```
connect localhost 192.168.254.1
```

The IP address referred to in the modified script should be the same IP address as is used to reach the instrument via a web browser, i.e. that which is indicated on the instrument LCD panel after startup.

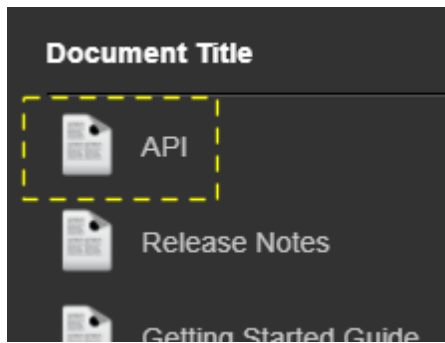
On-Line Command Reference

An online reference for remote control commands can be obtained via the 100G web application.

First select "Help" from the menu bar:



Then "API" from the Document list:



The resulting page represents a list of all available REST API commands. Click on any of the rolled-up sections (**app**, **cat**, **instrument**, etc.) to reveal a list of individual commands.

GET services are invoked via `calnexGet` script commands.

PUT services are invoked via `calnexSet` script commands.

Command Reference Concepts

The following concepts are common themes throughout the Paragon remote control manual.

Commands and Parameters

Since the native API for Paragon-100G and Paragon-neo is RESTful, each command is actually a URL. The format of commands is therefore:

```
<root>/<node>/... <node>/.../<node>
```

Where `<node>` may either be a fixed string or a variable string. In the case of the latter, the `<node>` is similar to a parameter which is encoded in the URL. For example:

To retrieve a SyncE TIE measurement from the CAT, you could use:

```
calnexGet "cat/measurement/SyncE/A/TIE/-"
```

The command being used here is:

```
cat/measurement/{Name}/{Port}/{MetricType}/{ExtId}
```

Where "cat" and "measurement" are fixed strings. "SyncE", "A", "TIE" and "-" are all variable strings that can be considered as parameters with, for example, parameter `Name = "SyncE"`.

In addition some parameters may be sent as arguments to the a command. For example:

<FIND AN EXAMPLE>

Settings

Individual instrument settings may be set or queried using remote control commands. Most settings can be set and queried individually, but some may only be queried. The general syntax for commands is as follows:

To **set** a setting:

```
calnexSet <url> <value>
```

To **query** a setting:

```
calnexGet <url>
```

This will return one or more `<parameter value>` results. The `<parameter value>` response from `calnexGet` will be either a single value or a JSON string containing multiple values. For example, the command:

```
calnexGet "cat/measurement/SyncE/A/TIE/-"
```

will return a JSON string similar to:

```
{'Enable': True, 'MaskName': 'No Mask', 'MaskState': 'NoMask', 'RemoveOffset':  
{'Enable': False, 'Offset': -0.0001, 'OffsetInRange': -0.0001, 'Unit': 'ppm',  
'InContext': True}}
```

The response contains a number of key/value pairs where the key is the name of an instrument setting. From the example above, `MaskName` is a key and `No Mask` is its associated value. In this case, this indicates that the SyncE TIE measurement is enabled.

Note that values may themselves be a series of key/value pairs. In the example above, the value associated with the key `RemoveOffset` is:

```
{'Enable': False, 'Offset': -0.0001, 'OffsetInRange': -0.0001, 'Unit': 'ppm',  
'InContext': True}
```

Each scripting language has different methods for extracting specific fields from these responses. See Simple Script Examples below for examples in each scripting language.

An additional helper function is available in the Paragon modules to retrieve a single value from a specified key:

```
calnexGetVal <url> <key>
```

In the example above, the command:


```
calnexGetVal "cat/measurement/SyncE/A/TIE/-" "MaskName"  
will return "No Mask".
```

The parameters taken by these commands are described in more detail later in this document.

Script Command Sequencing

The first Paragon command in any script must be `calnexInit`. This initialises the wrapper with the IP address of the instrument being controlled.

In general, a number of remote commands rely on the instrument being in a specific state. Before invoking these remote control commands, it is necessary to ensure that the system is in the appropriate state. For example, a script wishing to retrieve gain values from a SyncE Wander Transfer Table Sine Generation operation must first ensure that generation has completed.

The remote control API provides services that can be used to determine run state for each instrument "application" – items in square brackets are elements (i.e. fields) of the returned object(s):

- `/app/generation/synce/esmc/{Port}` [RunState]
- `/app/generation/synce/wander` [RunState]
- `/app/measurement/synce/wander/{PortNumber}` [RunState]
- `/cat/general/status` [IsApiCurrentlyProcessing]
- `/cat/general/status` [IsOpeningInProgress]

Taking the earlier example of SyncE Wander Generation, a remote control script may define the following utility function (in Python, for example) that would be called to ascertain whether SyncE Wander Generation had completed:

```
# Do not return until Wander Gen has stopped  
def waitForWanderGenStopped():  
    while (1):  
        res = calnexGet("/api/app/generation/synce/wander")  
        runState = res["RunState"]  
        if(runState == "Stopped"):  
            break  
        else:  
            time.sleep(2)
```

Likewise, before attempting to retrieve metrics from CAT:

```
# Do not return until CAT is ready  
def waitForCat():  
    while (1):  
        res = calnexGet("/api/cat/general/status")  
        isOpeningInProgress = res["IsOpeningInProgress"]  
        if(isOpeningInProgress == False):  
            break  
        else:  
            time.sleep(2)
```

Simple Script Examples

The following outlines scripts structure, language-by-language. Code highlighted in yellow represents that which is most likely to require editing.

TCL

```
# Set instrument IP
set IPAddress "192.168.254.1"

# Include wrapper
source "$IPAddress/calnex100g/RemoteControl/paragon.tcl"

# Initialise
calnexInit $IPAddress

# Execute instrument actions
# Set SyncE Wander measurement run state
calnexSet app/measurement/synce/wander/Port1/start
calnexSet app/measurement/synce/wander/Port1/stop

# Get the SyncE Wander sample period value
set responseJSON [calnexGet "app/measurement/synce/wander/Port1/sampleperiod"]
set value [dict get $responseJSON "Value"]
puts $value

# Get the SyncE Wander Generation transfer table row 2 'number of cycles' value
set responseJSON [calnexGet "app/generation/synce/wander/transfer/table"]
set rows [dict get $responseJSON "Rows"]
set row1 [lindex $rows 1]
set cycles [dict get $row1 "Cycles"]
set cyclesValue [dict get $cycles "Value"]
puts $cyclesValue
```

Variable `IPAddress` must be set to the IP address of the 100G instrument (as indicated on the instrument LCD panel after startup). When the script is being run against the same instrument from which it was generated, then there would not normally be any need to edit this variable, unless the instrument's IP address has changed since the script was generated.

Python

```
# Set instrument IP
IPAddress = "192.168.254.1"

# Include wrapper
import sys
sys.path.append('///' + IPAddress + '/calnex100g/RemoteControl/')
from calnexRest import calnexGet, calnexSet, calnexCreate, calnexDel, calnexGetVal

# Initialise
calnexInit(IPAddress)

# Execute instrument actions
# Set SyncE Wander measurement run state
calnexSet("app/measurement/synce/wander/Port1/start")
calnexSet("app/measurement/synce/wander/Port1/stop")

# Get the SyncE Wander sample period value
responseJSON = calnexGet("app/measurement/synce/wander/Port1/sampleperiod")
print(responseJSON['Value'])

# Get the SyncE Wander Generation transfer table row 2 'number of cycles' value
responseJSON = calnexGet("app/generation/synce/wander/transfer/table")
print(responseJSON['Rows'][1]['Cycles']['Value'])
```

Variable `IPAddress` must be set to the IP address of the 100G instrument (as indicated on the instrument LCD panel after startup). When the script is being run against the same instrument from which it was generated, then there would not normally be any need to edit this variable, unless the instrument's IP address has changed since the script was generated.

Perl

```
BEGIN {$IpAddress = '192.168.254.1'};

# Include wrapper
use lib "//$IpAddress/calnex100g/RemoteControl/";
use calnexRest;

# Initialise
calnexInit($IpAddress);

# Execute instrument actions
# Set SyncE Wander measurement run state
calnexSet("app/measurement/synce/wander/Port1/start");
calnexSet("app/measurement/synce/wander/Port1/stop");

# Get the SyncE Wander sample period value
my $responseJSON = calnexGet("app/measurement/synce/wander/Port1/sampleperiod");
print $responseJSON->{'Value'} . "\n";

# Get the SyncE Wander Generation transfer table row 2 'number of cycles' value
my $responseJSON = calnexGet("app/generation/synce/wander/transfer/table");
print $responseJSON->{'Rows'}[1]->{'Cycles'}->{'Value'} . "\n";
```

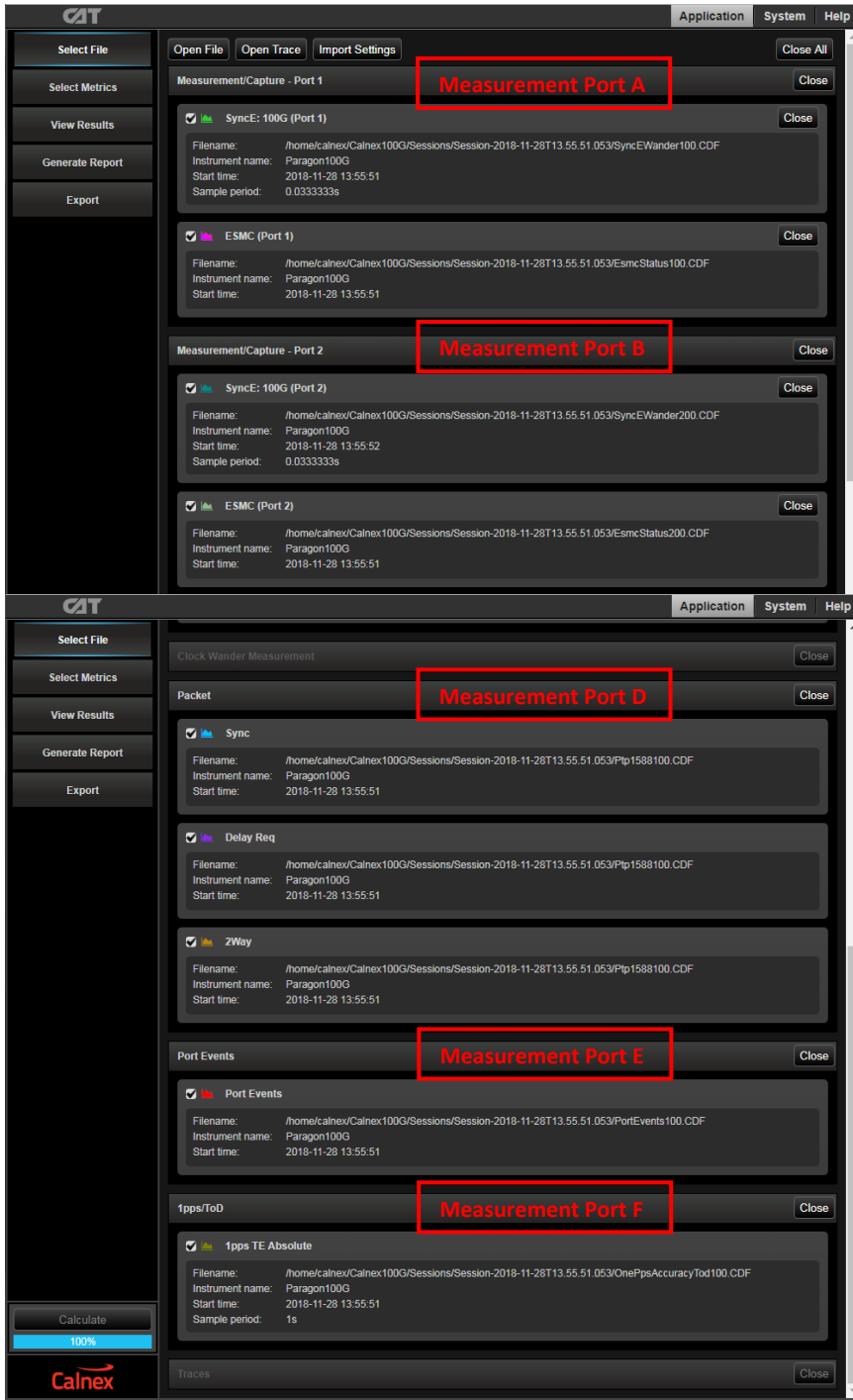
Variable `$IpAddress` must be set to the IP address of the 100G instrument (as indicated on the instrument LCD panel after startup). When the script is being run against the same instrument from which it was generated, then there would not normally be any need to edit this variable, unless the instrument's IP address has changed since the script was generated.

Calnex Analyser Tool (CAT) Overview

The commands in this section describe the commands used to control the Calnex Analyser Tool (CAT) settings and behaviour. The CAT tool is the main data analysis tool for Paragon products. It allows a user to load a pre-recorded file and have the raw data analysed immediately or it can be used as a capture is happening to obtain analysis in "real time".

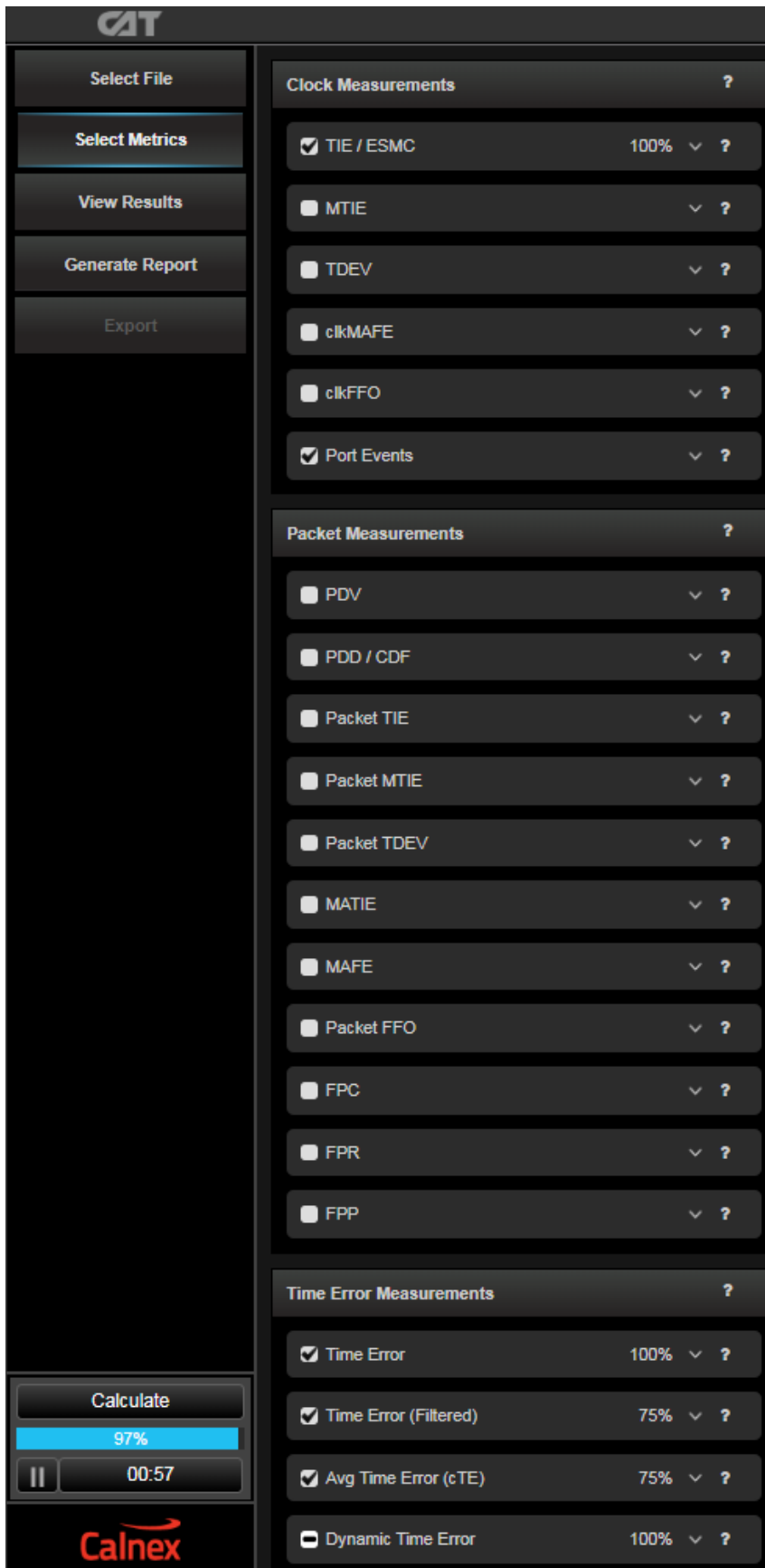
CAT Files

The following images show typical CAT displays indicating how the layout is organised. In the **Select File** pane, the files loaded are indicated along with the *Measurement Port* to which they have been allocated. Note that the CAT *Measurement Port* has nothing to do with the physical ports on the instrument; it is simply the name given to the file grouping used by the CAT.

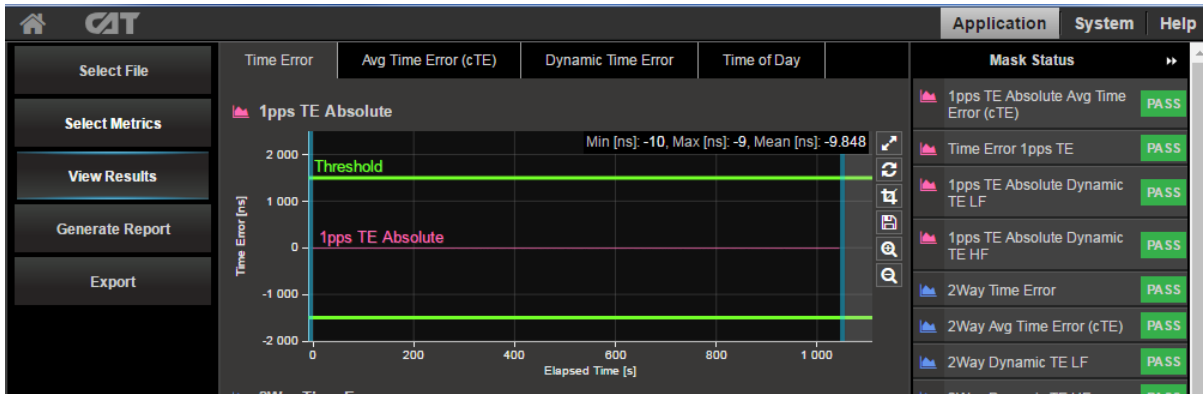


CAT Metrics

In the **Select Metrics** pane, the metrics available for the loaded files are displayed and can be selected or de-selected.



When a metric is selected, it appears in the **View Results** pane as a *Tab*.



CAT Metric / Measurement Analysis

Whenever a measurement has been performed then the user can select which metrics they want to be calculated and displayed. Only the metrics that are available for the currently loaded raw data sets can be selected.

For example, if a 1588 capture file is loaded then the PDV data can be displayed. Similarly, if a wander file of any sort is loaded then MTIE, TDEV and clock offset can be calculated and displayed.

Retrieving Measurement Results from CAT

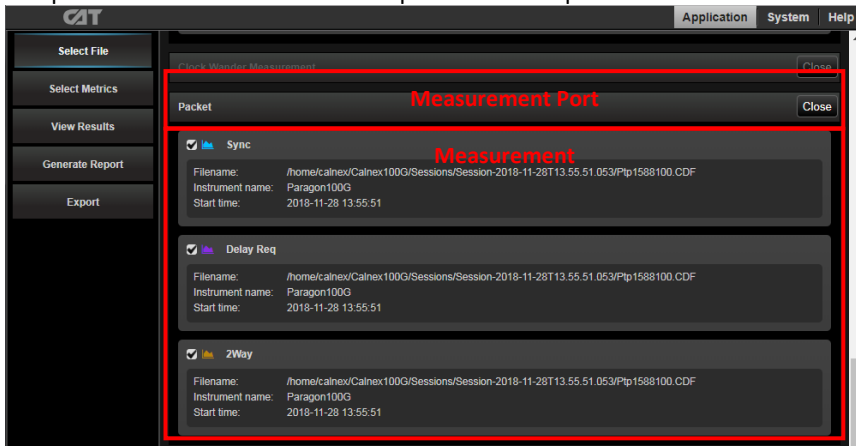
Measurements can be configured and retrieved using a number of commands in `/cat/measurement/`

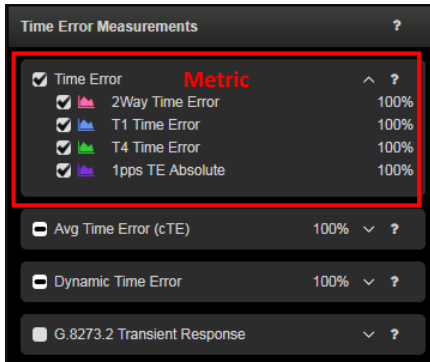
Ports, Measurements, Metrics and Extended Id

All commands to retrieve measurement results from the CAT involve four parameters:

- The *Measurement Port* in which the data file has been loaded
- The *Measurement* of interest
- The *Metric* from the *Measurement*
- The *Extended Id* (ExtId) for the *Metric*

The picture below shows how these parameters map to the CAT UI:





The table below describes the valid values for each of these parameters.

Data Type	CAT Port	Measurement	Metric	ExtId
SyncE Wander / Jitter / ESMC (measured on instrument Port 1)	A	ESMC	ESMC	Rx, Tx
		SyncE	TIE, MTIE, TDEV, CLKMAFE, CLKFFO	-
		Jitter	LongTermJitterPkPk, LongTermJitterRms, ShortTermJitterPkPk,	-
SyncE Wander / Jitter / ESMC (measured on instrument Port 2)	B	ESMC	ESMC	Rx, Tx
		SyncE	TIE, MTIE, TDEV, CLKMAFE, CLKFFO	-
		WanderGain	WanderTransfer	-
Packet Measurements	D	2Way, Sync, DelayReq	Packet: PDV, CF, PDD, CDF, PKTTIE, PKTMTIE, PKTTDEV, MATIE, MAFE, PKTFFO, FPC, FPR, FPP, Time Error: TIMEERROR, CTE DTE, DTEMTIE, DTETDEV, DTEHF, TransientResponse (2Way only)	-
Port Events	E	PortEvents	PortEvents	-
1pps / ToD	F	1ppsTEAbsolute	Packet: MTIE, TDEV, CLKFFO, Time Error: ONEPPS, CTE, DTE, DTEMTIE, DTETDEV, DTEHF, TransientResponse	-

Example: Enabling a Measurement and Retrieving a Result

The example below enables SyncE MTIE and TDEV measurements and applies a G.8262 mask to MTIE. The mask pass/fail result is then returned.

```
# Enable MTIE and TDEV in the CAT
calnexSet("cat/measurement/SyncE/A/MTIE/-/enable", "Value", True)
calnexSet("cat/measurement/SyncE/A/TDEV/-/enable", "Value", True)
#calnexSet("cat/measurement/SyncE/A/MTIE/-/isenabled", "Value", True)
#calnexSet("cat/measurement/SyncE/A/TDEV/-/isenabled", "Value", True)

# Select the G.8262 mask and calculate
calnexSet("cat/measurement/SyncE/A/MTIE/-/mask", "MaskName", 'G.8262 Wander Generation EEC
Op1')
calnexSet("cat/general/calculate/start")
time.sleep(5)

# Get the pass/fail result from the CAT
pfMTIE = calnexGetVal("cat/measurement/SyncE/A/MTIE/-", 'MaskState')
pfTDEV = calnexGetVal("cat/measurement/SyncE/A/TDEV/-", 'MaskState')

print ("MTIE mask: {}   TDEV mask: {}".format(pfMTIE, pfTDEV))
```

Accessing Table Data

A number of CAT measurements are represented with both a chart and a table. Table data can be retrieved using commands in `/cat/measurement/{Name}/{Port}/{MetricType}/{ExtId}/dgv/`

Example: Retrieving Table Data

The following is an Python example of how to access table data. This example shows how ESMC data may be monitored during a G.8262 Wander Tolerance test.

Note that table columns may be re-arranged by the user. This means that, before accessing the data from specific column, it is necessary to determine the column position in the table.


```

# Configure and start ESMC generation on port 2 - QL-PRC
calnexSet("app/generation/synce/esmc/Port2/ssmvalue", "SsmValue", 'QL-PRC')
calnexSet("app/generation/synce/esmc/Port2/start")
# Start Synce/ESMC measurement on port 1
calnexSet("app/measurement/synce/wander/Port1/start")
# Start generating the table sine tolerance pattern on port 2
calnexSet("app/generation/synce/wander/mode", "Mode", 'Tolerance', "OperationType", 'TableSine')
calnexSet("app/generation/synce/wander/tolerance/mask")
calnexSet("app/generation/synce/wander/start")
time.sleep(3)
# Disable the ESMC Tx measurement on port 1 - we are only interested in Rx
calnexSet("cat/measurement/ESMC/A/ESMC/Tx/enable", "Value", False)

# To get the ESMC transitions, we first have to figure out which columns
# contain the information we want
# The columns can be moved around by the user, so we can't rely on them being in a fixed position
# So, get the column info
sampleNumberCol = -1
esmcRxCol = -1
columns = calnexGetVal("cat/measurement/ESMC/A/ESMC/Rx/dgv/columns", "ColumnLayout")
for column in columns:
    #print(column)
    if (column["InnerName"] == "SampleNumber"):
        sampleNumberCol = column["Position"]
    elif (column["InnerName"] == "EsmcRx"):
        esmcRxCol = column["Position"]

if (sampleNumberCol == -1 || esmcRxCol == -1):
    print("ERROR: Unable to determine ESMC column positions. Aborting...")
    exit

print("sampleNumberCol: ", sampleNumberCol)
print("esmcRxCol: ", esmcRxCol)

# Get initial number of ESMC entries in the table
# This is an ESMC Measurement on port 1 (CAT measurement port A).
# The Metric is ESMC and we want Rx values only
initialNumRows = calnexGetVal("cat/measurement/ESMC/A/ESMC/Rx/dgv/count", "Count")
print("InitialNumRows: ", initialNumRows)

# Display the initial ESMC value
rowData = calnexGetVal("cat/measurement/ESMC/A/ESMC/Rx/dgv/data/0/1", "Rows")

esmcSampleNumber = rowData[0]["DataRow"]["ColumnData"][sampleNumberCol]["Value"]
esmcValue = rowData[0]["DataRow"]["ColumnData"][esmcRxCol]["Value"]
if ("No ESMC Data" in esmcValue):
    esmcValue = "No ESMC Data"
print("Initial ESMC: {} {}".format(esmcSampleNumber, esmcValue))

lastNumRows = initialNumRows
# Monitor ESMC while test is running
while isSynceNoiseGenerationRunning():
    time.sleep(30) # Polling period - use 30 seconds for this example
    currentNumRows = calnexGetVal("cat/measurement/ESMC/A/ESMC/Rx/dgv/count", "Count")
    if (currentNumRows > lastNumRows):
        # There has been a change - print out the changed table contents
        for idx in range(lastNumRows-1, currentNumRows-1):
            rowData = calnexGetVal("cat/measurement/ESMC/A/ESMC/Rx/dgv/data/"+str(idx)+"/"+str(1),
"Rows")
            esmcSampleNumber = rowData[0]["DataRow"]["ColumnData"][sampleNumberCol]["Value"]
            esmcValue = rowData[0]["DataRow"]["ColumnData"][esmcRxCol]["Value"]
            print("Changed Rx ESMC: {} {}".format(esmcSampleNumber, esmcValue))
            lastNumRows = currentNumRows

# Stop ESMC generation and measurement
calnexSet("app/generation/synce/esmc/Port2/stop")
calnexSet("app/measurement/synce/wander/Port1/stop")

```

Hints and Tips

Using the script recorder is the quickest and easiest way to determine how to make a setting (e.g. how to use `calnexSet`). Use the script recorder to determine the command and the parameter settings and then for further detail, use the on-line help.



Since the script recorder cannot record calnexGet, it is more difficult to determine the commands and parameters to use when retrieving data or results. In this case, use the on-line help documentation in conjunction with the **Try it out** button. In addition, many of the GET commands provide information that can help establish the path or parameters you need.

It should also be remembered that the above information can also be read, manipulated or printed in your script. This is useful in determining the structure of the returned values.