Cryostream 800

Software Architecture

Ref: Cryostream800.SAD Rev. 0.1

February 21

EMBL HAMBURG
Notkestrasse 85
22607 Hamburg

Document Status

Reference	Version	Date	Status
Cryostream800.SAD	**Rev. 0.1**	**17/03/2020**	**WIP**

Author(s)	Moises Bueno
Short Description	Description of the software architecture for cryogenics
Approvals:	Stefan Fiedler **Date**: 17/03/2020

Changes Registry

Version	Rev.	Date	Details	Author
0	**1**	**17/03/2020**	**Initial**	**Moises Bueno**

Acronyms & Abbreviations

Acronym	Meaning
EMBL	European Molecular Biology Laboratory
GUI	Graphical User Interface
TINE	Three-fold Integrated Networking Environment
LN2	Liquid Nitrogen
MX	Macrocrystallography
CDH	Crystal Direct Harvester
EPICS	Experimental Physics and Industrial Control Systems
PC	Personal Computer
VI	Virtual Instrument (Labview code)

Table of Contents

Table of Figures

CHAPTER 1. INTRODUCTION

1.1 Purpose

This document contains a detailed description of the software developed to connect, readout values and control the Cryostream 800 of Oxford Cryosystems using LabView.

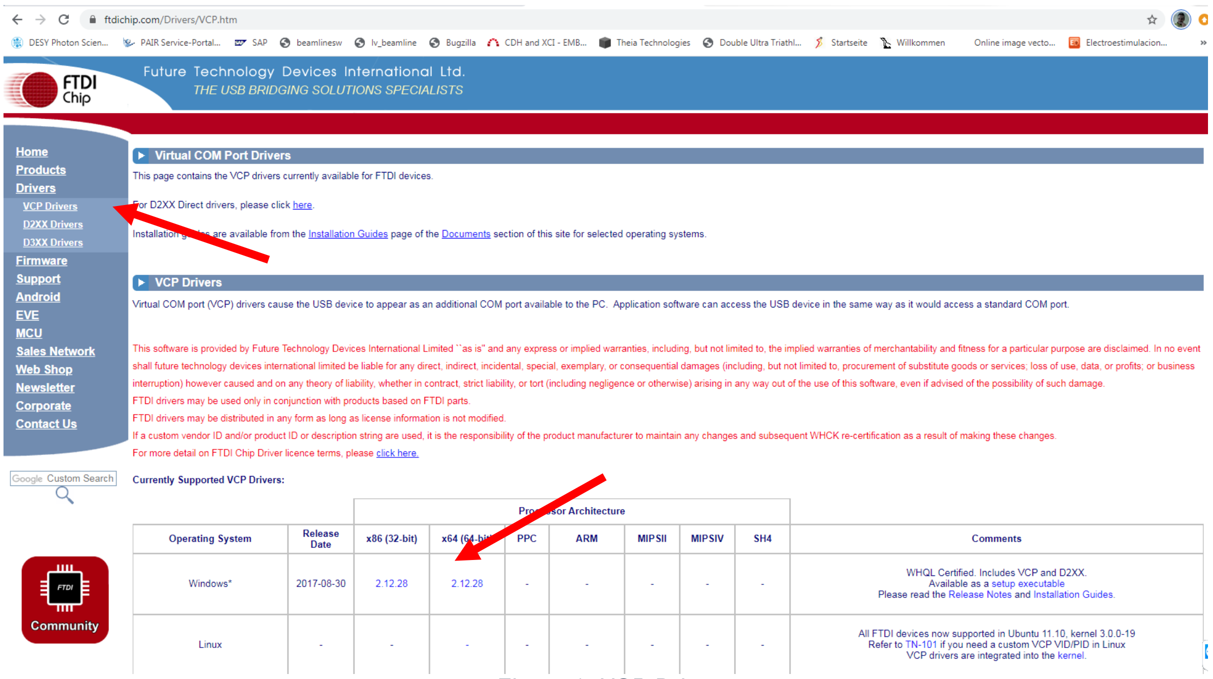
1.2 Overview

EMBL has at least 4 cryostream controllers, 3 in operation and 1 spare unit. They are located in P13 experimental hutch, P14 experimental hutch and in the CD harvester room. The cryostreams 800 allows the user to set the temperature of the LN2 gas and readout a hugh amount of values which are important for the correct functioning of the system. Oxford Cryosystems provides a piece of software called Cryoconnector which is the bridge between the hardware and the LabView application.

CHAPTER 2. Cryoconnector Software

The cryostream 800 controller is possible to connect via usb to a personal computer (PC) however the usb driver has to be downloaded and installed manually. The drivers can be downloaded from http://www.ftdichip.com/FTDrivers.htm

1. Click on VCP Drivers under Drivers
2. Choose according your Windows configuration


ftdichip.com/Drivers/VCP.htm
DESY Photon Scien... PAIR Service-Portal... SAP beamlinesw lv_beamline Bugzilla CDH and XCI - EMB... Theia Technologies Double Ultra Triathl... Startseite Willkommen Online image vecto... Electroestimulacion...
FTDI Chip
Future Technology Devices International Ltd.
THE USB BRIDGING SOLUTIONS SPECIALISTS
Home
Products
Drivers
VCP Drivers
D2XX Drivers
D3XX Drivers
Firmware
Support
Android
EVE
MCU
Sales Network
Web Shop
Newsletter
Corporate
Contact Us
Google Custom Search
FTDI
Community

Virtual COM Port Drivers

This page contains the VCP drivers currently available for FTDI devices.

For D2XX Direct drivers, please click here.

Installation guides are available from the Installation Guides page of the Documents section of this site for selected operating systems.

VCP Drivers

Virtual COM port (VCP) drivers cause the USB device to appear as an additional COM port available to the PC. Application software can access the USB device in the same way as it would access a standard COM port.

This software is provided by Future Technology Devices International Limited ``as is'' and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall future technology devices international limited be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.
FTDI drivers may be used only in conjunction with products based on FTDI parts.
FTDI drivers may be distributed in any form as long as license information is not modified.
If a custom vendor ID and/or product ID or description string are used, it is the responsibility of the product manufacturer to maintain any changes and subsequent WHCK re-certification as a result of making these changes.
For more detail on FTDI Chip Driver licence terms, please click here.

Currently Supported VCP Drivers:

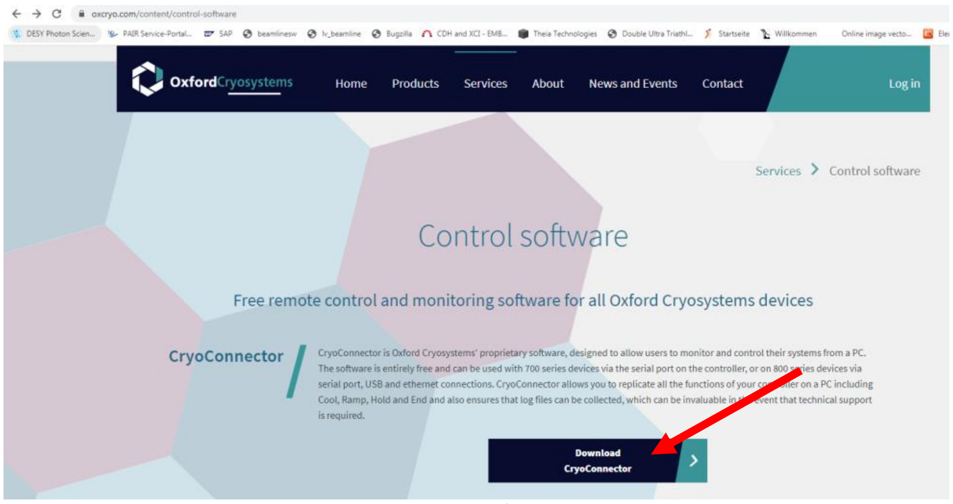
		Processor Architecture							
Operating System	Release Date	x86 (32-bit)	x64 (64-bit)	PPC	ARM	MIPSII	MIPSIV	SH4	Comments
Windows*	2017-08-30	2.12.28	2.12.28	-	-	-	-	-	WHQL Certified. Includes VCP and D2XX. Available as a setup executable Please read the Release Notes and Installation Guides.
Linux	-	-	-	-	-	-	-	-	All FTDI devices now supported in Ubuntu 11.10, kernel 3.0.0-19 Refer to TN-101 if you need a custom VCP VID/PID in Linux VCP drivers are integrated into the kernel.

Figure 1. USB Driver

Using this option you are sure you get the latest, nevertheless there is an already downloaded version in tineshare\Programs called CDM v2.12.28 WHQL Certified

For installing the driver, locate the unknown device under Start\Control Panel\Device Manager, right click on Update driver and choose the location where the file has been downloaded (and unzipped). Restart the PC probably is not necessary but good to do.

Download the latest version of cryoconnector from https://www.oxcryo.com/content/control-software



Control software

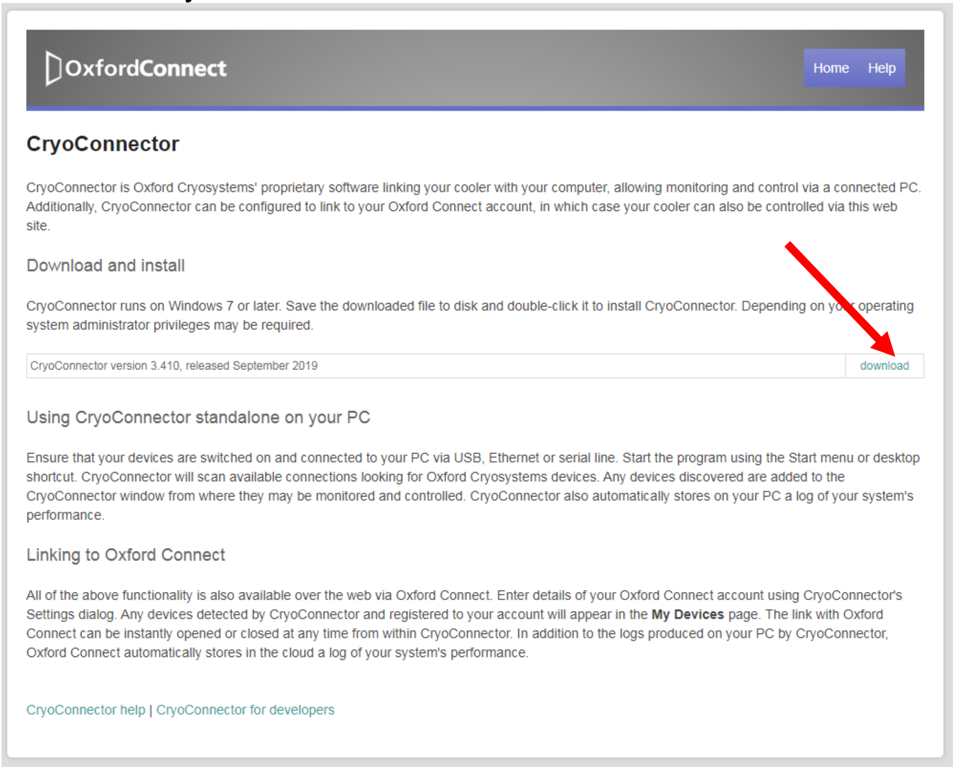
Free remote control and monitoring software for all Oxford Cryosystems devices

CryoConnector

CryoConnector is Oxford Cryosystems' proprietary software, designed to allow users to monitor and control their systems from a PC. The software is entirely free and can be used with 700 series devices via the serial port on the controller, or on 800 series devices via serial port, USB and ethernet connections. CryoConnector allows you to replicate all the functions of your controller on a PC including Cool, Ramp, Hold and End and also ensures that log files can be collected, which can be invaluable in the event that technical support is required.

Figure 2. Website Cryoconnector

Click on Download CryoConnector



CryoConnector

CryoConnector is Oxford Cryosystems' proprietary software linking your cooler with your computer, allowing monitoring and control via a connected PC. Additionally, CryoConnector can be configured to link to your Oxford Connect account, in which case your cooler can also be controlled via this web site.

Download and install

CryoConnector runs on Windows 7 or later. Save the downloaded file to disk and double-click it to install CryoConnector. Depending on your operating system administrator privileges may be required.

CryoConnector version 3.410, released September 2019	download

Using CryoConnector standalone on your PC

Ensure that your devices are switched on and connected to your PC via USB, Ethernet or serial line. Start the program using the Start menu or desktop shortcut. CryoConnector will scan available connections looking for Oxford Cryosystems devices. Any devices discovered are added to the CryoConnector window from where they may be monitored and controlled. CryoConnector also automatically stores on your PC a log of your system's performance.

Linking to Oxford Connect

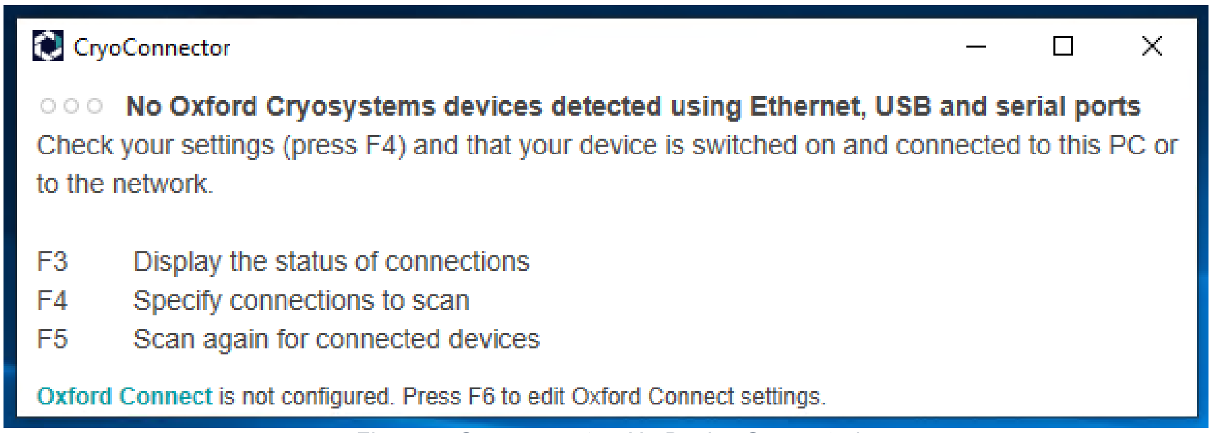
All of the above functionality is also available over the web via Oxford Connect. Enter details of your Oxford Connect account using CryoConnector's Settings dialog. Any devices detected by CryoConnector and registered to your account will appear in the **My Devices** page. The link with Oxford Connect can be instantly opened or closed at any time from within CryoConnector. In addition to the logs produced on your PC by CryoConnector, Oxford Connect automatically stores in the cloud a log of your system's performance.

CryoConnector help | CryoConnector for developers

Figure 3. Download Cryoconnector

Click on download.
Once downloaded, it can be installed.
Using the icon on the desktop the software can be launched.

This software will scan for any controller connected. In case of no controller detected, the software will report accordingly showing the message as shown in the figure below.



CryoConnector

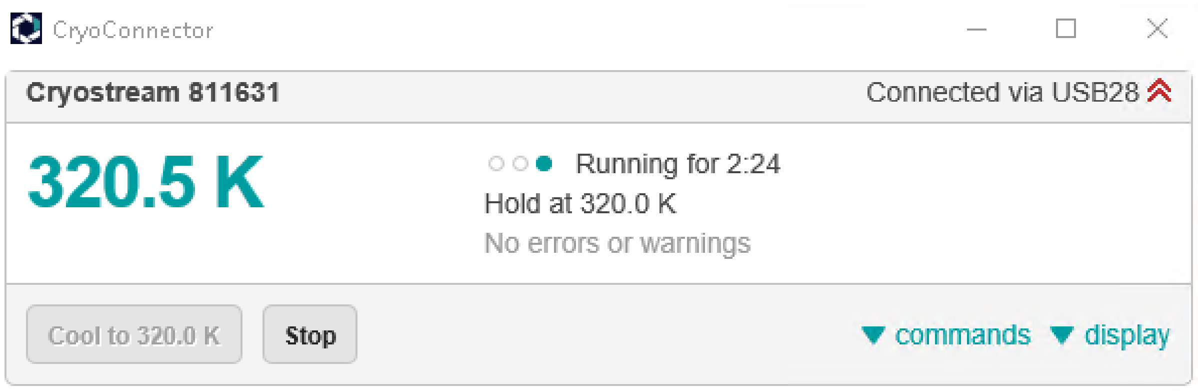
No Oxford Cryosystems devices detected using Ethernet, USB and serial ports
Check your settings (press F4) and that your device is switched on and connected to this PC or to the network.

F3	Display the status of connections
F4	Specify connections to scan
F5	Scan again for connected devices

Oxford Connect is not configured. Press F6 to edit Oxford Connect settings.

Figure 4. Cryoconnector: No Device Connected

In case of a controller detected, the data will be shown as in the figure below.



CryoConnector

Cryostream 811631 Connected via USB28

320.5 K

Running for 2:24
Hold at 320.0 K
No errors or warnings

Figure 5. Cryoconnector: Device Connected

CryoConnector allows the communication with the Cryostream 800 controllers by the exchange of XML files, whilst CryoConnector handles all communication with the device. During the installation of Cryoconnector the file paths are creatred. In our case there are two paths important for use

1. C:\Users\user\AppData\Roaming\CryoConnector -> Here we can find the files
   a. connections.xml. This is a read only file, it offers all the information related to the connection, like serial number of the device connected.
   b. Lastcommand.xml. This is a write only file, it is used to command the controller. We will use it to set the temperature. Cryoconnector will check for this file, in case there is a change it will execute the command.

2. C:\Users\user\AppData\Roaming\CryoConnector\[serial number] -> Here we can find the file
    a. status.xml. This is a read only file where Cryoconnector is updating all the values and units of the controller.

CHAPTER 3. Labview Software

3.1 Overview

The aim of this project is to create a Labview code that can read and write XML files and push and pull data from and to TINE servers. In addition to this a client will be developed in order to read and send data to the devices through TINE

3.2 Server Project

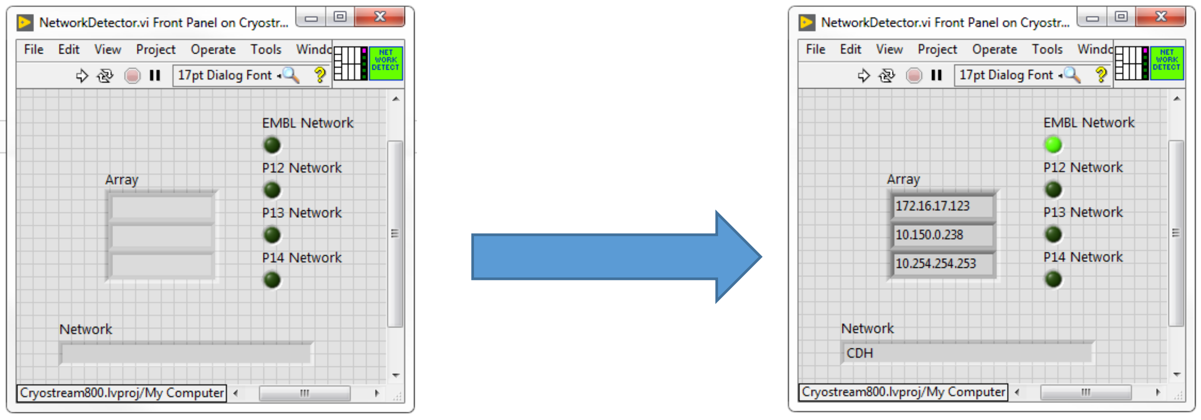
The server project is compound of the main VI called Cryostream800.vi and several sub-vi's used to dial with not only the XML files but also with the network.

3.2.1 Sub-functions (Sub-vi's)

Sub-function, also called sub-vi's have been programmed and used to have a clearer environment, making easier to follow the code.

3.2.1.1 Network Detector (NetworkDetector.vi)

As it was mentioned earlier in this document, EMBL has several cryostream controllers. Therefore, the idea is to have only one piece of code that can be used with each one of them without changing the code. Then, the first step is to detect in which network the device is connected in order to start the proper server.


NetworkDetector.vi Front Panel on Cryostr...
File Edit View Project Operate Tools Windo
17pt Dialog Font
EMBL Network
P12 Network
P13 Network
P14 Network
Array
Network
Cryostream800.lvproj/My Computer
NetworkDetector.vi Front Panel on Cryostr...
File Edit View Project Operate Tools Windo
17pt Dialog Font
EMBL Network
P12 Network
P13 Network
P14 Network
Cryostream800.lvproj/My Computer

Array
172.16.17.123
10.150.0.238
10.254.254.253

Network
CDH

Figure 6. Network Detector - GUI

When this VI is run, it will check all the Ethernet ports to detect to which network is connected to.

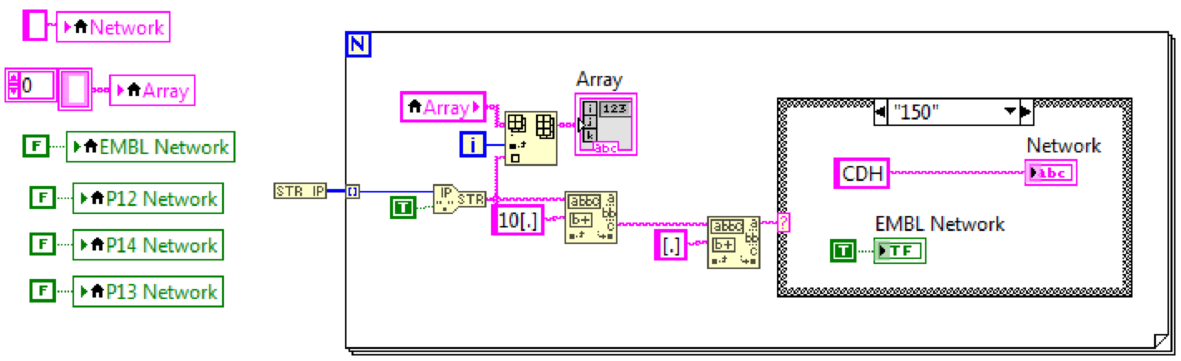


Figure 7. Network Detector - Block Diagram

The *FOR* loop is used in order to scan all the Ethernet ports available in the computer, otherwise it would take only the first one, that could happen to be one that is not connected. In every iteration of the loop each IP found is parsed and stored in an array.

The parsing is done by looking in the second field of the IP address (the string between the first two points), that means that when an IP like 10.14.10.19 is found, the second field is 14, then the P14 network is detected. **It is important to put the point between brackets, otherwise the parsing fails. The point is considered a special character and needs to be between brackets.**

This VI provides 5 outputs: 4 booleans corresponding to EMBL, P12, P13 and P14 and 1 string indicating network. In this case, when the EMBL network is detected corresponds to the CDH Cryostream controller.

3.2.1.1 Read Connection (ReadConnection.vi)

This VI is used to detect whether there is a controller connected to the PC or not.
It has an input parameter which is the XPath expression to be used in the XML file to find the node.
The task of this VI is to readout the data from the XML file called connections.xml. This VI provides important information:

1. Id: serial number of the controller connected. This is a unique number.
    a. ConnectionStatus: if an Id is found in the xml file, the connection status is set to “Active”. When there is no id in the file, this output is set to “No connection”.
    b. Path: when an Id is found, the path where the status.xml is located can be also found in the file.
2. LastUpdated: timestamp of the last update of values. It allow us to detect whether the Cryoconnector program is running or not.

The connection.xml is written all the time by the Cryoconnector software. To avoid conflicts between Cryoconnector and Labview to access the file, the file is copied to a different location (the folder where the executable is stored) and the parsing is done to the copy.

As it can be seen in the figure below, the steps would be:

1. Copy the xml file
2. Open the xml file
3. Get the data using the XPath expression */LIST_OF_CONNECTIONS*
4. Parsing the output to get the id and the timestamp of the last update
5. If the id is found, parsing the output to get the path of the status.xml.
6. Close all open nodes. The way we stablish the sequence as usual is using the error wire, then the idea is keep in mind this: First In, Last Out (FILO). The first node that you open, must be the last to be closed.

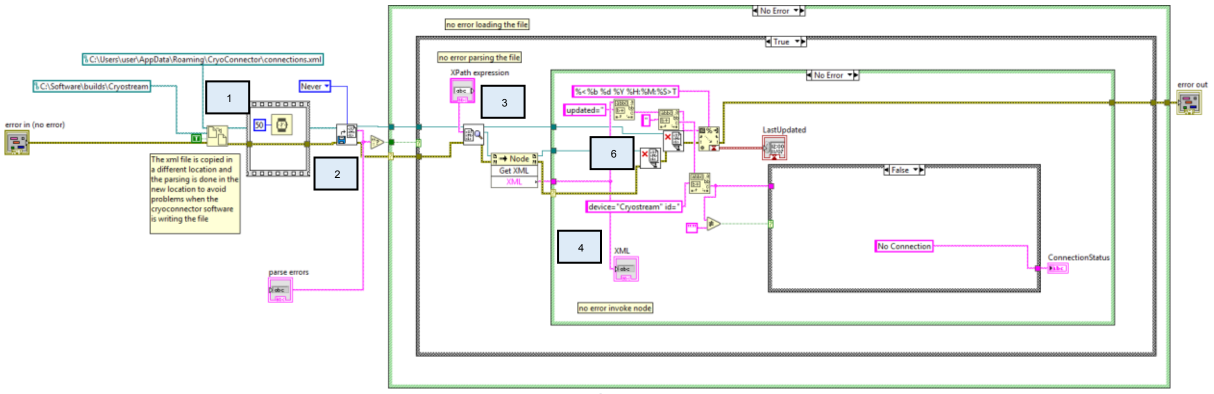

The xml file is copied in a different location and the parsing is done in the new location to avoid problems when the cryoconnector software is writing the file

Figure 8. Read Connection - steps

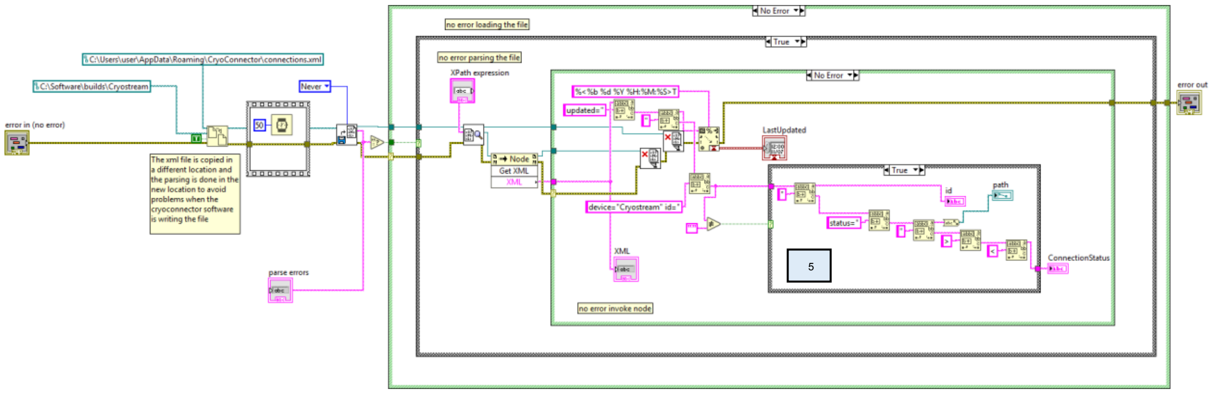

The xml file is copied in a different location and the parsing is done in the new location to avoid problems when the cryoconnector software is writing the file

Figure 9. Read Connection – Active Connection

In case there is an error loading the file, the node is closed.

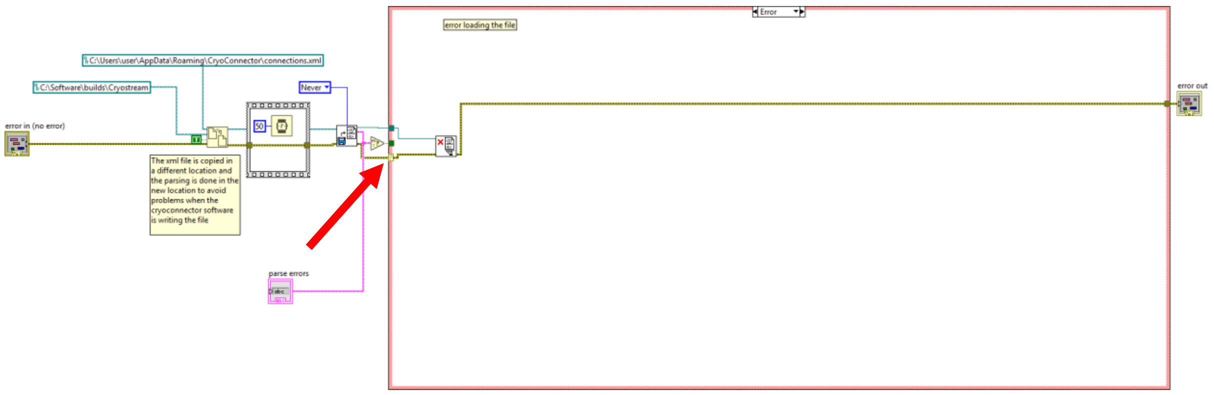

Error
error loading the file
C:\Users\user\AppData\Roaming\CryoConnector\connections.xml
C:\Software\builds\Cryostream
Never
error in (no error)
The xml file is copied in a different location and the parsing is done in the new location to avoid problems when the cryoconnector software is writing the file
parse errors
error out

Figure 10. Read Connection - Loading File Error

The same happens when the error occurs when getting the data from the node. It is very important to close the nodes. Otherwise there will be an important leak of memory.

There is another aspect to be taken into account. In order to be sure that the parsing is done properly, we check that the error string output of the open XML file VI is empty. This output only gets values when there is an error. In case of an error during parsing, the node is closed.

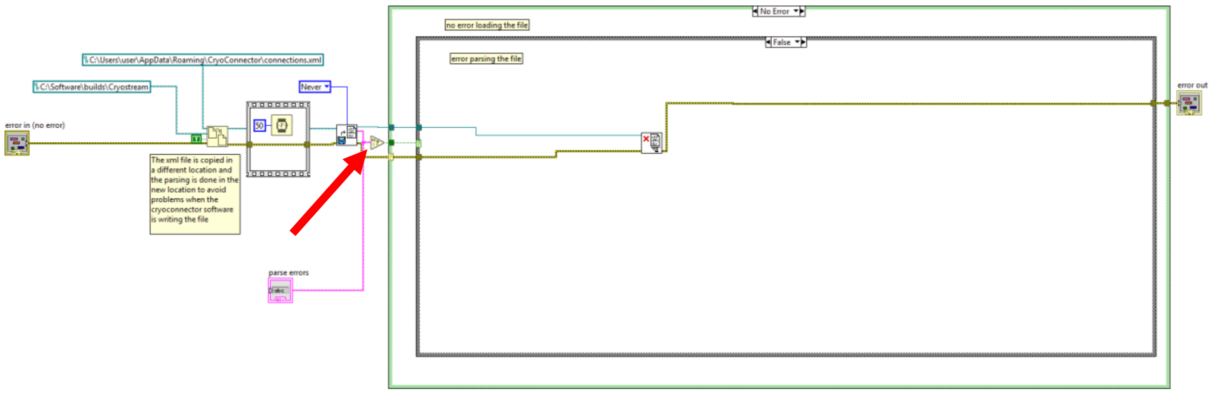

No Error
no error loading the file
False
error parsing the file
C:\Users\user\AppData\Roaming\CryoConnector\connections.xml
C:\Software\builds\Cryostream
Never
error in (no error)
The xml file is copied in a different location and the parsing is done in the new location to avoid problems when the cryoconnector software is writing the file
parse errors
error out

Figure 11. Read Connection - Parsing Error

It is important to close the nodes not only when they are used but also when there are errors.
In the figure below the FILO is explained.

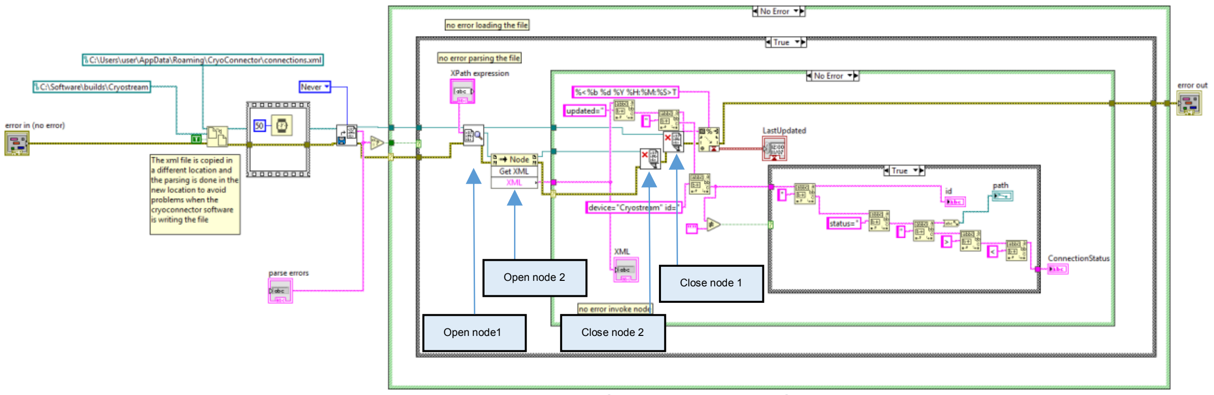
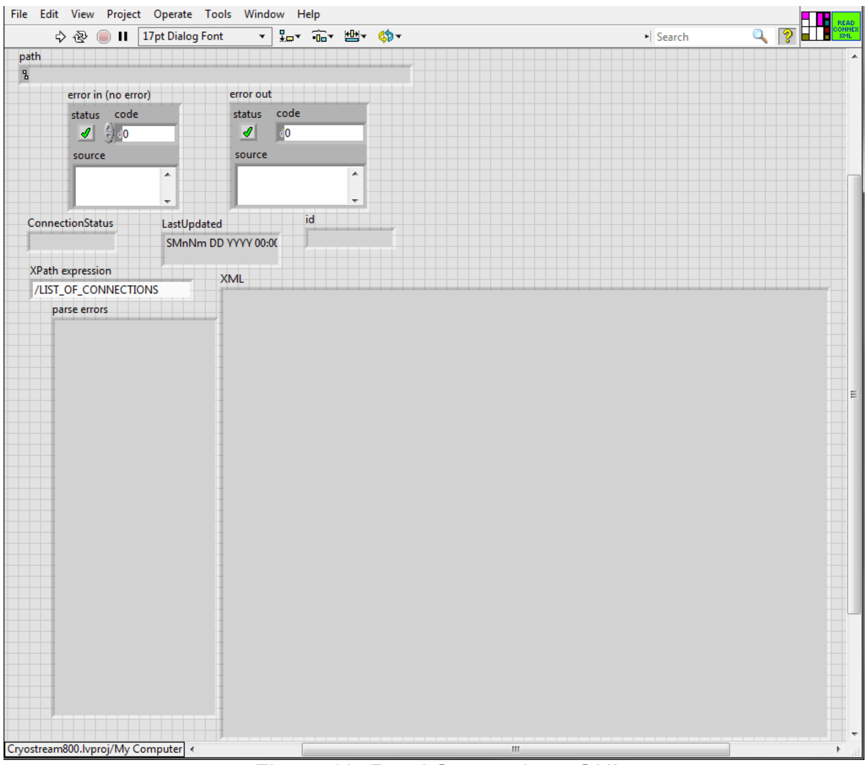

No Error
no error loading the file
True
no error parsing the file
XPath expression
No Error
%<%b %d %Y %H:%M:%S>T
updated="
LastUpdated
True
device="Cryostream" id="
id
path
status="
ConnectionStatus
XML
no error invoke node
Open node 2
Open node1
Close node 1
Close node 2
C:\Users\user\AppData\Roaming\CryoConnector\connections.xml
C:\Software\builds\Cryostream
Never
error in (no error)
The xml file is copied in a different location and the parsing is done in the new location to avoid problems when the cryoconnector software is writing the file
parse errors
error out

Figure 12. Read Connection - FILO

The GUI of this VI is shown below.



path

error in (no error): status ✓ code 0 source

error out: status ✓ code 0 source

ConnectionStatus

LastUpdated: SMnNm DD YYYY 00:0(

id

XPath expression: /LIST_OF_CONNECTIONS

parse errors

XML

Figure 13. Read Connection - GUI

As it can be seen, the VI has two input terminals (controls) and 5 output terminals (indicators) and two internal indicators:

1. Controls:
    a. XPath (input terminal)
    b. Input error (input terminal)
2. Indicators:
    a. Connection status (output terminal)
    b. Id (output terminal)
    c. LastUpdated (output terminal)
    d. Path (output terminal)
    e. Error out (output terminal)
    f. Parse error (error indicator when loading a xml file)
    g. XML (content of the XML file)

An example of the node in the xml file could be these one:

```
<LIST_OF_CONNECTIONS updated="Mar 25 2020 12:33:58">
<CONNECTION                                              server="401"
properties="C:\Users\User\AppData\Roaming\CryoConnector\coolers\Cryostream.
xml" status="C:\Users\User\AppData\Roaming\CryoConnector\811631\status.xml"
alias="" device="Cryostream" port="USB28" id="811631">Active</CONNECTION>
</LIST_OF_CONNECTIONS>
```

3.2.1.1 Read Info (ReadInfo.vi)

This VI is used to read the phase status of the controller, therefore it is possible to know whether the controller is cooling down, ramping up or holding the sample temperature that has been set. This information is extracted from the status.xml but from a different node.
This VI must worked in conjunction with the readconnection.vi, because readconnection.vi is providing the path where the status.xml file is located. The procedure is quite similar.
As it can be seen in the figure below, the steps would be:

1. Copy the xml file. After copying the file there is a small delay of 50ms.
2. Open the xml file
3. Get the data using the XPath expression */COOLER_STATUS/LIST_OF_INFO/INFO*
4. Parsing the output to get the info and the values in form of two arrays. These arrays obtained using the FOR loop.
5. Close all open nodes. The sequence of closing nodes is very important, otherwise the VI will fail. The way we stablish the sequence as usual is using the error wire, then the idea is keep in mind this: First In, Last Out (FILO). The first node that you open, must be the last to be closed.

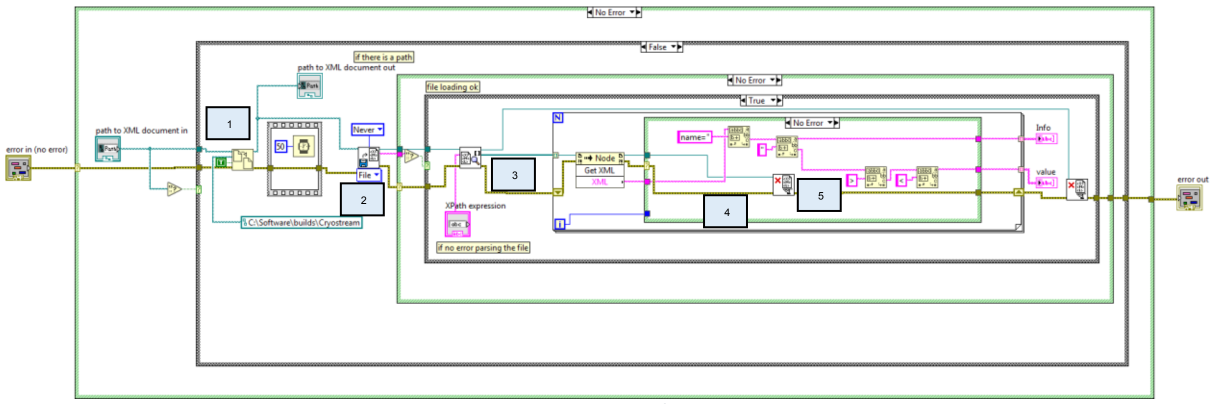


Figure 14. Read Info - steps

This VI has a compulsory parameter that is the path to the status.xml. In case that the path is empty, nothing will happen inside the VI.

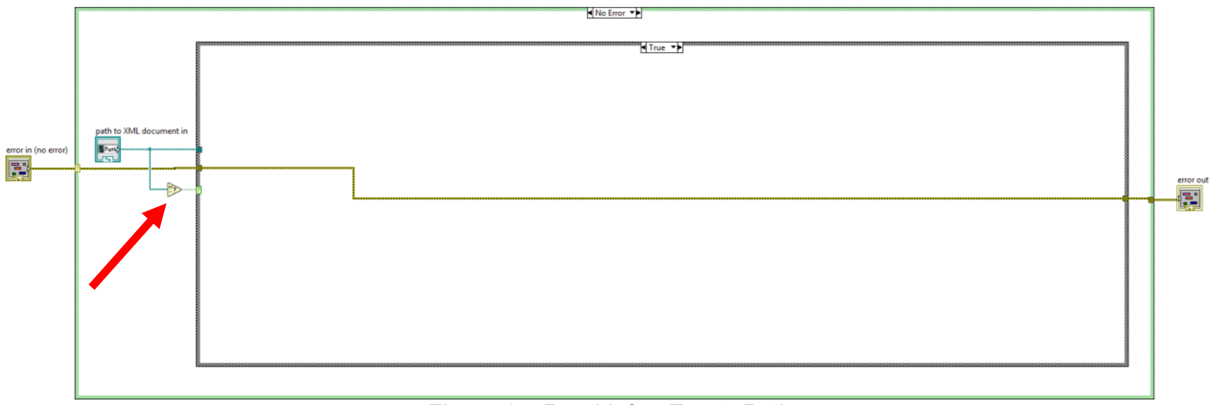


Figure 15. Read Info - Empty Path

In case of errors, we proceed exactly the same manner than in the readconnection.vi. In case there is an error loading the file, the node is closed.

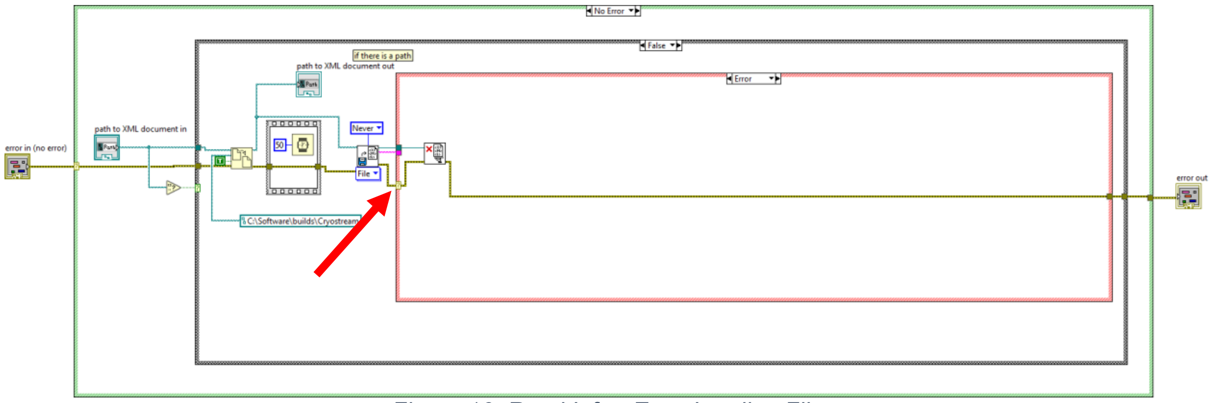


Figure 16. Read Info - Error Loading File

There is another aspect to be taken into account. In order to be sure that the parsing is done properly, we check that the error string output of the open XML file VI is empty. This output only gets values when there is an error.

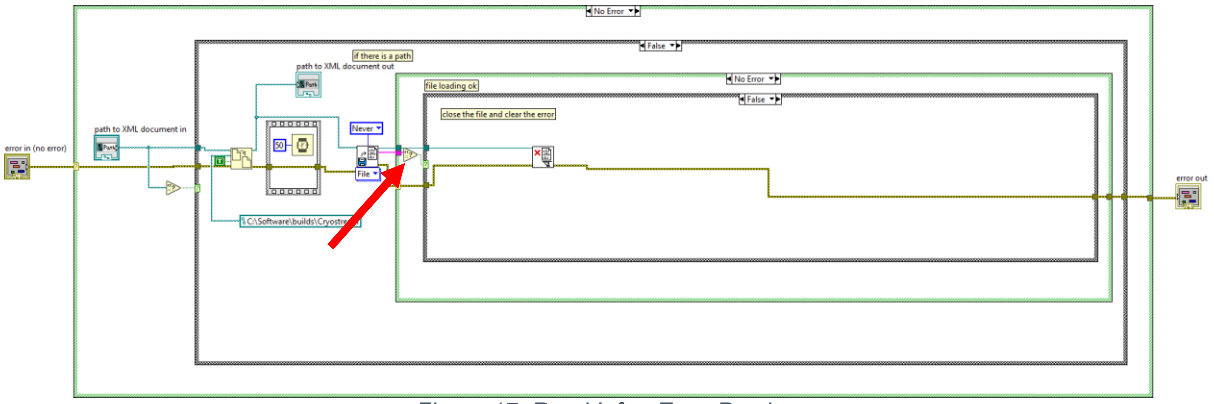


Figure 17. Read Info - Error Parsing

It is important to close the nodes not only when they are used but also when there are errors.
In the figure below the FILO is explained (again).

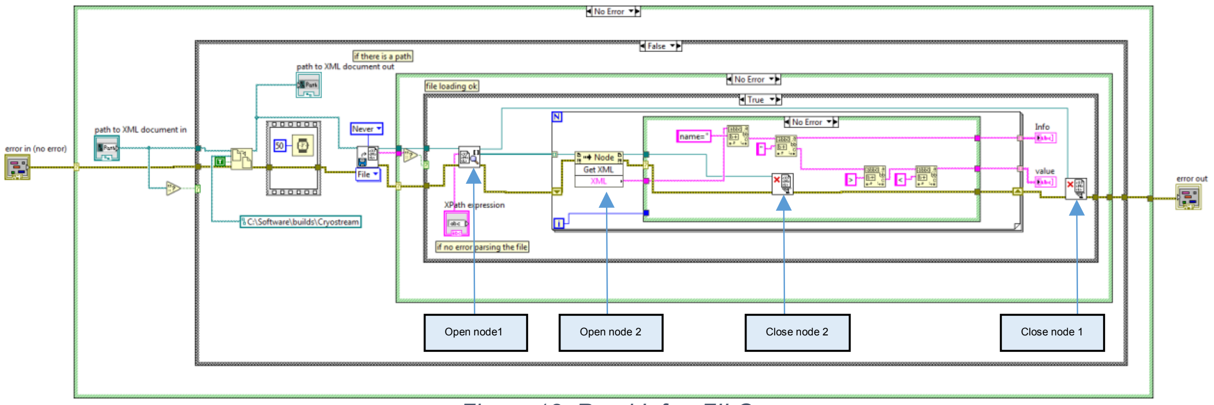
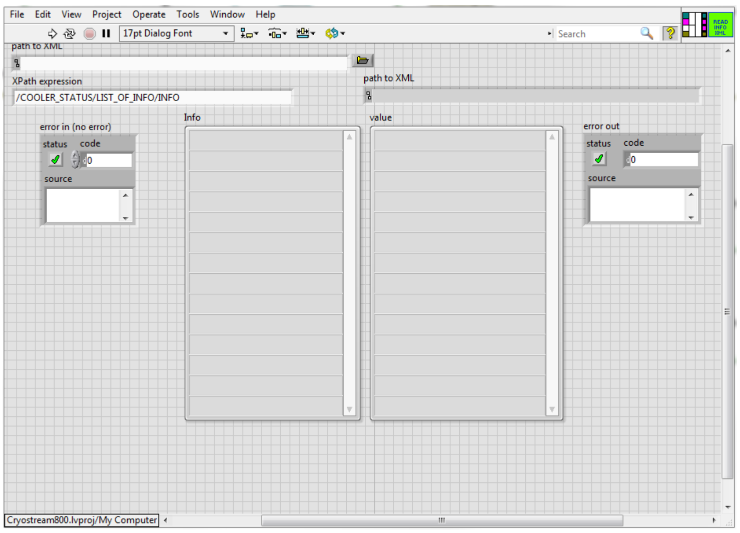


Figure 18. Read Info - FILO

The GUI is shown in the figure below



XPath expression
/COOLER_STATUS/LIST_OF_INFO/INFO

Figure 19. Read Info - GUI

As it can be seen, the VI has three input terminals (controls) and 4 output terminals (indicators):

1. Controls:
    a. XPath (input terminal)
    b. Path (input terminal)
    c. Input error (input terminal)
2. Indicators:
    a. Path (output terminal)
    b. Info (output terminal)
    c. values (output terminal)
    d. Error out (output terminal)

An example of the node in the xml file could be these one:

```
  <LIST_OF_INFO>
    <INFO name="Device name">Cryostream 7111023</INFO>
    <INFO name="Device alias">My Cryostream </INFO>
    <INFO name="Device status">Running for 3.0 days</INFO>
    <INFO name="Phase status">Ramp at 360.0 K/hour to 100.0 K</INFO>
    <INFO name="Alarm status">No Alarm</INFO>
    <INFO name="Time stamp">Jul 03 2012 17:47:11</INFO>
   </LIST_OF_INFO>
</COOLER_STATUS>
```

Although we are retrieving all the information, the current value used is the Phase status to determine what the controller is doing at this exact moment..

3.2.1.1 Read Status (ReadStatus.vi)

This VI is used to read all the important data of the controller. This information is extracted from the status.xml but from a different node.

This VI must worked in conjunction with the readconnection.vi and readinfo.vi because they provides the path where the status.xml file is located and also generates the copy of the XML.

The procedure is quite similar.
As it can be seen in the figure below, the steps would be:

1. Open the xml file (this is the copy)
2. Get the data using the XPath expression */COOLER_STATUS/LIST_OF_PROPERTIES/PROPERTY*
3. Parsing the output to get the info and the values in form of three arrays (properties, values and units). These arrays are obtained using the FOR loop.
4. Close all open nodes. The sequence of closing nodes is very important, otherwise the VI will fail. The way we stablish the sequence as usual is using the error wire, then the idea is keep in mind this: First In, Last Out (FILO). The first node that you open, must be the last to be closed.

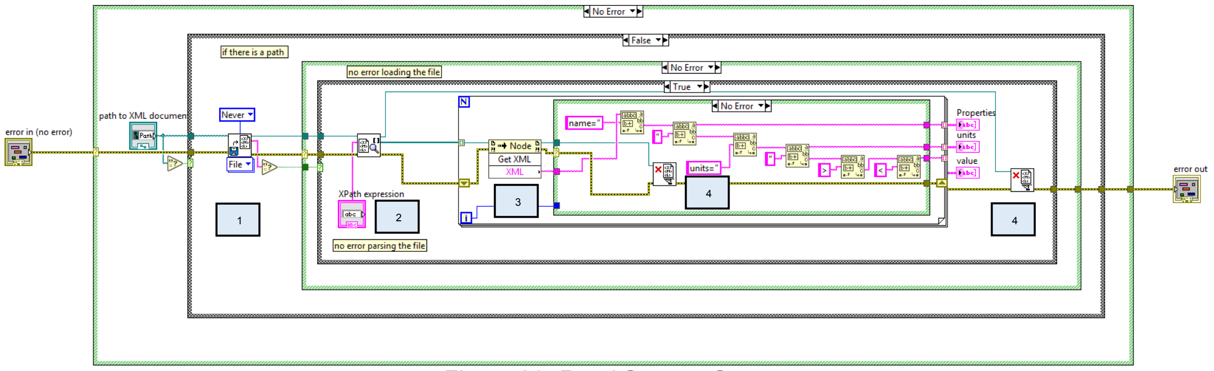


Figure 20. Read Status - Steps

If an error is coming from the previous VI, this VI will not be executed.

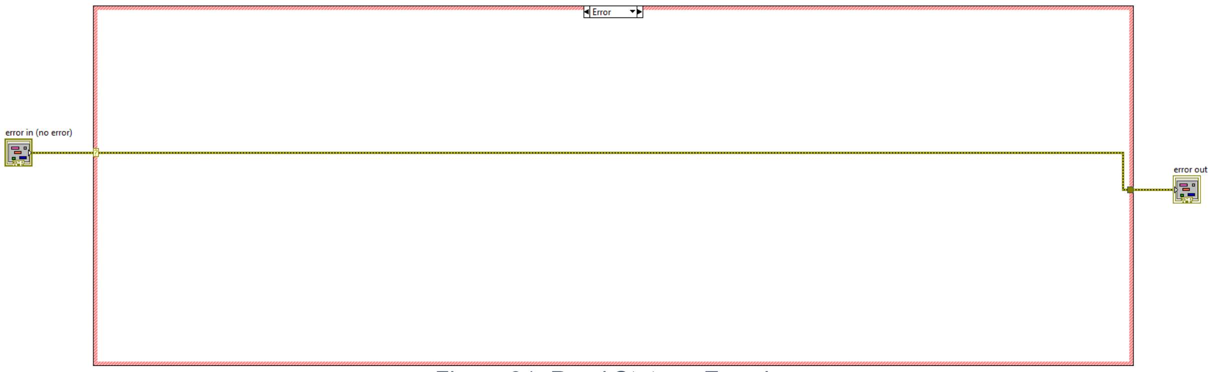


Figure 21. Read Status - Error In

This VI has a compulsory parameter that is the path to the status.xml. In case that the path is empty, nothing will happen inside the VI.

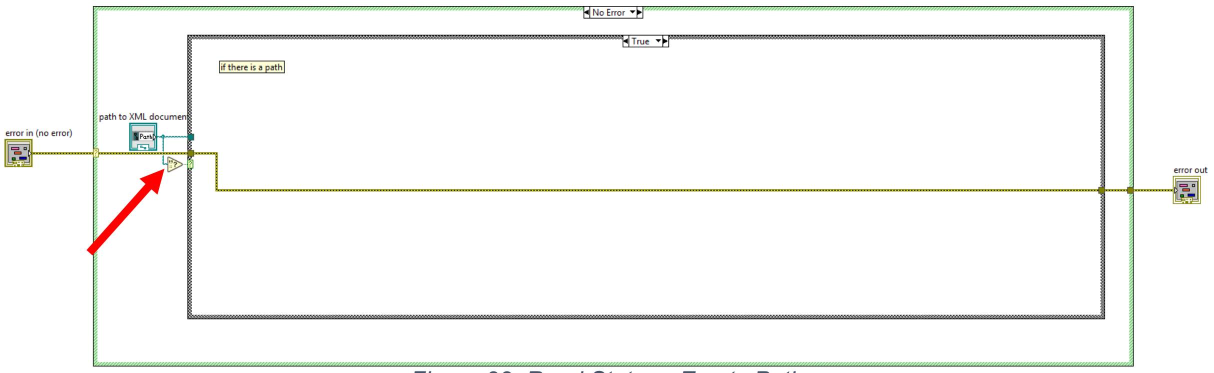


Figure 22. Read Status - Empty Path

In case of errors, we proceed exactly the same manner than in the readconnection.vi and readinfo.vi In case there is an error loading the file, the node is closed.

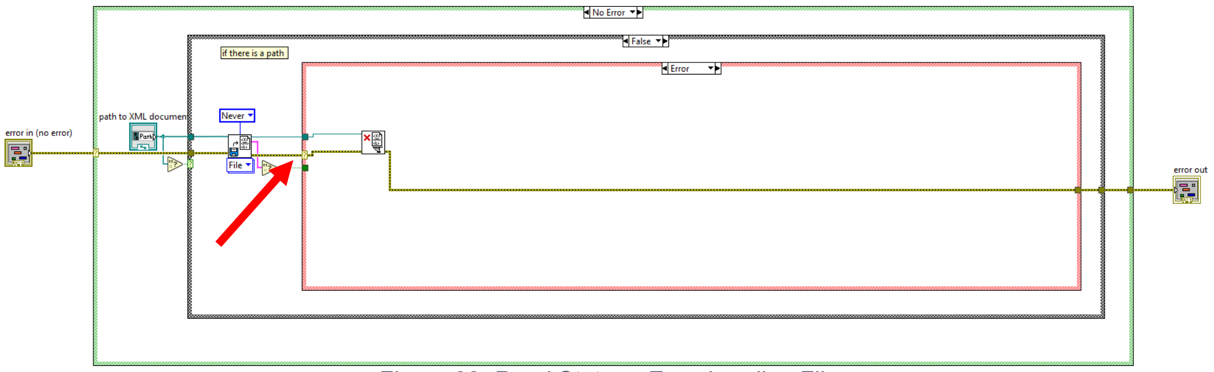


Figure 23. Read Status - Error Loading File

There is another aspect to be taken into account. In order to be sure that the parsing is done properly, we check that the error string output of the open XML file VI is empty. This output only gets values when there is an error.

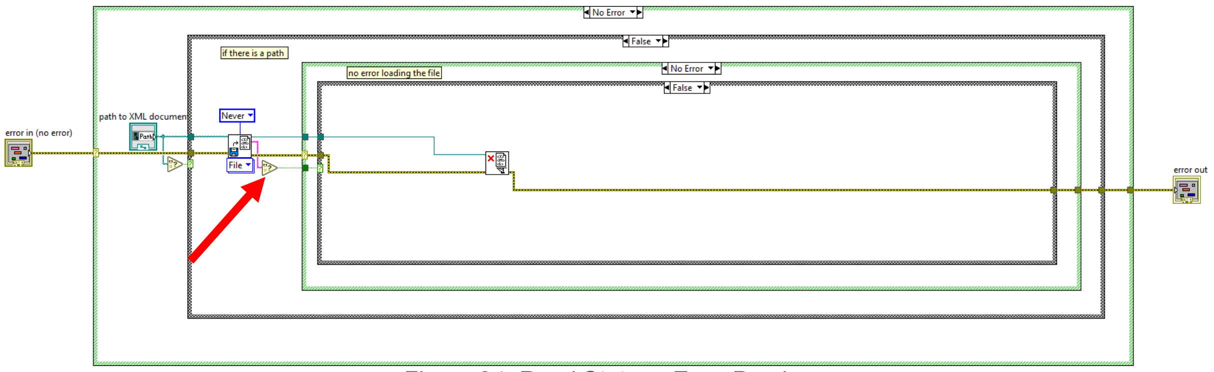


Figure 24. Read Status - Error Parsing

It is important to close the nodes not only when they are used but also when there are errors.

In the figure below the FILO is explained.

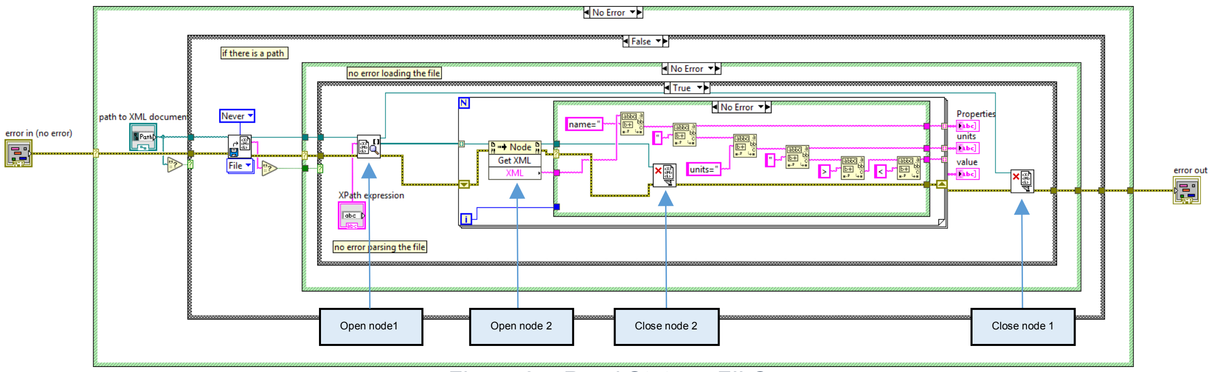


Figure 25. Read Status - FILO

The GUI is shown in the figure below

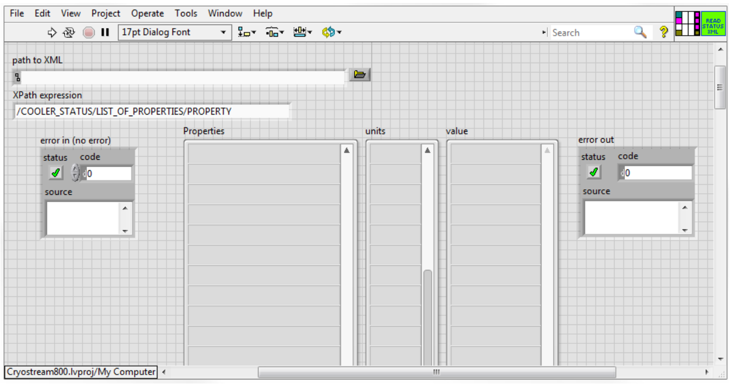


Figure 26. Read Status - GUI

As it can be seen, the VI has three input terminals (controls) and 4 output terminals (indicators):

3. Controls:
    a. XPath (input terminal)
    b. Path (input terminal)
    c. Input error (input terminal)
4. Indicators:
    a. Properties array (output terminal)
    b. Units array (output terminal)
    c. Values array (output terminal)
    d. Error out (output terminal)

An example of the node in the xml file could be these one (with every upgrade of the firmware the list is increased!):

```
<?xml version="1.0" encoding="UTF-8"?>
<COOLER_STATUS>
  <COOLER name="Cryostream"></COOLER>
  <COOLER_PROPERTIES>Cryostream.xml</COOLER_PROPERTIES>
```

```
<LIST_OF_PROPERTIES>
  <PROPERTY name="Set temp" units="K">101.6</PROPERTY>
  <PROPERTY name="Sample temp" units="K">101.5</PROPERTY>
  <PROPERTY name="Run mode" units="">3</PROPERTY>
  <PROPERTY name="Phase id" units="">0</PROPERTY>
  <PROPERTY name="Ramp rate" units="K/hour">360</PROPERTY>
  <PROPERTY name="Evap temp" units="K">77.5</PROPERTY>
  <PROPERTY name="Suct temp" units="K">290.5</PROPERTY>
  <PROPERTY name="Gas flow" units="l/min">10.0</PROPERTY>
  <PROPERTY name="Gas heat" units="%">26</PROPERTY>
  <PROPERTY name="Evap heat" units="%">54</PROPERTY>
  <PROPERTY name="Average suct heat" units="%">12</PROPERTY>
  <PROPERTY name="Back pressure" units="mbar">150</PROPERTY>
  <PROPERTY name="Alarm code" units="">0</PROPERTY>
  <PROPERTY name="Run time" units="hour">4321</PROPERTY>
  <PROPERTY name="Firmware version" units="">46</PROPERTY>
  <PROPERTY name="Average gas heat" units="%">32</PROPERTY>
  <PROPERTY name="Suct heat" units="">1</PROPERTY>
  <PROPERTY name="Total hours" units="hour">23102</PROPERTY>
</LIST_OF_PROPERTIES>
```

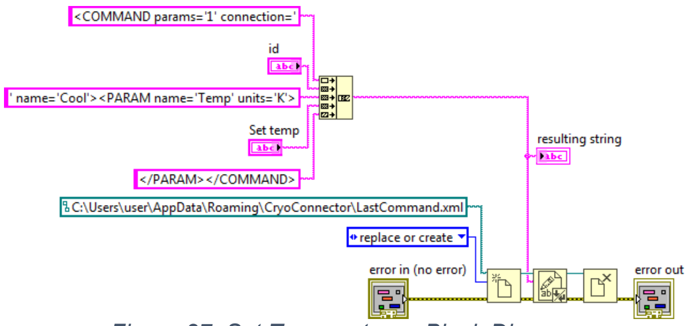
Although we are retrieving all the information, the properties used are:

1. Sample temp
2. Evap temp
3. Suct temp
4. Gas heat
5. Evap heat
6. Average suct heat
7. FC Gas flow
8. FC Back pressure
9. FC Valve opening

These properties will be later used as TINE properties that the client will readout from the server.

3.2.1.1 Set Temperature (Cool.vi)

This VI is used to write the xml file with the command that we want to execute. Therefore, the xml is created by adding the ID of the controller and the desired temperature. The Cryoconnector will recognize that a new XML is there and the command will be executed. Although there are more commands available, in our application we only set the sample temperature.
As it was mentioned before, the path to the location of the xml files is constant (set by the installation of the Cryoconnector program.



<COMMAND params='1' connection='
id
' name='Cool'><PARAM name='Temp' units='K'>
Set temp
</PARAM></COMMAND>
C:\Users\user\AppData\Roaming\CryoConnector\LastCommand.xml
replace or create
resulting string
error in (no error)
error out

Figure 27. Set Temperature - Block Diagram

The GUI is shown below.

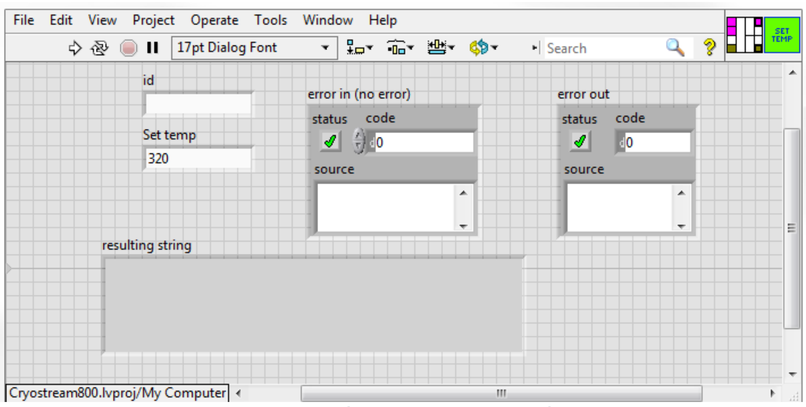


Figure 28. Set Temperature - GUI

As it can be seen, the VI has three input controls terminals (controls) and 1 output terminals (indicators):

1. Controls:
    a. Id (input terminal)
    b. Set Temp in Kelvin degrees (input terminal)
    c. Input error (input terminal)
2. Indicators:
    a. Resulting string (it shows the content of the created xml file)
    b. Error out (output terminal)

An example of the xml to set the temperature could be this where the connection id (81187) and the temperature (320) are the inputs parameters of the VI:

```
<COMMAND name="Cool" connection="811187" params="1">
      <PARAM name="Temp" units="K">320</PARAM>
</COMMAND>
```

3.2.2 Server

This VI is considered the main VI of the application because it is considered the server. This VI is in charge of:

1. Detect the network.
2. Initiate the server according to the network detected
3. Check the connection to detect whether there is a controller connected.
4. Restart the Cryoconnector software if the last update is older than 5 minutes (this can be disabled)
5. Read the information of the controller (info and value arrays).
6. Read the status of the controller (properties, values and units arrays).
7. Extract values from arrays and push the data into the server.
8. Set the temperature locally (commands thread).
9. Accept requests from the client (clients thread). The client can only set the sample temperature.

This VI is compound of 3 threads:

1. Main thread is in charge of stablishing the sequence of calls of sub-vi's, gathering all the information, parsing the arrays where needed and pushing the data into the server. The timing is not crucial, therefore the loop is executed every second (997ms).

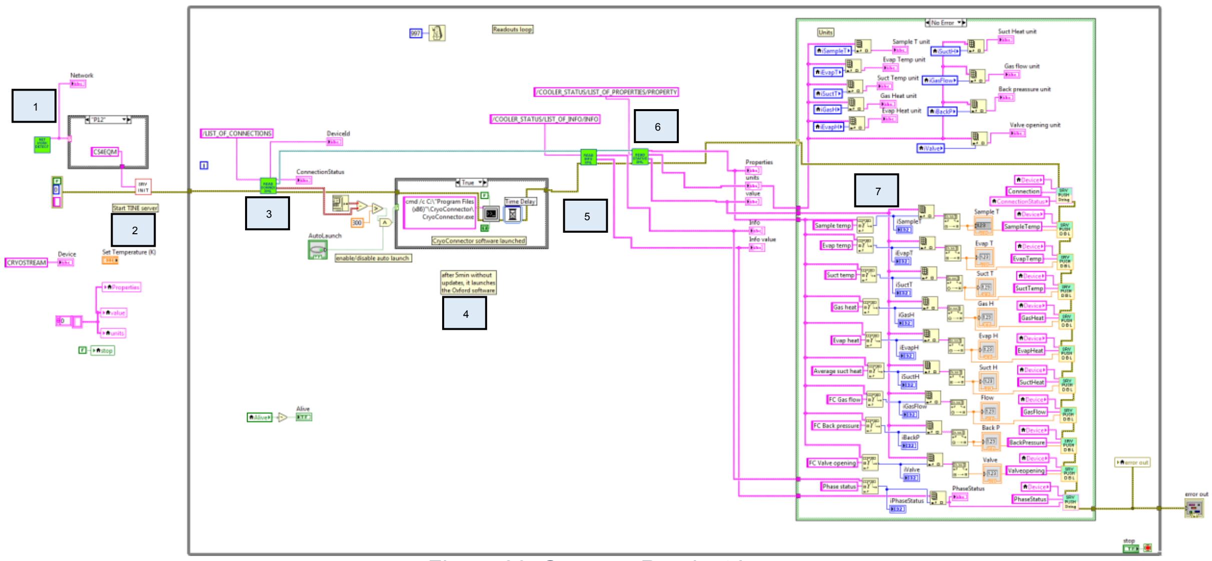


Figure 29. Server – Readout Loop

2. The commands thread is in charge of executing the commands coming from the server locally. Only the temperature can be set locally.

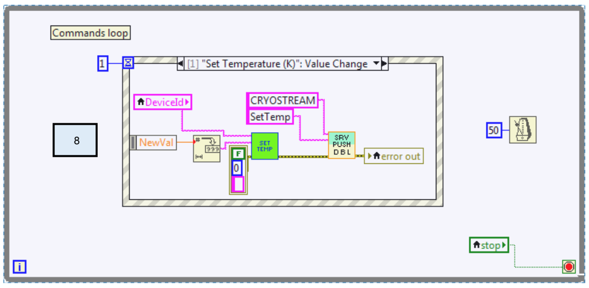


Figure 30. Server - Command Loop

3. Clients thread is waiting for commands coming from clients. Only the temperature can be set by the clients. In order to execute the command, first the data has to be pulled from the server and after the execution of the command the data is pushed to the server.

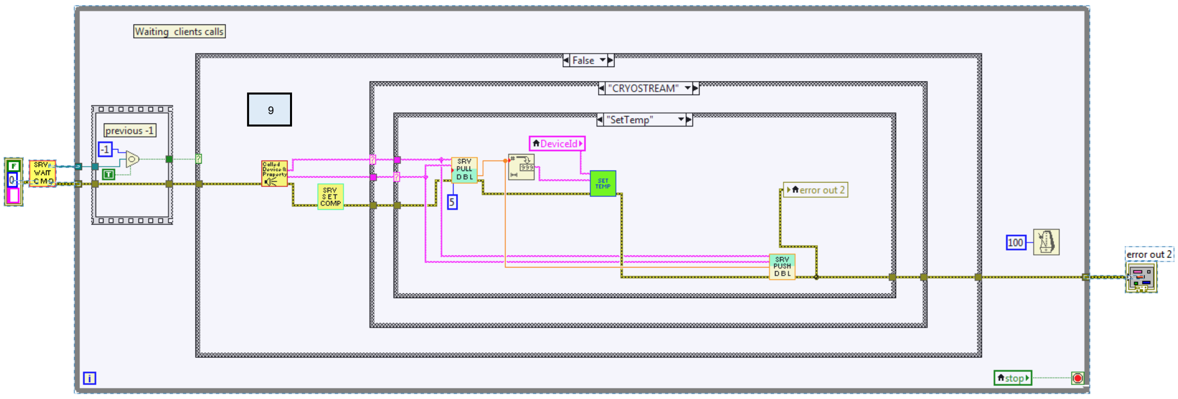


Figure 31. Server - Clients Loop

Thi GUI is presenting all the values and units that are considered most important at a first glance, taking that priority the sample temperature. The GUI is devided in 4 parts:

1. Program alive. The green indicator is blinking when everything is right.
2. Connection status. This indicator shows ‘No connection’ when the controller is off or ‘Active’ when the controller is on.
3. Status of the controller. These indicators show back pressure, flow, evaporation temperature, suction temperature, suction humidity, gas humidity, evaporation humidity, sample temperature and valve aperture.
4. This is the only controller available in the GUI, to allow the user to set the sample temperature from the server locally.

The GUI for "No Connection" and "Active" are shown below:

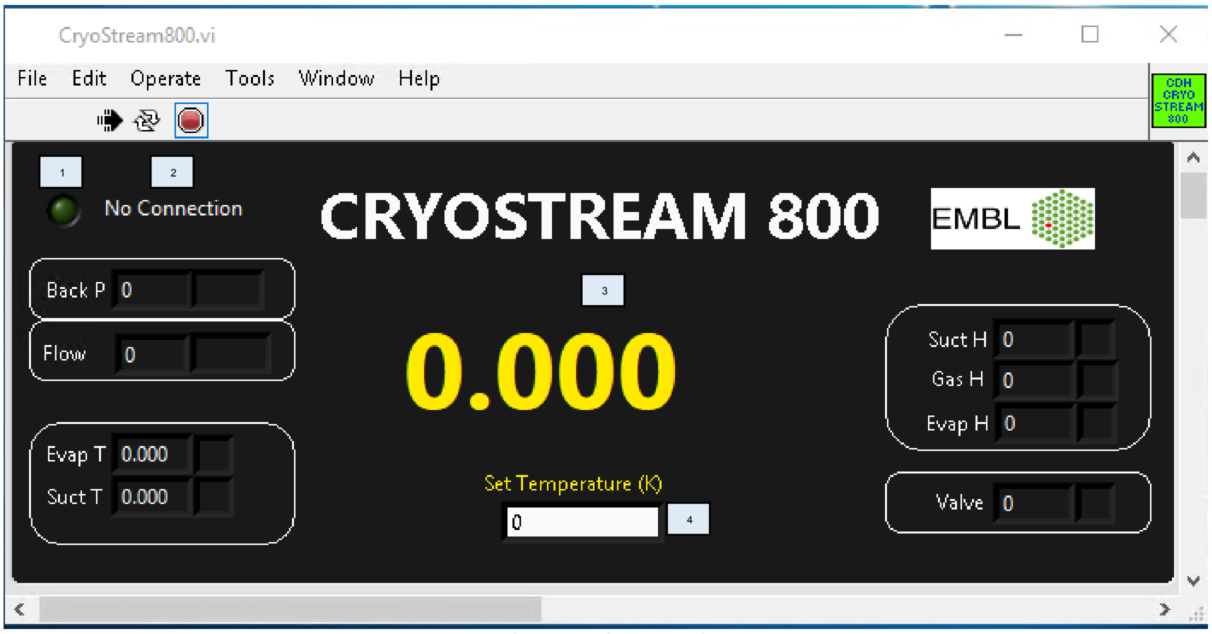


Figure 32. Server - GUI (No Connection)

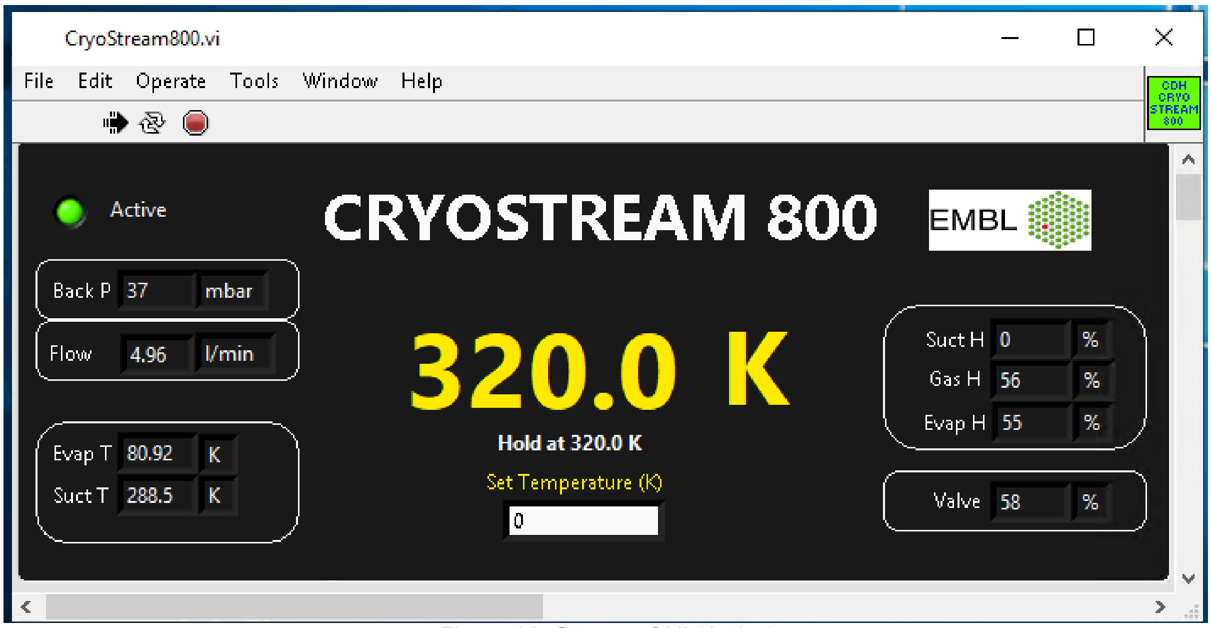


Figure 33. Server - GUI (Active)

3.3 Client

The client is customized for each one of the locations where cryostreams are used because each one is talking to a different server.

This VI is considered the main client of the application. All the communication between client and server is done via TINE.
This VI is in charge of:

1. Pull all data from the server. Depending to which server is talking to, the context and server variables will change:

Context	Server
P13	P13Cryostream
P14	P14Cryostream
SPC	CDHCryostream

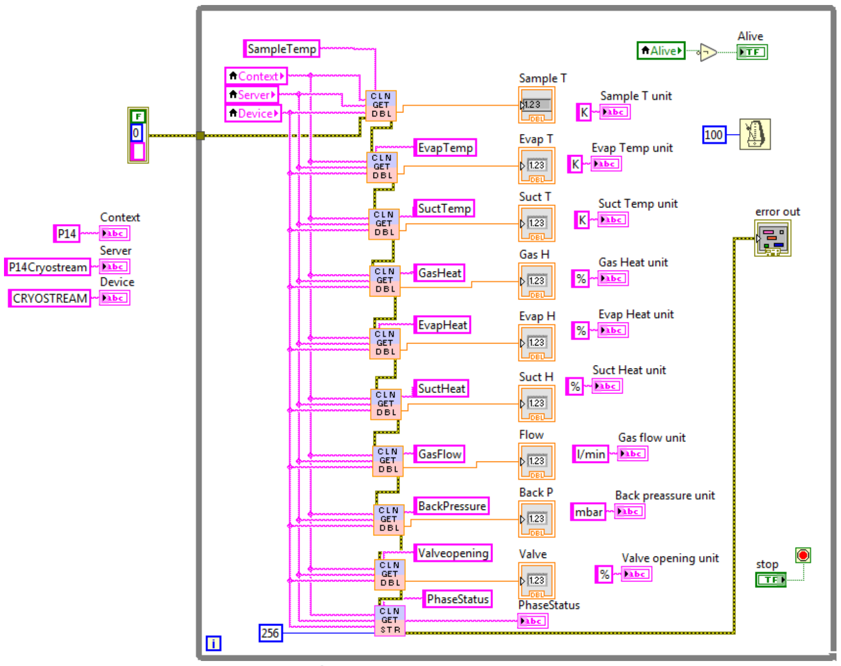


Figure 34. Client – Readout Loop

2. Put the new sample temperature into the server.

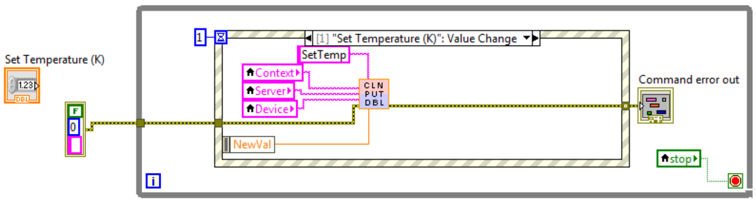


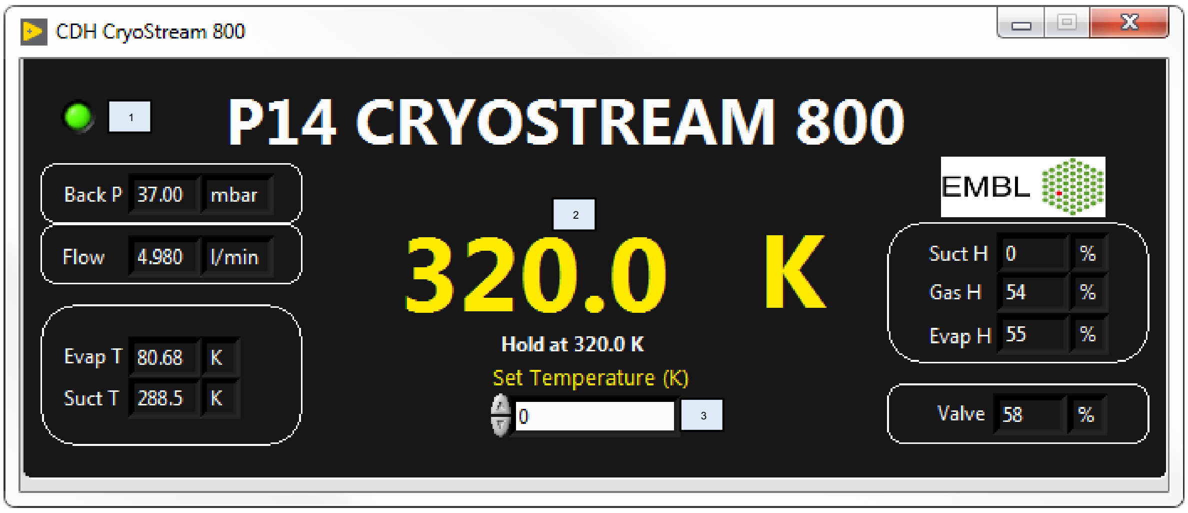
Figure 35. Client - Command Loop

This GUI is identical to the server's GUI, presenting all the values and units that are considered most important at a first glance, taking that priority the sample temperature. The GUI is divided in 4 parts:

1. Program alive. The green indicator is blinking when everything is right.
2. Status of the controller. These indicators show back pressure, flow, evaporation temperature, suction temperature, suction humidity, gas humidity, evaporation humidity, sample temperature and valve aperture.
3. This is the only control available in the GUI, to allow the user to set the sample temperature from the server locally. The user introduces the new value and by

hitting the enter key the command is executed.

The GUI is shown below:



CDH CryoStream 800

P14 CRYOSTREAM 800

Back P 37.00 mbar

Flow 4.980 l/min

Evap T 80.68 K

Suct T 288.5 K

320.0 K

Hold at 320.0 K

Set Temperature (K) 0

Suct H 0 %

Gas H 54 %

Evap H 55 %

Valve 58 %

Figure 36. Client - GUI