MEET-CINCH

(Project Number: 754 972)

DELIVERABLE D7.1

RoboLab NXG Template

Lead Beneficiary: UiO

Due date: 31/12/2020 Released on: DD/MM/YYYY

Authors:	Jon Petter Omtvedt and Olga N. Salina	
For the Lead Beneficiary	Reviewed by Work package Leader	Approved by Coordinator
Jon Petter Omtvedt	Jon Petter Omtvedt	Clemens Walter
Jon Petter Omtvedt	Jon Petter Omtvedt	signature

Start date of project: 01/06/2017 Duration: 36 Months
Project Coordinator: Prof Clemens Walther
Project Coordinator Organisation: LUH VERSION: 1.1

Project co-funded by the European Commission under the Euratom Research and Training Programme on Nuclear Energy within the Horizon 2020 Programme

Dissemination Level		
PU	Public	X
RE	Restricted to a group specified by the Beneficiaries of the MEET-CINCH project	
CO	Confidential, only for Beneficiaries of the MEET-CINCH project	

Version control table

Version number	Date of issue	Author(s)	Brief description of changes made
1.0	20/Dec/2019	J. P. Omtvedt	Draft document written
1.1	6/1/2020	Jana Peroutkova	MST check
1.2	6/Jan/2020	J. P. Omtvedt	Accepted changes
1.3	21/Jan/2020	CW	Minor corrections

Project information

Project full title:	A Modular European Education and Training Concept In Nuclear and RadioCHemistry
Acronym:	MEET-CINCH
Funding scheme:	Coordination Action
ECGA number:	754972
Programme and call	H2020 EURATOM, Euratom Fission 2016-2017, NFRP-12
Coordinator:	Clemens Walther
EC Project Officer:	Katerina Ptackova
Start date – End date:	01/06/17 – 31/05/20 i.e. 36 months
Coordinator contact:	+49 511 762 3312, walther@irs.uni-hannover.de
Administrative contact:	+420 245 008 599, cinch@evalion.cz
Online contacts:	http://www.cinch-project.eu/

"This project has received funding from the Euratom research and training programme 2014-2018 under grant agreement No 754 972."

The Research Council of Norway

The Norwegian part (WP7) of this project is funded by the Norwegian Research Council under grant agreement No 300310

EXECUTIVE SUMMARY

A functional LabView NXG program using the System Link cloud service was written for the Neutron Activation of Silver RoboLab system. All the old functionality was retained and the LabView code will serve as a template for upgrading other RoboLab systems to the new LabView NXG code and web-functionality.

Using the System Link service adds stability and ease of use to the RoboLabs. It also enables added functionality and possibilities. However, the programming is slightly more complex than with the old method (which is no longer supported by National Instruments). The template developed in this Deliverable should make upgrading substantially more easy.

Integrating video streaming into the RoboLab web-page is not covered by this Deliverable, but should be straightforward using commercial streaming services that can be embedded in HTML pages. Examples on how to do this will be demonstrated in the code which will be delivered in Deliverable 7.2 (due in May 2020).

CONTENT

1 INTRODUCTION

Using modern computer controlled hardware, remote operation of an experiment in a real radiochemical laboratory is possible, even on a limited budget. This was demonstrated in a pilot project ran at the University of Oslo, where two such "RoboLab" remote controlled exercises were implemented and used in their basic nuclear chemistry course in 2006-2009.

Such remote operated exercises can be used as preparation training before entering a real lab, or as a stand-alone exercise. Such exercises will never be a substitute for real hands-on training, but it is a useful addition to traditional training. It will be of particular value to institutions without laboratories classified for radioactive work, or just laboratories with very limited equipment and radioactive sources, as the learners can nevertheless do some practical work.

In CINCH-II, two sites were established (at IRS and UiO), which provide remote access to a total of five remote controlled exercises, based on the UiO RoboLab concept. A sixth exercise was piloted at UiO, but never brought to full readiness.

The technical foundation of the RoboLabs is the LabView programming environment developed by National Instruments. Part of the LabView package provides the possibilities to quickly and easily develop web-based "virtual instruments", which is what is used to run the RoboLabs. This technology goes back to before 2004 and is now obsolete and outdated. It will not be supported by National Instruments in the future. This endangers the future usage of the RoboLabs, as they cannot be migrated to future LabView versions and current operating systems like Windows 10 by the Microsoft Corporation. Thus, there is dire need for upgrading the RoboLabs to the new NXG version of LabView that also supports web-based application in a modern and completely new way. This entails a complete rewrite of the code for all the RoboLabs.

The major part of Work Package 7 – Sustaining RoboLab and NucWik – is dedicated to updating the LabView code used for running the RoboLabs. This work is divided in two tasks, each with their own Deliverable:

Task/Delivarable 7.1: Develop and test a new "template" for RoboLabs running on the new NXG version of LabView. This will provide a skeleton which can be used for individual RoboLab exercises and should make porting the existing RoboLabs to the new version easy.

Task/Deliverable 7.2: Upgrade the existing RoboLab exercises (at UiO) to the new LabView technology, using the template developed in step 1.

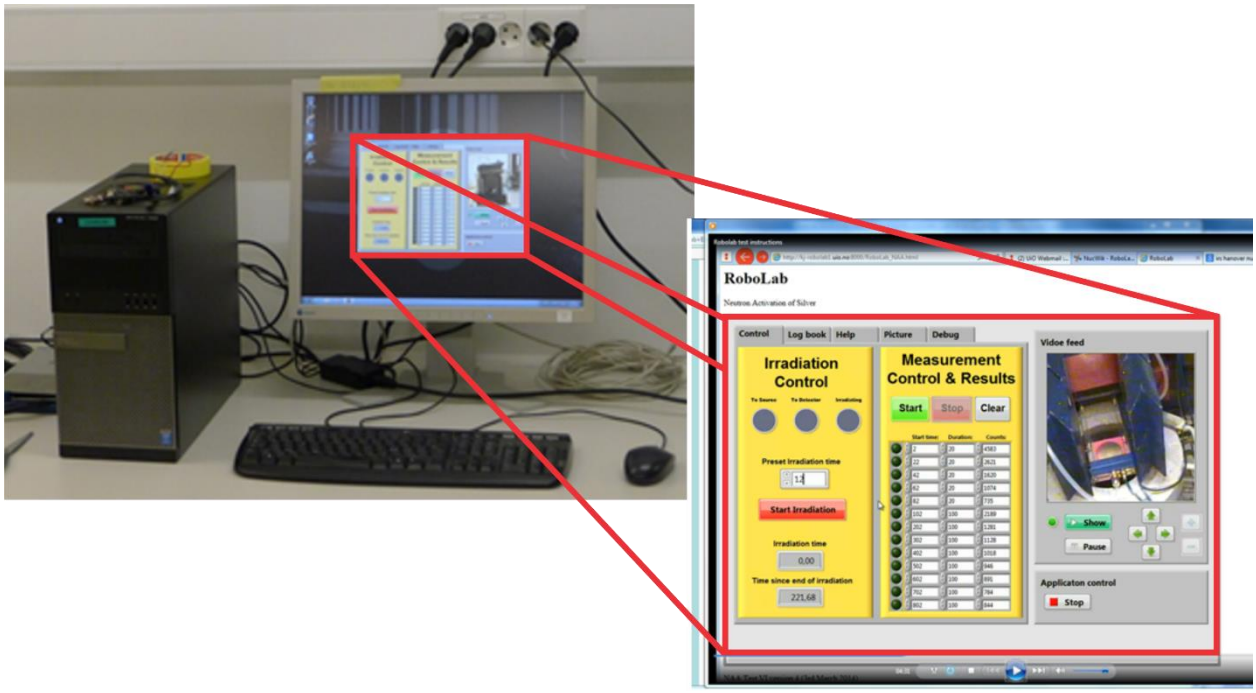
This document reports on the work done for the first task and describes how the new LabView NXG programming environment was successfully used to rewrite and modernise the remote access method used for the RoboLab exercises. The report is divided into three Chapters: The first describes how the NXG technology for creating virtual instruments (VIs) running on a web-page works. The second describes how the NXG based RoboLabs will be managed and used by teachers (none programmers). The third Chapter describes programming details only of interest to LabView programmers, but documents the methods developed that can be used as a template for upgrading or writing new RoboLabs.

2 LABVIEW NXG TECHNOLOGY FOR WEB-PAGE VIRTUAL INSTRUMENTS

Abbreviations

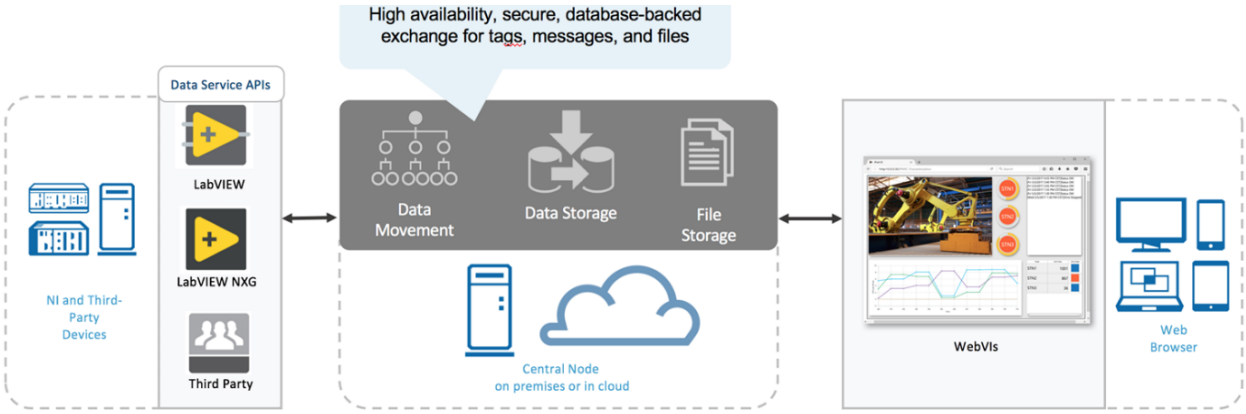
- VI = Virtual Instrument
- LabView = Graphical Programming Language from NI
- NXG = The new LabView platform from NI
- NI = National Instruments

The existing remote controlled laboratory, nicknamed "RoboLab", was originally developed in 2004 at UiO. It used the then new and innovative LabView tools for publishing virtual instruments (VI) with very little effort to a web-page. This was achieved by built in technology in LabView that generated an interactive image of the VI on a web-page provided that a run-time LabView module was installed in the web browser (on the client machine). The web-page on the client communicated directly, through the plug-in, with the actual VI running and controlling the RoboLab hardware on the host computer in the physical laboratory. The VI image on the client was an exact mirror of the host's VI, as illustrated on the image below (picture is from the RoboLab Neutron Activation of Silver PC host).



Communication is directly between the client and the host. This works well once properly set up, but the client user has to ensure that the correct version of the LabView plug-in is installed and that the local firewall is configured properly. These two requirements turned out to become an increasing hurdle as new versions of LabView were introduced on a yearly basis, at the end in both 32- and 64-bit versions, and network security gradually was increased at most organisations. As RoboLabs was developed both in Oslo and in Hanover and not always running on the same version of LabView (i.e. different plug-ins were needed), it became more and more difficult to set up for the teacher at the client end. Furthermore, not all browsers were supported by LabView (Microsoft's Internet Explorer was the recommended browser). All in all, it became impractical to run the original system as intended and we reverted to run the RoboLabs through Remote Desktop where the teacher accessed the host computer directly. Clearly, the old system was not sustainable and support and development were also discontinued by National Instrument for future versions of LabView. The new LabView system launched in 2017, named NXG, did not incorporate the old web VI mirroring technology.

Based on all this it is necessary to rewrite the RoboLab software to provide remote access to teachers and students in a different way. The aim was to ensure that teachers and students should be able to access and use the RoboLabs with a minimum of hassle, preferentially not needing to install or download any kind of special software/plug-ins or similar at all. The obvious choice was to use the new web-VI functionality provided by National Instruments for NXG. This method of exporting VIs to web-pages is based on ordinary HTML and Java-script programming, thus needs no plug-ins or similar. Basically, LabView NXG compiles a program developed in LabView to Java and embeds it in a HTML document. Communication between host and client now runs via a cloud service, which avoids providing access rights through the host's firewall and safety regulations. National Instruments provides a free of charge cloud service called "System Link". The new RoboLab template was developed using this service and the NXG Web-VI compiler. It should be noted that with this system the software running on the host and client are two different programs, this is of no concern to the client user (student/teacher), but adds some complexity to the programming of the RoboLabs. A bonus is that it also adds higher flexibility, e.g. by allowing added functionality on the server that is invisible to the client. An illustration, copied from the National Instruments LabView site, of how the new Web-VI system is interlinked with the host is shown below.



The cloud service – System Link – can contain data (variables) which can be updated and read by both the host and client. Each variable is "tagged" by a unique name used by the host and client(s) to access the data. The data can only be of simple types (boolean, integer, real, string); Arrays or other more complex datatypes are not supported.

With the tagged variables there is no functionally to ensure that an updated value uploaded to the cloud by e.g. the host will automatically be passed on to the client. It is the programmer's responsibility to either check for updated values on a regular basis ("polling") or incorporate some sort of handshake protocol to discover and download updated data values when they are available.

Fortunately, the System Link cloud service provided by National Instruments anticipated such needs and added another type of service: Messaging. With this functionality there is no (tagged) variable in the cloud service, instead the host and client subscribe to one or more message channel(s). For a given message channel any message published to it will be received by all subscribers to that particular channel. Messages can be whatever the programmer defines, e.g. it can flag that (tagged) data have been updated with new values; it can be a command generated by

the user; it can be information that certain conditions have been reached by the laboratory hardware (like end of measurement when a preset time has been reached).

Basically, the combination of tagged variables and the message channels provides a versatile tool for passing data back and forth between the host and client(s). The disadvantage compared to the old VI-mirror technology is that this communication now must explicitly be programmed as part of the software.

The programmer has to keep in mind bandwidth limitations of the link between the host, cloud and client(s). This naturally depends on where the host and users are located. Testing of bandwidth with the System Link cloud service is still going on, but so far it is clear that a significant lag can be encountered (in the seconds range). Furthermore, transfer of large amounts of data is not very rapid. This must be anticipated when writing the VIs on the host and client. This can be especially troublesome when presenting a "live" spectrum, i.e. a spectrum during data acquisition where a large number of channels will change during measurement. "Smart" updating were only channels with changed counts are transmitted or similar might be required to provide a reasonably "live" feeling to spectrum acquisition.

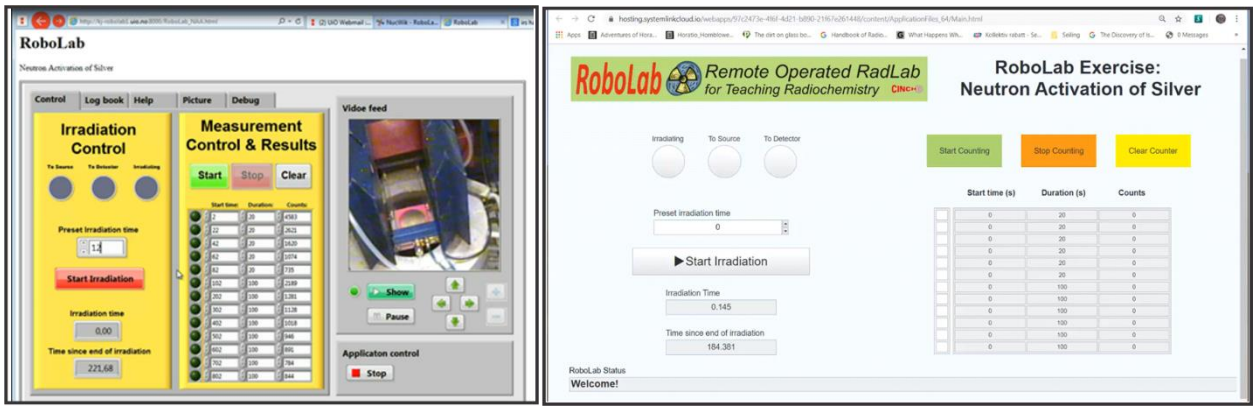
For the same reason any transfer of video or high quality pictures is out of the question through the cloud service – it is not set up for handling such amounts of data. Therefore, video transfer of what is going on in the remote laboratory must be transferred with other methods. We plan to use video streaming services for this, which is available from a number of service providers. Many providers offer the possibility to embed such a video stream into a HTML webpage, which is how we will do since the NXG compiler produces HTML-code that can be edited after creating just for such purposes.

The System Link cloud service also offer storage of files with data. This can be useful for transferring measurement results, pictures or similar to the user. This functionality has not been investigated at the current stage.

3 HOW TEACHERS WILL USE NXG BASED ROBOLABS

The new NXG based RoboLab systems will not change much for the users (teachers and students). They will access the RoboLab as before from links provided on the NucWik web-site (which provides access to teaching material for nuclear and radiochemistry). The difference is that they no longer will need to install any plug-in or other proprietary software. Neither should there be any difficulties with firewalls and data access, although this has to be verified and tested once the systems are ready for testing (this will be done as part of Deliverable 7.2 due in May 2020).

The work planned and promised for MEET-CINCH (Deliverable 7.2) comprises NXG versions of the RoboLabs available from UiO. This will be the Neutron Activation of Silver and Gamma Absorption systems, delivered with the same functionality as the old versions. Most users will probably not notice any changes except for the slight change in appearance and layout. For the template developed for the Deliverable described here (D7.1) the interface of the old Neutron Activation of Silver system is already available. The old (to the left) and new (to the right) VI is shown below:



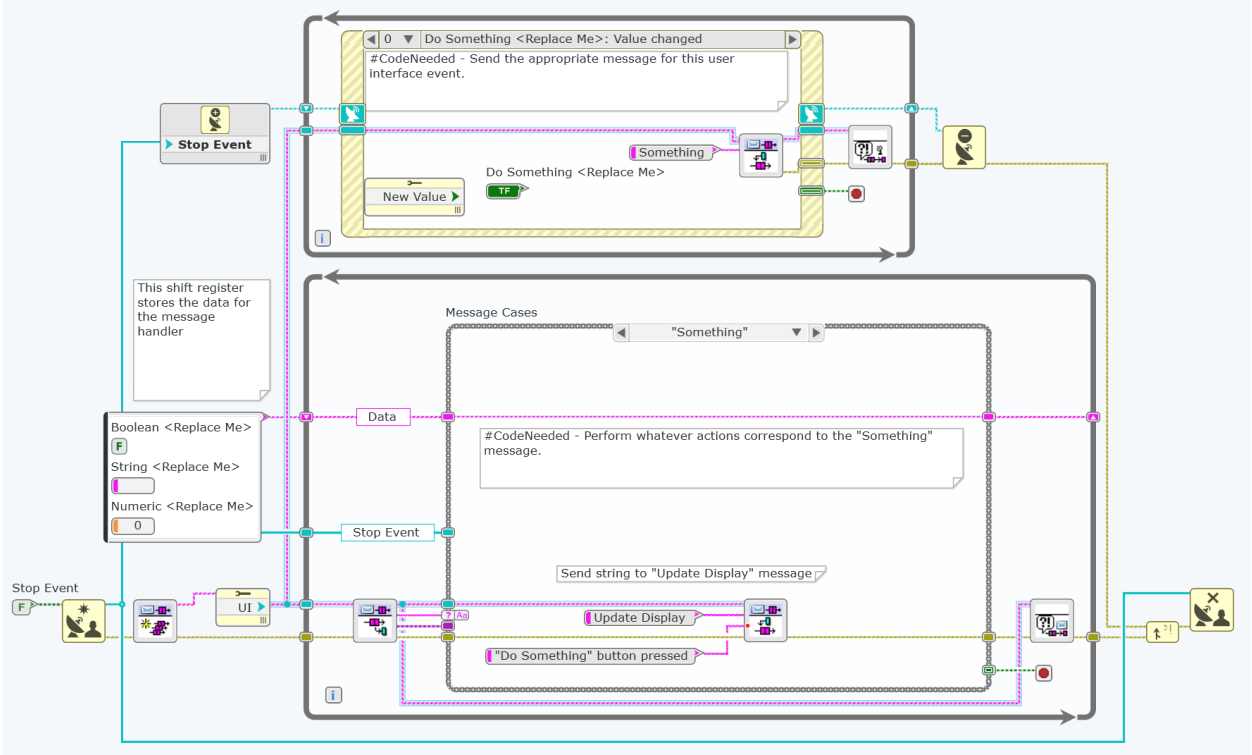
The embedded video streaming is not yet included in the new VI and not shown in the screen shot above.

A major improvement with the new system which will make them much easier to use is that they will run on any platform able to show HTML web-pages that run java-scripts. That should be almost any platform. This implies that with the NXG based RoboLabs they can now be viewed and run on e.g. tablets, smartphones and similar devices in addition to ordinary computers. The LabView Web-VI compiler cater for different form-factors based on what type of device the users have. It is therefore possible to have the “traditional” layout for ordinary computer screens but automatically adapt the layout to fit on smaller screens found on smart phones, etc.

Basically, the new NXG RoboLabs should be rather transparent to the users. However, the separate VI software running on the host and client provides the possibility to provide teachers with added control. It will be possible to provide client versions with added control features aimed at teachers monitoring and/or solving experiment issues during an exercise. This can be e.g. resetting the system upon lock-ups or access to controls not provided to the students (e.g. regenerating a separation column). Such features will not be added during the initial upgrade performed as part of the MEET-CINCH project (WP7), but can be explored in the future. It certainly adds new possibilities to how a RoboLab can be used in teaching.

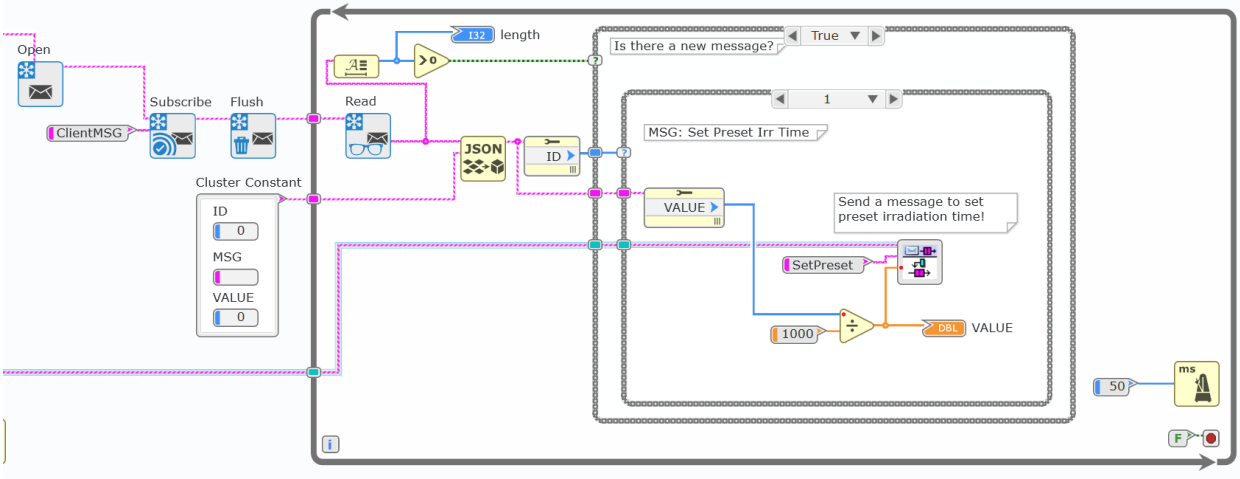
4 NXG PROGRAMMING DETAILS

The old UiO RoboLabs are based on the standard LabView Queued Message Handler which facilitates multiple sections of code running in parallel and sending messages between them. The generic layout is shown below.



Typically, the event handler in the upper while-loop is responding to user actions and generating messages that is passed on the second while-loop at the bottom. This loop handles the hardware and acts upon the messages generated by the user event loop. Additional while-loops can be added to e.g. handle measurement timing and system safety. This is how the old RoboLabs were constructed.

The Queued Message Handler method is fortunately very well suited to porting to the new NXG system. For the host, the basic structure can be reused. The old event handler loop is retained to respond to user input in the physical lab (where the host is located). Then an extra while-loop is added that responds to client user-events sent through the System Link cloud service:



Before the message stream from the System Link can be subscribed to (shown to the left in the picture), one needs to establish a link to the System Link cloud service:

This is done before any other code is executed. The web-address to System Link is provided together with an API key that provides access to "our" part of the cloud. The code and address is hardcoded into the compiled program and do not need to be provided by the RoboLab user, which simplifies operation. The configuration data obtained is used by all other System Link nodes, including the subscribe node already mentioned.

API keys are created and managed (access rights, etc.) in the System Link cloud service provided by NI. It is somewhat a little tricky to set up, but once working it has proved very stable. Client users will not need to either access or know about the API keys, as already mentioned they are hard coded into the client program. This makes using a client easy and straight forward. To avoid confusion, the programmer should include error code handling that gives the user clear instructions what is wrong and how to remedy the situation if contact with the cloud service is for whatever reason not established.

4.1 Responding to Client Commands

Once the configuration data are obtained and the message stream (called "ClientMSG" in the version shown above) has been subscribed to, messages from the client will be continuously processed as they arrive in the while-loop shown. These messages are generated by the client in response to user input. E.g. if the user clicks on the "Start Counting" button, a corresponding message is sent to the host through the System Link. When received, it will generate a corresponding message for distribution by the Queued Message handler. This message will be equal to what is generated by the Event Handler while-loop when the "Start Counting" button on the physical host-computer is pushed. From then on, the processing is exactly as before and no changes to the code are needed.

As can be seen in the System Link while-loop that receives messages (previous page), we have encoded our messages to contain a message ID number, a string and a value. We use the ID number to identify different types of messages, a unique ID is associated with each type of message. For example, ID number = 1 is used for setting present irradiation time. LabView provides a standard way of "packing" the cluster consisting our message, using the notation defined as a UTF-8 JavaScript Object Notation (JSON) string. This ensures that data always will be interpreted identically regardless of platform.

For each type of message (ID number) a case is programmed to handle that particular message. Since the action for each command usually is identical to how local commands are handled, it is relatively straightforward to respond to commands from the client: Identify the message/case number and respond similarly to the corresponding local event.

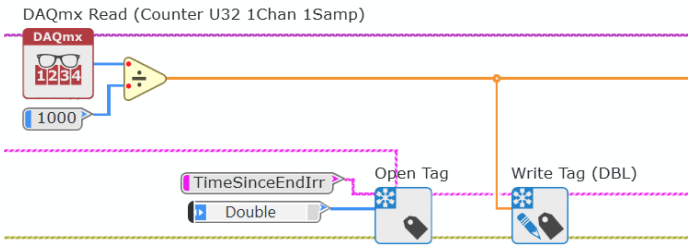
It should be noted that there seems to be a time-out expiration on subscription channels. If there is about 5 minutes (?) of inactivity, the subscription is cancelled. This will render the client responseless to user commands (they are no longer transmitted to the host). Code must be included to either keep the message channel alive (e.g. send an "is alive" message every 10 second) or automatically re-subscribe to the channel if it is cancelled. We are currently investigating which method is better.

4.2 Updating Client with Current Data

The data shown by the client need to reflect the current data at the host. Updating a variable in the host VI will not automatically update the corresponding variable on the client (as was the case for the old RoboLab system). Therefore, a method has to be chosen for keeping the client up to date. Basically, this can be done in two ways: Either by sending messages to the client each time a variable needs to be updated, or by updating tagged data on the System Link and rely on the client

to poll and update its own data. The two methods can be combined by updating tagged data and then sending a message that there is fresh data to be polled to the client. Furthermore, different methods can be selected for different types of data, depending on update frequency etc.

Writing data to System Link tagged variables is straight forward, as shown below:



The configuration for the system link connection is input to the Open Tag variable. The Open Tag node can be called outside a while-loop and data written as often as needed. Proper exit code needs to be included, to shut down the link to the System Link (this also applies to the message handling).

The frequency of sending updates to the client (and System Link) has to be carefully selected to obtain a balance between “feeling alive” and overburdening the system. Monitoring the amount of data and latency is probably wise to ensure the client does not fall behind and starts showing obsolete data. Work is in progress to investigate system capacity and establish recommendations for update frequencies.

4.3 The Client Web-VI

The VI which is to run as HTML/Java code must be written as a special type of VI, identified as a “web-vi” in LabView NXG. It uses the same graphical programming environment as normal NXG code, but a more limited set of nodes, commands and functionality is available. Developing a web-vi is done like any other VI. Once a final version is ready, it is compiled to run under a HTML page with javascript and will operate as a stand-alone web-page application.

Since the web-client now is a stand-alone application it is possible to build in extra data handling capability – there is a choice whether the host should do all data processing or if all or part of the data processing should be done by the client. Arguments for transferring data handling from the host to the client is to unburden the host (if many clients are connected) and to improve response time on user input (there is no communication overhead). Arguments against is to keep the client as simple as possible, hence being able to run on client platforms with very limited computing capacity. As a start, we have selected to keep data processing on the host since it is already implemented in the old RoboLabs. Transferring it to the client web-vi application is an option which can be explored in the future but will not be done in the MEET-CINCH WP7.

This implies that the client’s task is quite simple: Show up-to-date data and send user requests to the host. Responding to user events is straightforward and done using the Event Response structure. Upon triggering on a user event (e.g. pushing a button) a corresponding message is transmitted to the host where it is received and processed as already explained in chapter 4.1.

Keeping the client up-to-date is currently done by the two methods previously described (polling and messages). For data which are constantly changing (e.g. measurement time) polling is preferred, since it seems to use less bandwidth than constantly sending messages with new data. For data that is only changing infrequently, messaging seems to be the most efficient method. Constantly polling the data is using a lot of bandwidth for just showing the same value; sending a message each time the data changes is much more efficient. Of course, care must be taken that such a message is not lost and an obsolete data value is being shown due to the data loss. A refresh all data button or update with a given frequency (e.g. once per 15 second) might be useful to avoid such trouble. We have tested out and have solutions for all the mentioned methods. Which one or

what combination one selects will depend on the particular implantation of a RoboLab system which is being made.

5 CONCLUSION

A functional LabView NXG program using the System Link cloud service was written for the Neutron Activation of Silver RoboLab system. All the old functionality was retained and the LabView code will serve as a template for upgrading other RoboLab systems to the new LabView NXG code and web-functionality.

Using the System Link service adds stability and ease of use to the RoboLabs. It also enables added functionality and possibilities. However, the programming is slightly more complex than with the old method (which is no longer supported by National Instruments). The template developed in this Deliverable should make upgrading substantially more easy.

As work continues on developing final version of all the UiO RoboLabs new and improved versions are continuously being developed. Therefore, no code is submitted together with this Deliverable, but the most updated and current code is always available from the UiO crew for anyone interested. Please contact:

Olga N. Salina (o.n.salina@kjemi.uio.no) or

Jon Petter Omtvedt (j.p.omtvedt@kjemi.uio.no)

Integrating video streaming into the RoboLab web-page is not covered by this Deliverable, but should be straightforward using commercial streaming services that can be embedded in HTML pages. Examples on how to do this will be demonstrated in the code which will be delivered in Deliverable 7.2 (due in May 2020).