A written document for Marshall Plan Scholarship Foundation

A LabVIEW Toolkit for the Development of iLab Batched Lab Servers

Author:

Chintan B. Rajyaguru

Supervisor:

Danilo Garbi Zutin

Prof. Dr.-Ing Dr. sc (habil)

Michael E. Auer

Index

Carinthia University of Applied Sciences
School of Systems Engineering – Prof. DDr. Michael E. Auer
Head of the Center of Competence in Online Labs and Open Learning

Europastrasse 4, A-9524 Villach, Austria
Phone: +43 5 90500-2115 , Fax: +43 5 90500-2110
M.Auer@IEEE.org ,www.ctl.ac.at/auer

FACHHOCHSCHULE KÄRNTEN

Center of Competence in Online Labs and Open Learning

Chintan Rajyaguru

University of Bridgeport

Letter of Appreciation

Chintan Rajyaguru was part of our group from 1st April, 2010 to 1st September, 2010 within an internship funded by Marshall Plan Scholarship Program.

The work of Chintan Rajyaguru was the development of a LabView Toolkit to streamline the process of developing an iLab lab server for any type of laboratory based on LabView. The toolkit is targeted at batched lab servers within the framework of the iLab Shared Architecture (ISA) developed at the MIT. The task of Mr. Rajyaguru was to implement a batched lab server experiment engine in LabView. By shifting part of the code to LabView allow lab developers familiar with LabView to create new lab servers in an easier way by configuring some parameters by use of a common interface.

The presented talks and reports were well written and structured. It shows the ability of Chintan Rajyaguru to autonomous scientific work.

Together with his supervisor, MSc. Danilo Garbi Zutin, the results of the work done will be submitted as proposal for a presentation on the EDUCON2011 conference.

We thank Chintan Rajyaguru for his diligent and intensive research work within an international cooperation project.
To the best of my knowledge I can recommend Chintan Rajyaguru for a PhD study.

Kind regards,

Michael E. Auer

Acknowledgements

Foremost I would like to acknowledge Marshall Plan Foundation for providing funding opportunity for this very important research project. I am obliged by Dr. Michael Auer, Dr. Andreas Pester at the Carinthia University of Applied Science, Dr. Tarek Sobh and Dr. Navarun Gupta at the University of Bridgeport for introducing this opportunity to me and for guiding me whenever needed.

I am grateful to my supervisor Mr. Danilo Garbi Zutin for constantly motivating and guiding me throughout the period of research. He has helped throughout for troubleshooting various problems and helped increase the productivity of the project. I would also like to thank Mr. Christian Maier and Ms. Diana Pope for camaraderie shown at the work place to make it more friendly and enjoyable.

At the same time, I would like to acknowledge International student service department staff such as Ms. A. Jama, Ms. N. Doris for their support and guidance from invitation to farewell.

Last but not the least I would like to thank a million to my family in India and the United States of America for bringing me up in such a nice and disciplined environment and to support my education to help me reach at this level of success.

Introduction:

iLabs [1] is dedicated to the proposition that online laboratories - real laboratories accessed through the Internet - can enrich science and engineering education by greatly expanding the range of experiments that students are exposed to in the course of their education. Unlike conventional laboratories, iLabs can be shared across a university or across the world. The iLabs vision is to share expensive equipment and educational materials associated with lab experiments as broadly as possible within higher education and beyond.

Construction of iLabs is known as iLabs Shared Architecture (ISA). ISA is divided into three tiers that provide different services. These tiers are client, Service Broker and lab server. One of its main features, Interactive Service Broker (ISB) plays key role for Authentication, Authorization, Administration, and Ticketing.

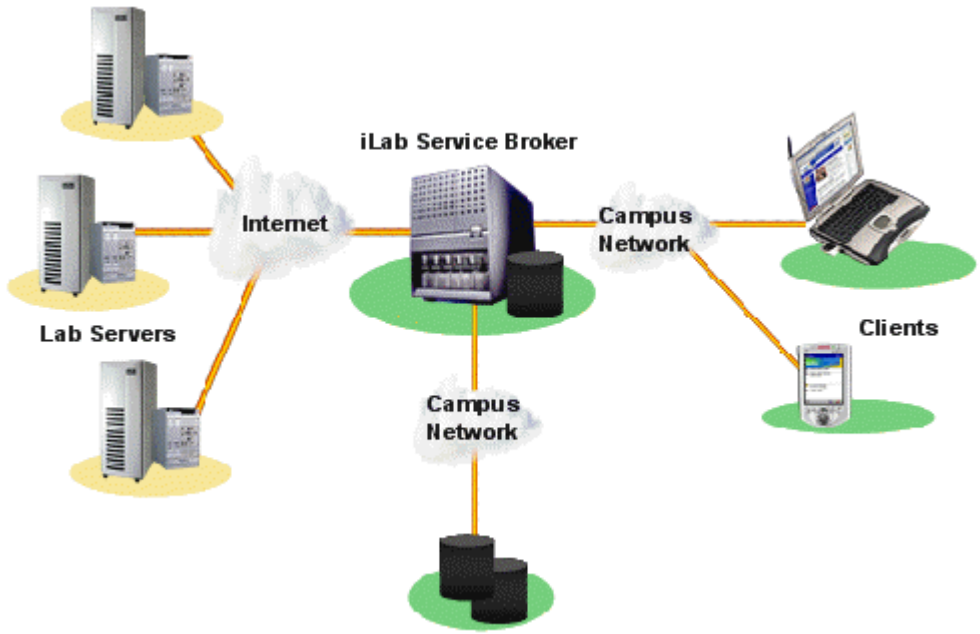


Figure 1: iLabs architecture

iLabs teams have created remote laboratories at MIT in microelectronics, chemical engineering, polymer crystallization, structural engineering, and signal processing as case studies for understanding the complex requirements of operating remote lab experiments and scaling their use to large groups of students at MIT and around the world.

Based on the experiences of the different iLab development teams, *the iLabs Project* is developing a suite of software tools that makes it efficient to bring online complex laboratory experiments, and provides the infrastructure for user management. The ISA has the following design goals:

- Minimize development and management effort for users and providers of remote labs.
- Provide a common set of services and development tools.
- Scale to large numbers of users worldwide.
- Allow multiple universities with diverse network infrastructures to share remote labs.

Addressed Issue:

In recent work, we have focused on the batched lab server architecture. Batched lab servers run experiments that can be specified in a batched mode that means, with no user intervention during the execution of the experiment. In this mode, users have to specify a set of parameters that will be used by the lab server to execute the experiment and return the results back to the lab client. Furthermore the lab server must queue the experiments requests in a database and process them. In other existing batched labs the execution of an experiment is the task of a separated piece of software named experiment engine (EE). Basically it keeps checking the database for unexecuted experiments. As soon as if finds one it fetches the experiments specification parameters and execute the experiment. The results are written back in the database fetched by the Service Broker via the Web services interface (As shown in figure 2). This interface mainly exchanges following information; Lab Hardware Configuration and setup, Experiment Parameters & Results and Lab Hardware Status.

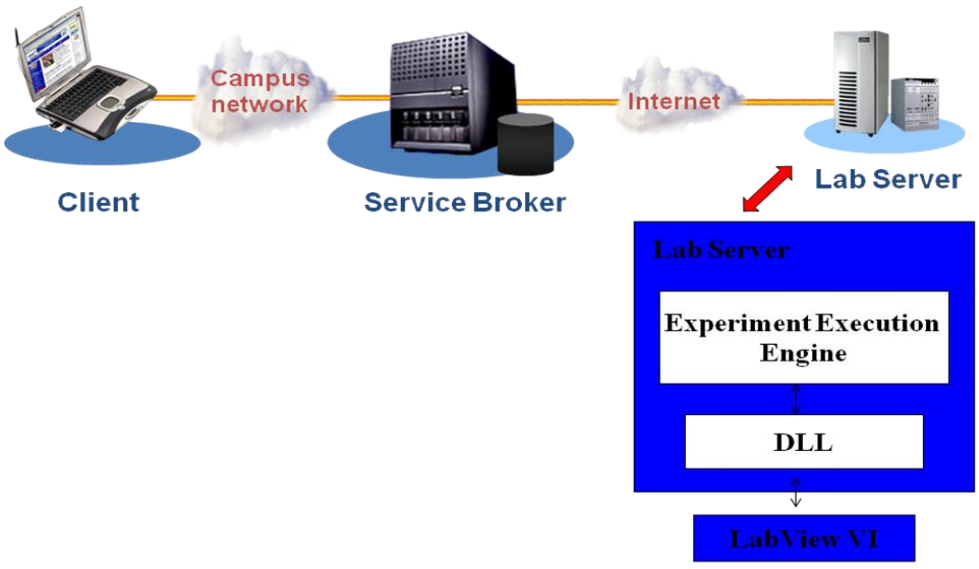


Figure 2: Block Diagram

Whenever the Lab configuration was changed it made Lab developers job tedious by rewriting code by making required changes, on top of that lab developer had to recompile and rebuild the whole code all the time; made it more time consuming. Allied to that, above mention issue provoked an excesive need for lab developer's coding aptitude that is not necessarily possible for all the cases. To get rid of afore mentioned issues an adaquate solution was introduced; configurable XML parser for the LabView based labserver. XML is used as vehicle to transfer the batched parameters; therefore the developed LabView toolkit must implement an XML parser that is easy to be updated to support additional parameters and data types. The implementation initiated by development of xml parser for an EE of the batched lab architecture in so doing an express VI was introduced that has an intelligence level high enough to interpret the xml input and perform changes accordingly on the lab configurations on the LabView script of lab server.

Procedure:

Early phase

During the research period, first few days were invested for the thorough literature study. This study includes detailed study of iLabs multi-tier architecture, Lab server and ISB web interface, ISBs functionality, Experiment execution engine code.

Another important task whilst literature study was building up the system. Such as,

Installation of Visual Studio .Net 2005 Express Edition

Windows 2003 Server Enterprise Edition

Setup of SQL server 2K5

Microsoft Dot Net Framework Ver 2.0

LabView 2009

Above mention setups are requirement for installation of iLabs service broker and a model lab server. An installation manual is provided by MIT's iLab team. By following instructions from this manual one can easily build up service broker and lab server. The installation process is described step by step in the file named as Merged_iLabs_Bootstraping_v5.doc. The general process and important details for installing the Lab server is given below:

For each of the installed iLab services the following general processes will have to be followed:

- Create virtual directory
- Create the database (tables and stored procedures)
- Edit Web.config as needed
- Perform self-registration

After a service is installed the Service Broker needs to create a trusted relationship with the service, this is done in the bootstrapping section of this document.

Once the IIS website and virtual directory is created following checklist is repeated for all the Lab server software pieces. E.g;

iLab Service Broker

- Create Virtual Web Site
- Create Database
- Database Permissions
- Database Process Agent scripts
- Database Ticketing scripts
- Database Interactive Service Broker scripts
- Edit Web.Config file
- Test service
- Register domain services

Above mentioned checklist repeats for little or no modification for other services such as; Experiment Storage Server, UserSide Scheduling Service, LabSide Scheduling Service, Interactive Time Of Day Server. Once all the steps given in the manuals are completed one can setup lab server and service broker locally and verify it. This process provides thorough insight of the functionality of architecture. In so doing various practices can be performed such as executing experiments on newly installed lab server and to create user accounts to learn functionality of service broker. Such as; Authentication, Authorization, Administration, and Ticketing.

After initial setup of the system, it requires to have LabView (LV) setup as we are doing certain modification on LV specific lab experiments. Study of XML toolkit of LV is very important as the web interfaces uses the most widely used xml platform for exchange of data between Lab client and Lab server. With this reason LV introduced an xml parser toolkit. The block diagram of xml parsing within the LV is shown in figure 3.

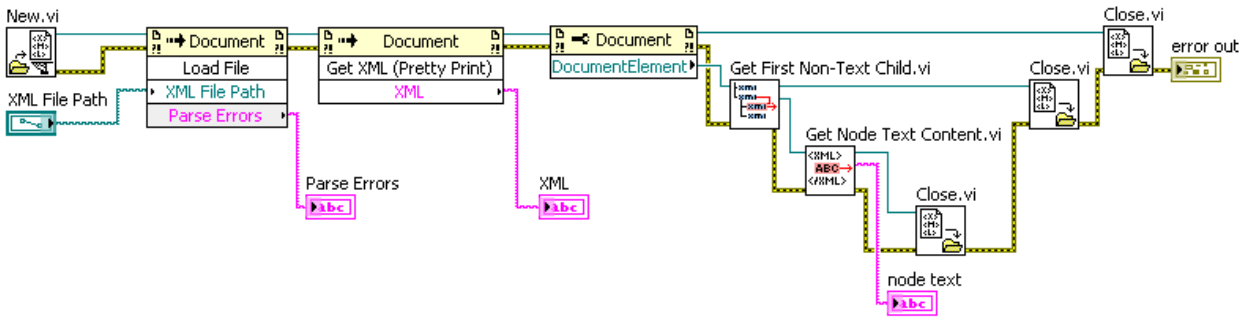


Figure 3: XML Parsing in LabView [2]

LabView uses xml Schema for this purpose where various data types of input and output of functional elements of a VI are interpreted through it. Whereas using LV xml schema put few limitations, to make in more generic the configurable parser toolkit is required.

Before moving forward, it is necessary to know how? Exactly LV schema converts any type of provided data input/output into xml and vice versa.

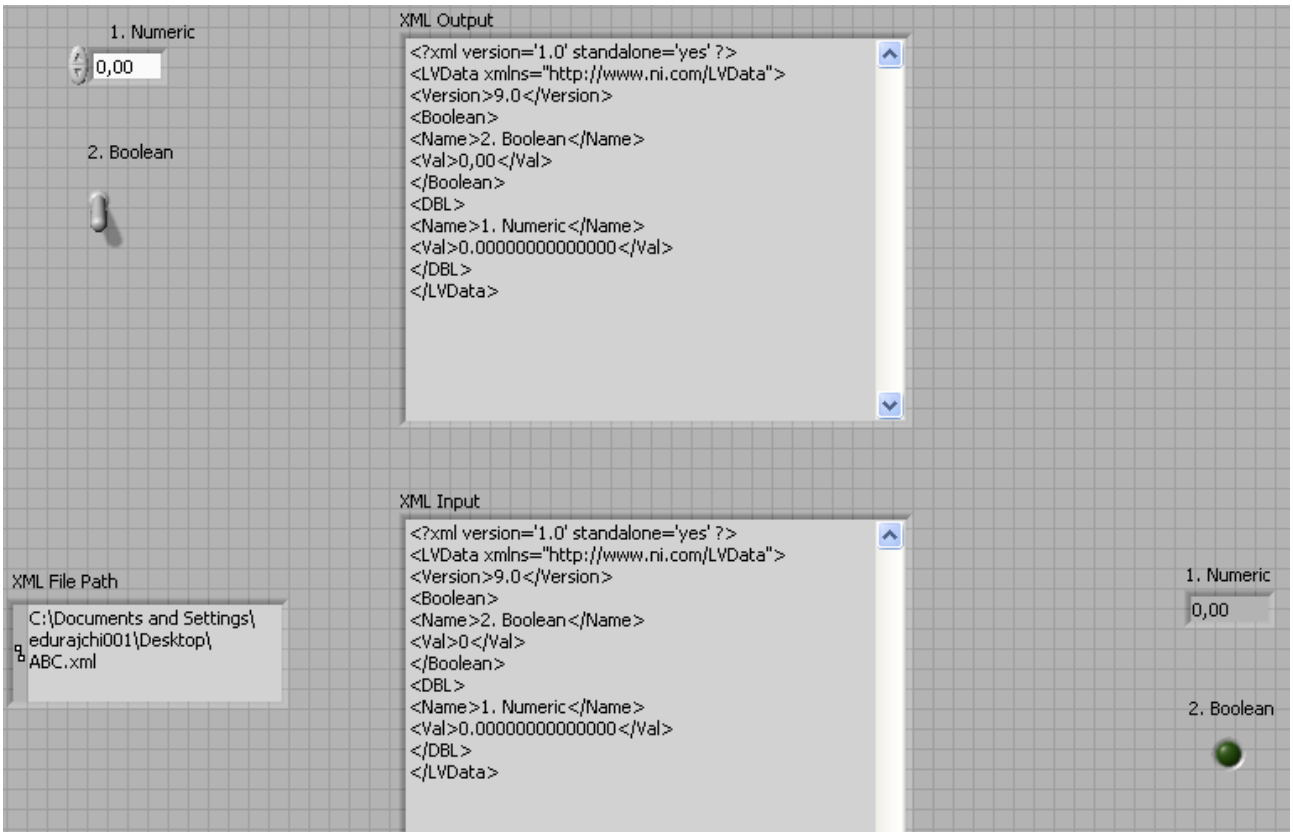


Figure 4: xml parsing in LabView

As shown in the above figure 'XML Output' text indicator shows xml header for LV it also distinguishes the data types from one another such as, Boolean, DBL, etc. It also mentions the tag name and related value. The other part of the figure demonstrate parsing of values from given xml file path. It also indicates the effect on the text box and led indicators of the virtual instrument (VI).

Limitations with XML LabView Schema:

LV converts data to an established XML schema. Currently, you cannot create customized schemas, and you cannot control how LV tags each piece of data. Also you cannot convert entire function or VI to XML. [2]

If the above mentioned limitation was not an obstacle, any change made could be delivered by an xml script and that could be converted to the LV functions. As because of this limitation it is necessary to build a configurable xml parser that collects an xml file and derives the values from it. Not only that, it should also be able to help lab developer change the parameters. The change in parameters not only meant with the values related but also the tag names. The available set of parameters can be used anywhere by just drawing out the wire from the terminals. In all, foremost important is to maintain the data type of specified parameter. As LV supports many data types it is extremely important to change the available xml file into original data type. This makes a coding bit complex, at the same time; we have to measure the time constraint by adding up an extra parser function before an experiment starts.

Two approaches

Let us discuss two approaches that were applied.

Approach 1:In the first approach, it is to make a configuration interface that consist all the available tools that are necessary for a specific type of lab are mounted. A developer whenever needed to change lab configuration changes an xml script. The front end of configuration interface must be smart enough to utilize that information in such a way to modify the tools. Such as, discard few of the unnecessary tools and enable other tools according to the specified in the xml script. For an example,

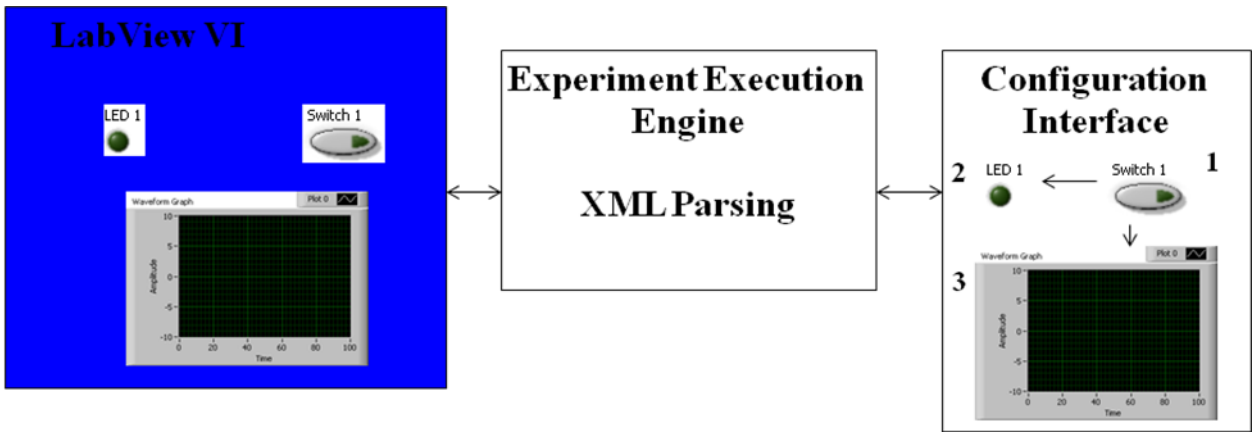


Figure 5: demonstration of configuration interface with toolkit, Approach 1.

As shown in the figure, lab developer describes the function of an experiment and not only that it specifies the order of function in an execution process. This information is transmitted using simple xml code. Once the code is interpreted by the front end or a LV based experiment execution engine that enables the necessary function and an experiment is run

according to the need of the situation and as desired by the lab developer. This process does not even require any type of modification in the code of experiment execution engine. According to the need of the lab this type of labs can be designed in advance.

As shown in the snapshot (figure 6), is a very basic example for this approach. It shows an xml code that contains information of the flow of an experiment. At the same time it contains information that enables or disables a particular function of the LV VI. It is shown the block diagram of the configurable interface VI, which is mainly taking advantage of the case structure function of LV toolkit. Each function of the LV VI is connected to a virtual switch; this switch turns ON/OFF according to the xml file that basically manages the operation of the switch. In more advance form of this approach we also tried property nodes such as visible; it not only disables a particular control, indicator but also makes it visible or invisible from the sccreen at the time of execution. It makes this approach pragmatic rather than a ludicrious scheme.

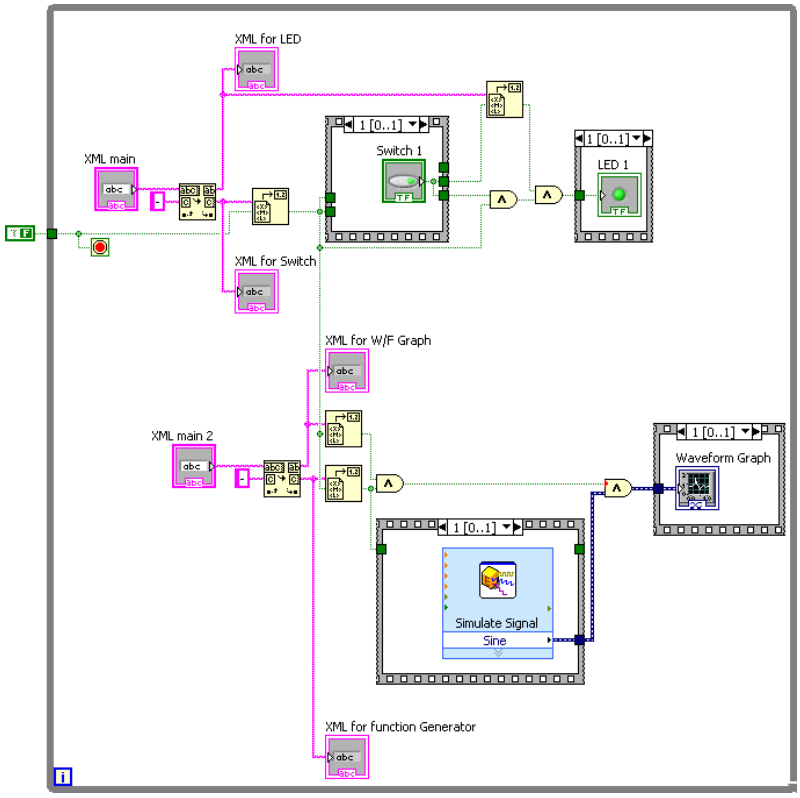
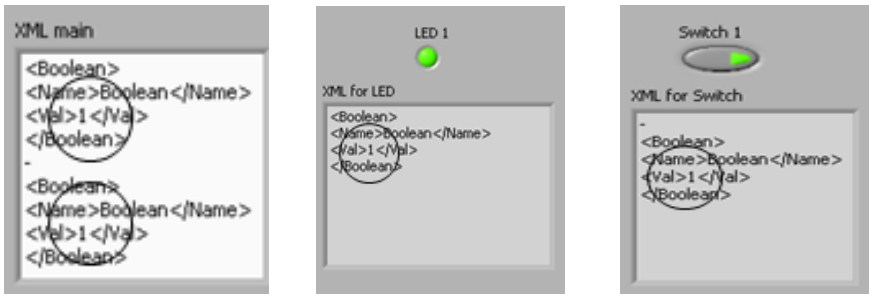


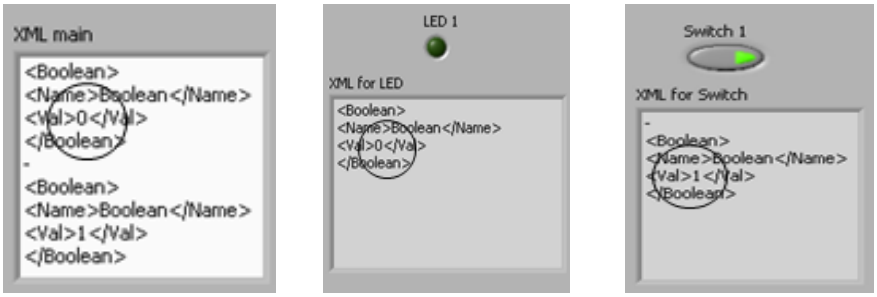
Figure 6: block diagram of configuration interface approach 1

Another snapshot (Figure 7) of coding shows the front panel of the configuration interface. It portrays a basic example where xml code syntax with LV xml schema interacts with the functional units to enable and disable then from an experiment execution. It mentions not only

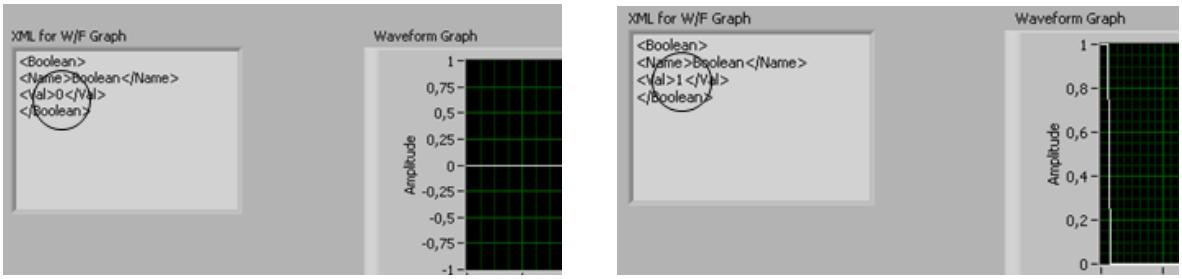
an access to that function but also maintained the flow. Hence by developing this approach further and adding further things make it very sophisticated configurable toolkit for a specific experiment. In the figure 7a, on the left it shows main xml code that contains status for LED and switch 1. Whereas, in center it shows that LED is ON because of switch 1 is ON and according to XML for LED it is enabled. As you notice a change in the figure 7b about where LED is disabled and even if the switch is ON as LED stays OFF. At the same time, figure 7c



(a)



(b)



(c)

Figure 7: Front panel of configuration interface approach 1

shows waveform graph indicator. As looking at afore mentioned reasons it seems that approach number one is an ultimate option for the addressed issues. But we started measuring on the dark sides of approach 1. Enlisted below are few of them that inspired to move on towards the basic approach and that were a recommendation of more experienced supervisor.

1) Modification of the toolkit all the time:

The Laboratory experiment and its configuration change a lot during a coursework. In approach one modification of the code written in LV becomes a mandatory part. And not only that the codes become more and more complex as we try to improve representation of the configuration panel and its toolkit. There must be a way, where such changes become a least priority or unnecessary.

2) Complex coding: Not only that it requires special attention from lab developer to modify xml all the time as per the need of the laboratory specifications, it also becomes must to make changes in the LV VIs. Summing up all shows that the coding is increasing efforts with the trade off for some mere facilities.

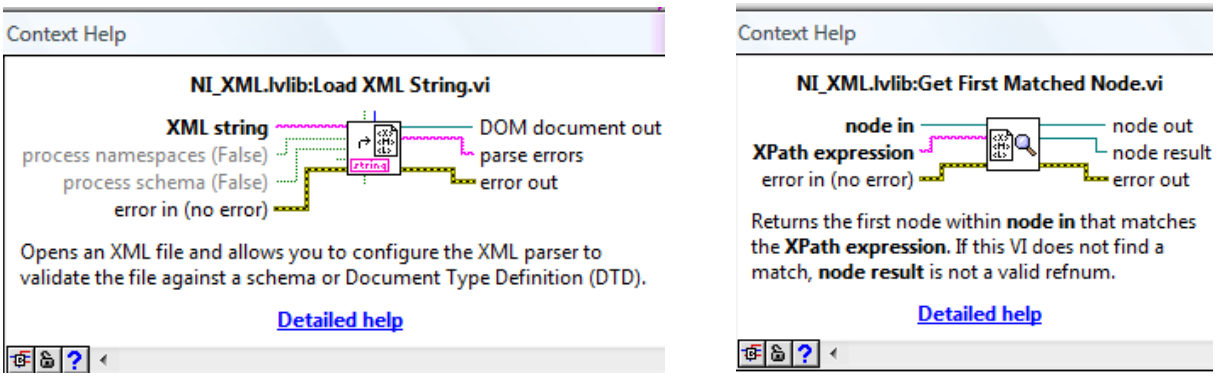
3) Processing time and accuracy: The coding shown in the block diagram shows that it not only consumes time in parsing the xml and conversion to verified data types but in selection process of the functions and assignment. The execution speed is directly proportional to an amount of elements in an experiment. Hence, it slows down the execution process makes an experiment error prone due to unwanted and unpredictable delays.

Above mentioned negatives made few and many improvements in our approach towards the solution of addressed issue such as to reduce efforts from lab developer and save time whilst developing a laboratory experiment.

Approach 2 (A2): In this approach the most significant difference from approach 1 is that it is simple. According to A2, we make a configurable xml parser of our own that runs independent from LV xml schema. The salient feature of such configurable xml parser is that it is volatile towards any addendum parameter to a laboratory experiment with no modification required on the lab experiment VIs. Whenever lab developer wants to add any new parameter to the lab experiment, xml input file has value of that new parameter with all other parameters. The only task that lab developer needs to do is to make changes in the configurable XML parser function's configuration interface panel. The whole function is divided into various parts such as, **xml parser, data type verification, xml input and output (front panel) arrays and Express VI.**

XML Parser: Input XML string from experiment execution engine taps to the "Load XML String.vi" (Fig 8A) sub VI. It creates a document file that consist an xml file. It also indicates if there are any parsing errors. Xml file is given as an input to the "Get First Matched

Node.vi" (Fig 8B) sub VI from XML toolkit at the same time it is given an address of the desired parameter in the xml file as an input, xml path expression. The function gives out the value of xml line that holds desired parameter then this line is given as an input to another xml function to get a value of that particular parameter. An output value is still in the form of string so it is necessary to convert them into appropriate or predefined data type.



A B

Figure 8: Source: LV Help, xml parser tools for the configurable toolkit

Data type verification: As shown in the figure 10, various data types are converted from the xml input to the case structure. This case structure provides distinguished arrays at the end of the execution. Arrays contain the meaningful values of a particular parameter and in other places it appends "0" value or " " (spaces) to filter out the required value by a simple indexing process. These data types are in fact all the data types those are supported by LV 2009. It includes, double, signed integer (8, 16, 32 and 64) bit, unsigned integer (8, 16, 32 and 64) bit, and single precision (8, 16 and 32) bit, fixed point, extended real precision, extended complex precision, single complex, Boolean, and string. This case structure makes its decision through the data type information provided by the lab developer. As shown in the figure 9, it is shown an indicator for the cluster of data name and data type. This cluster is split as shown in the block diagram figure 10 and the data type array goes to the case structure for case selection. In the case structure according to case a string/ number conversion function is used with its conditions set according to a particular data type to be converted.

Front Panel: Figure 9 is the figure of front panel of the source VI that consists the input and output terminals. Such as it includes, xml input file in which it is written a code of real example of one laboratory named as MIT ELVIS Lab. In this xml script it consists of various parameter specifications such as amplitude, frequency, waveform type, offset and phase.

These values have different data types. The task of the lab developer is to provide the parameter names for xml path expression and to provide its data types for the very first time when he sets up the lab. To facilitate the lab developer tasks the root of the xml path expression is already provided and that is also changeable at the times whenever there is a need of modification in the xml code. Later on the parameter names get concatenated with the root to create xml path expression that helps to derive a particular value of a desired

parameter in the output. As the data type list has gone bigger another facility provided in the front panel is to give a drop down menu of all the available data types. It makes developers

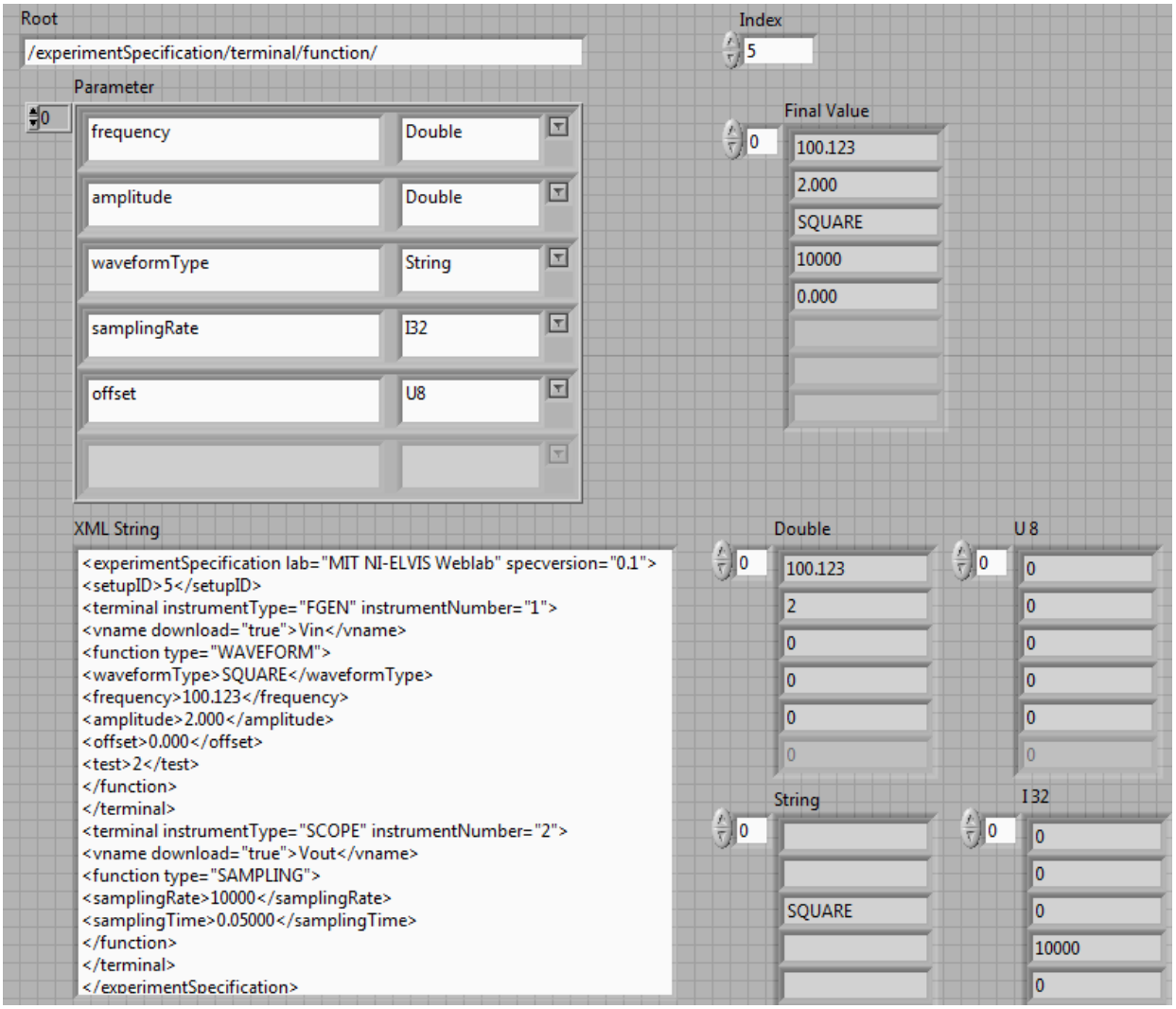


Figure 9: Front panel of the final source VI (final.vi Front Panel).

task easier even if the laboratory consists of many parameters in it. The other side of the front panel describes various output arrays. The first array “Final Value” contains all the derived parameter in the form of string data type. Whereas, other arrays such as double, U8, String

and I32 represents the values with an appropriate data types. As it is not possible to show all the arrays on the screen shot other indicators are hidden. Summing up all above represents the front panel and a working model of our approach 2. But it is not yet ready to be implemented as ready to use end product. To design an end product that is maximally user friendly a latest approach of an Express VI is introduced.

Express VI: An Express VI is a VI whose settings you can configure interactively through a dialog box. Express VIs appear on the block diagram as expandable nodes with icons surrounded by a blue field.

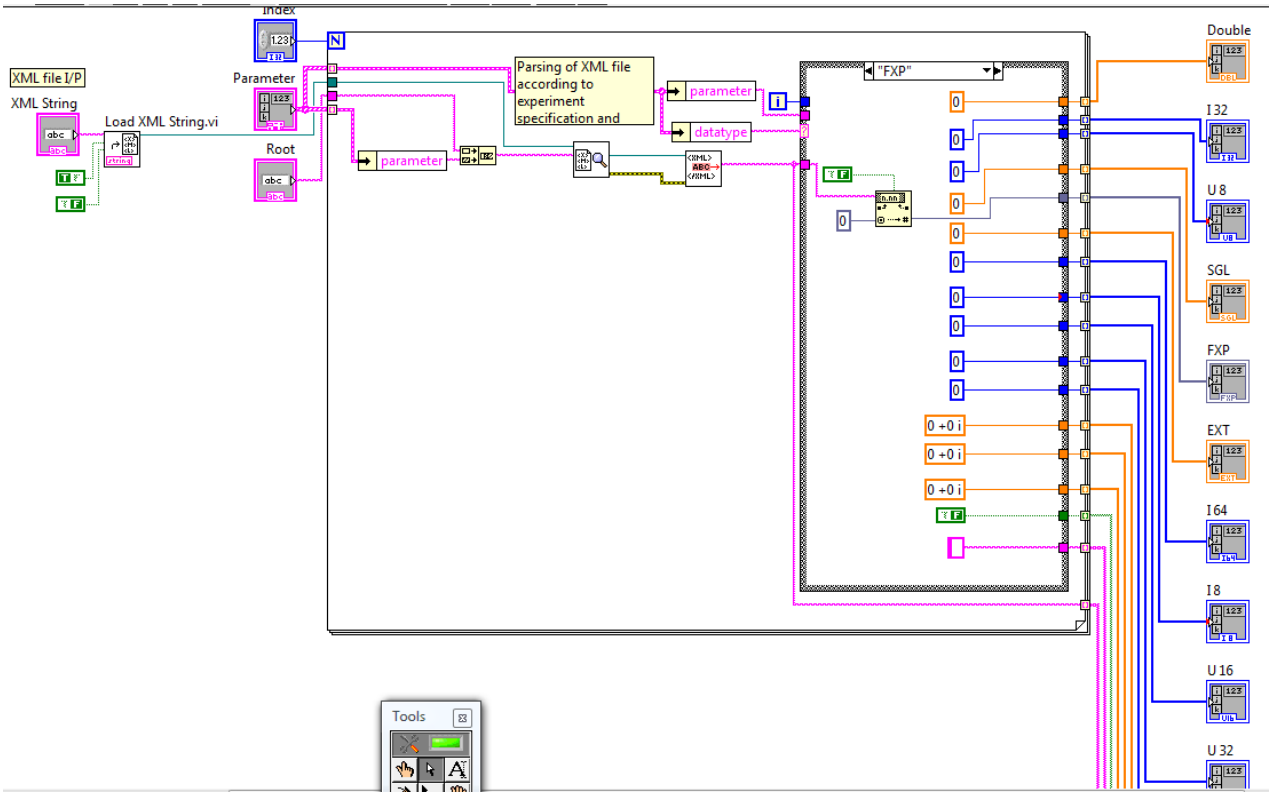


Figure 10: Block diagram of the final source VI ; final.vi Block Diagram

You can configure an Express VI by setting options in the configuration dialog box that appears when you place the Express VI on the block diagram. You can double-click the Express VI or right-click the Express VI and select **Properties** from the shortcut menu to reopen the configuration dialog box. You also can configure the Express VI by wiring values to terminals of the Express VI on the block diagram.

Parameters that you can configure for an Express VI from the configuration dialog box are called configurable parameters. Parameters that you can configure for an Express VI from the block diagram are called expandable parameters or expandable terminals. A parameter can be both configurable and expandable, which means you can configure it from both locations. The configuration you specify for an Express VI with the configurable parameters determines the expandable terminals that appear on the block diagram.

You can use Express VIs to perform common measurement tasks. You also can create and edit Express VIs to distribute to users for building applications easily. Use the Create or Edit Express VI dialog box, available by selecting **Tools»Advanced»Create or Edit Express VI**, to create an Express VI from an existing VI, from another Express VI, or from a blank VI.

Benefits of Express VI: The primary benefit of Express VIs is their interactive configurability. Express VIs are useful when you want to give users a VI or library of VIs for building their own applications easily with minimal programming expertise.

The configurability of Express VIs provides an interactive way to determine settings for operations that the user might not fully understand.

Another advantage of Express VIs is that separate instances of an Express VI function independently. When you place an Express VI on the block diagram, an instance of the Express VI is embedded in that particular block diagram. The settings that you select in the configuration dialog box affect only that instance of the Express VI. If you place a standard VI in five different areas on the same block diagram, the result is five exact copies of the VI. The source code, default values, and front panel remain the same for all five copies. However, if you place an Express VI in five areas on a block diagram, the result is five separate Express VIs with different names, all independently configurable. Each instance of an Express VI can include different numbers and types of expandable terminals. The **Context Help** window also updates with the different settings for each instance. Express VI is mainly divided into two parts. Source VI and Configuration VI.

Source VI: The source VI of an Express VI contains the underlying code for the Express VI. The source VI serves as a wrapper around a subVI that determines how the Express VI behaves. The source VI also determines the connector pane and icon of the Express VI. After you create a new Express VI with the Express VI Creator wizard, build the source VI to

specify how the Express VI behaves and to design the connector pane and icon. The Source VI front panel and block diagram VIs are already built and shown previously. Yet few modifications for express VI can be found out in LV help. As mentioned earlier, to determine connector pane and icon for express VI is necessary. To do that following instructions are necessary. After building the front panel and block diagram of the source VI, design the connector pane and a unique icon for the source VI.

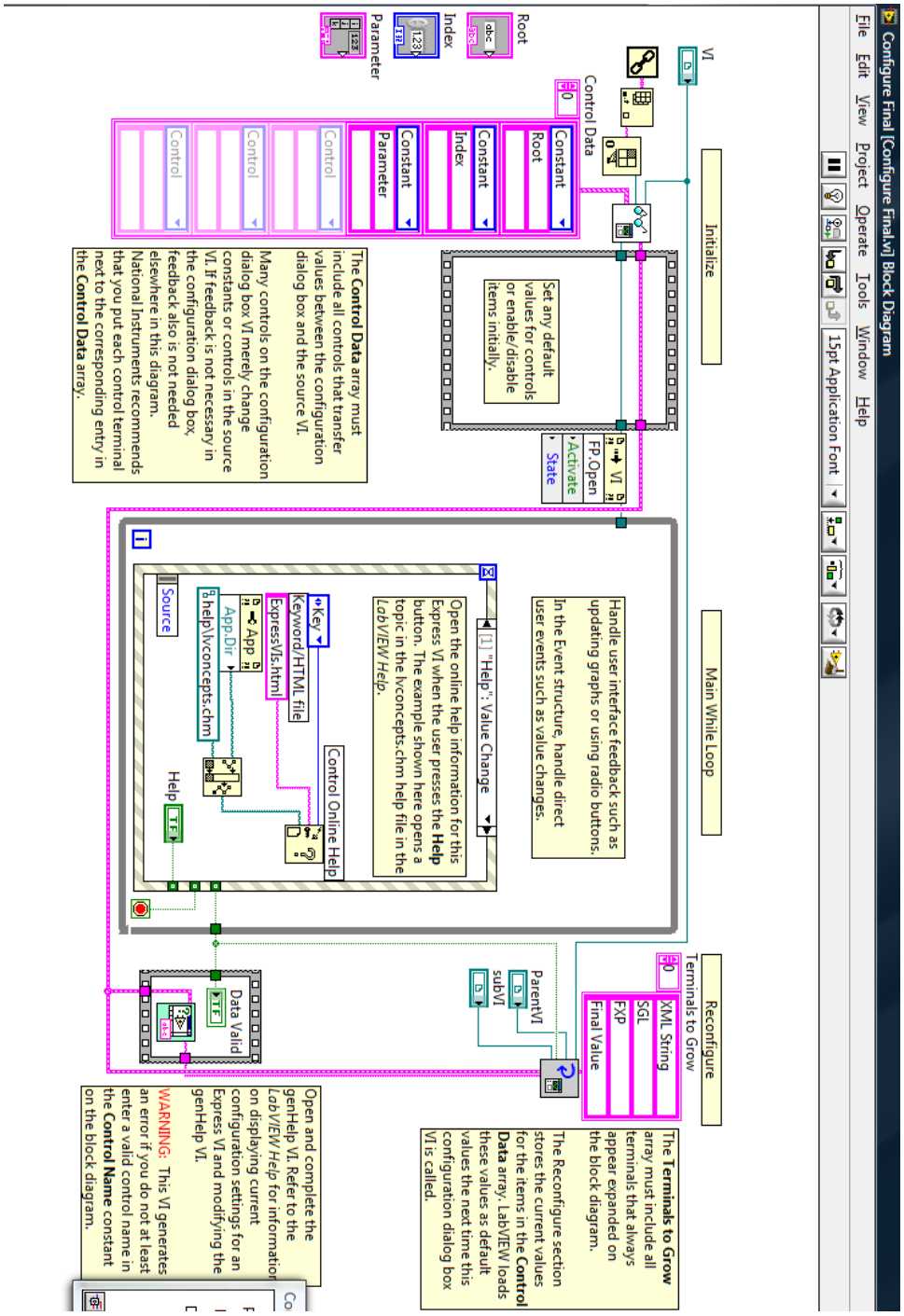


Figure 10: Block diagram of Configuration dialogue box VI

You must include all expandable terminals on the connector pane of the source VI. You do not need to include configurable parameters on the connector pane. The connector pane and icon of the source VI correspond to the connector pane and icon of the Express VI.

The configuration dialog box VI : The configuration dialog box VI of an Express VI allows users to configure settings for the run-time behavior of the Express VI. The configuration dialog box VI also contains the user interface for the Express VI. After you create a new Express VI with the Express VI Creator wizard and design the source VI, you can design the front panel of the configuration dialog box VI and then modify the provided block diagram template to configure the run-time behavior of the Express VI. Whereas building the block diagram (figure 10) and front panel of configuration dialog box VI can be found out at the LV help.

Limitations of Express VI: Express VIs do not provide run-time interactive configuration for VIs. If you need run-time reconfiguration, build an application with a user interface that contains features similar to a configuration dialog box.

Express VIs are designed for ease of use. If you need an application to run with strict memory restrictions or high execution speeds, use standard VIs.

Result: All those efforts produce an end product that is the configurable xml parser toolkit express VI. This express VI once created can be found out in the user libraries as a function that you can utilize at any place for any type of laboratory experiment just as another function of NI's LV. Once the function is selected and placed in the block diagram VI of your lab experiment VI it indicates in the context help about all the expandable terminals of the express VI. Not only that whenever you load express VI it pops up configure window as shown in the figure 11.

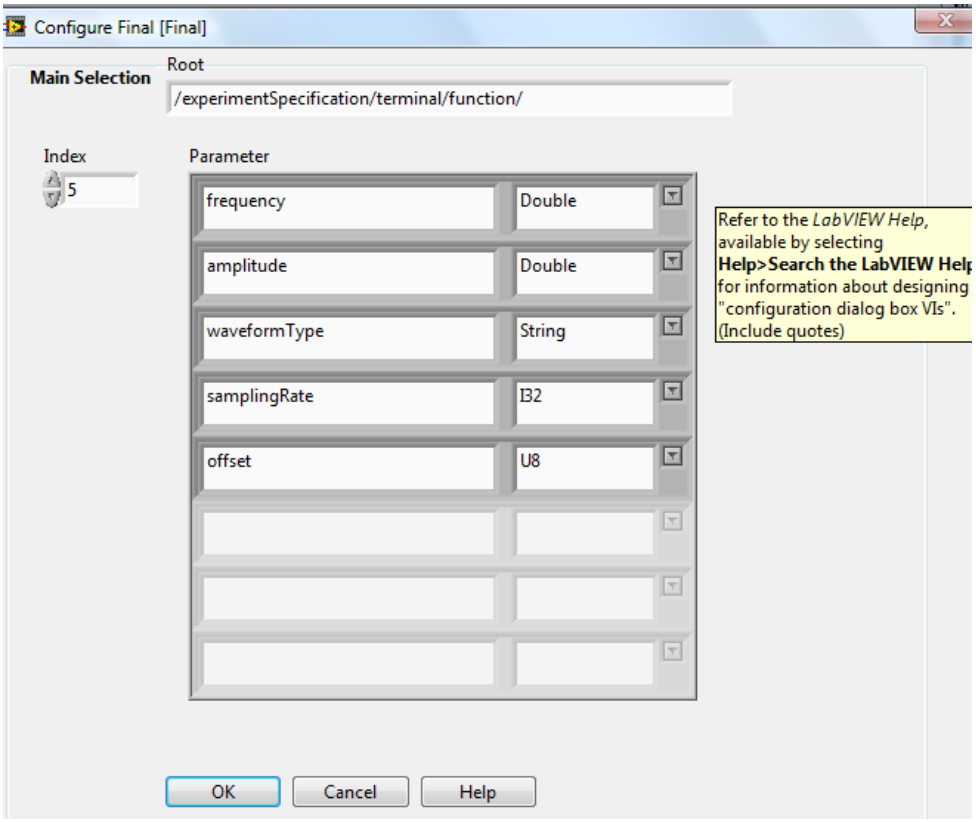


Figure 11: Configuration window for configurable xml parser toolkit express VI

In this figure one can see that it has all those things available for lab developer those were an obstacle for him to develop laboratory far more conveniently. Such as, Parameter names can be changed easily, their data type can be changed as well according to need. At the same time, if lab developer wants to add new parameters what all needed is to increase the Index number and provide their names to the configuration interface. It is also required or beneficiary to choose an appropriate data type of newly added parameter. On top of that if there is any change in the xml coding, developer can change the root that addresses the location of the parameters. Once the configuration interface is set, lab developer needs to extract the parameters by indexing them with the help of indexing table terminal. By doing so, it keeps

the VI free from tag names and there index in the parsing. These parameters are ready use for any experiment. Shown in figure 12 is an example how configurable xml parser toolkit can be used in any LV block diagram independently with very little modification. It truly stands for a need of a configurable parsing toolkit volatile to the parameter updates.

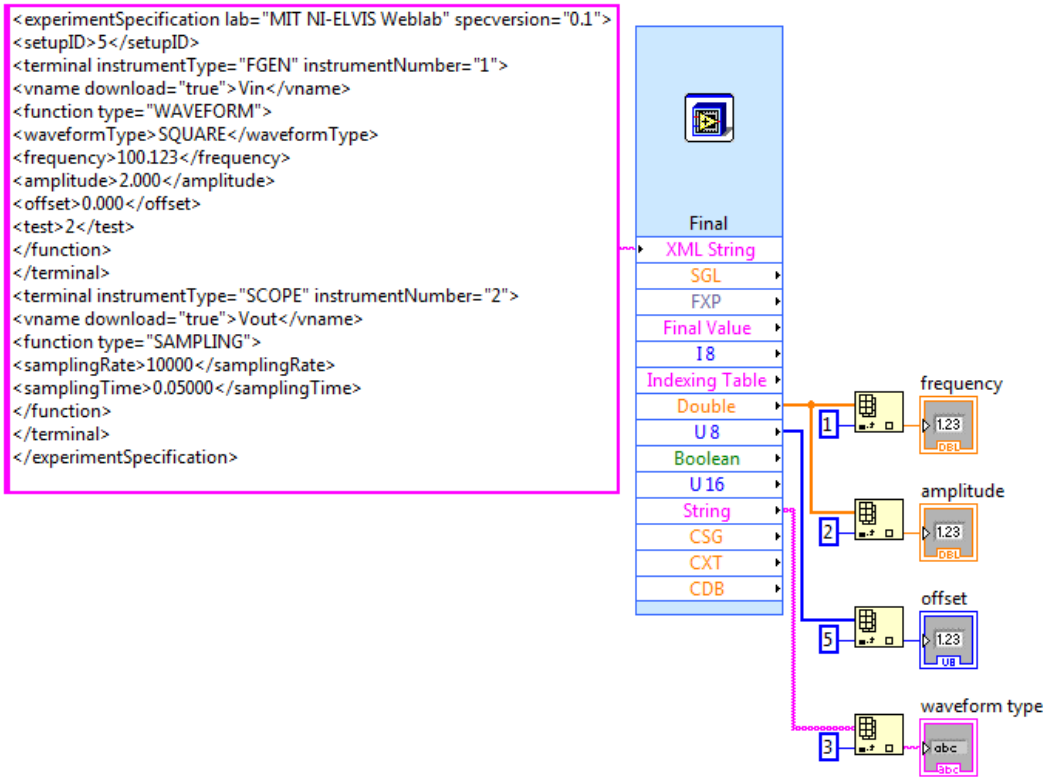


Figure 12: Extraction of parameters from configurable xml parser express VI.

Limitations or future expansion: After building up this toolkit there are certain necessary tests to be done. Such as the amount of time taken by this additional feature and its comparison with earlier version of experiment execution engine where parsing was done in the same code outside the LV. Here after calculation it shows mere 90 ms of delay added when tested with 5 parameters. There are possibilities pragmatically of Labs available with 20 to maximum 50 parameters as well. In these cases the time taken by the parser is more that makes system a bit slower whereas this is the tradeoff for providing facilities to the lab developer.

In this toolkit lab developer has to mention the tag names of his own efforts. With the possible coding capacity and my reasoning competence I can manage no programming that can help modify the tag names automatically with the change in xml file. The tag names of parameters must prevail until they are fetched from the expandable terminals of the express VI, which is

not the case. For this, in future expansion to use LV scripting is highly recommended to the successor of mine in this project. LV scripting is a very complex and the most recent facility provided by National Instruments in their LabVIEW 2010 that helps change block diagram of the VI meanwhile the execution.

Ideally, what we wish from this toolkit is whenever lab developer mentions the parameter names and index in the configuration interface it shall create an express VI functions with predetermined parameters itself with their appropriate data types as an expandable terminals. This requires an avid programming effort that we look forward to be accomplished in the future phases of this project. In other words, configurable interface can be made as sophisticated as needed but being considerate about the execution time.

Conclusion:

At the end, the addressed issues for this research assignment are resolved by putting an immaculate solution. This solution certainly simplifies many tedious efforts taken by a lab developer to build or modify a LabVIEW based MIT's iLabs batched lab servers. In addition, it presents a user friendly lab design environment that not only saves time but makes lab development a piece of cake for any lab developer. In doing so it does not affect the current architecture of the lab server. Whereas experiment execution engine becomes faster and more available for other vital activities those are performed by it other than parsing. Looking at above all benefits, this research assignment is a success and a primary step towards a highly sophisticated and user friendly lab development environment for LabVIEW based MIT's iLabs.

References:

1) http://icampus.mit.edu/ilabs/ : MIT iLabs home page.

2) National Instruments LabVIEW 2009 help.

3) Merged_iLabs_Bootstraping_v5.doc: Merged iLab Bootstrapping and Time of Day Test Lab Configuration.