

# **DDS Communication between ROS 2 and Labview (guide)**

Written by: Roald Ong

# Table of Contents

1.) Introduction.....	3
2.) Installation guide (ROS 2).....	4
2.1) Install ROS 2.....	4
2.2) Source ROS 2 setup.bash & set ROS Domain ID.....	4
2.3) ROS 2 Tutorials.....	4
2.4) ROS2 Example programs.....	4
3.) Installation guide (Labview 2018 RTI DDS Toolkit).....	5
3.1) Install RTI DDS Toolkit.....	5
3.2) Verify installation of RTI DDS Toolkit.....	5
3.3) Labview example programs.....	5
4.) Connection between Labview PC and ROS PC.....	5
4.1) LAN Connection.....	5
5.) Key points to take note.....	7
5.1) Naming of ROS2 topics in Labview.....	7
5.2) Naming of ROS2 message(.msg) types in Labview.....	7
6.) Publish/Subscribe with primitive types.....	8
6.1) Publishing an integer from Labview to ROS2.....	8
6.2) Publishing an integer from ROS2 to Labview.....	16
6.3) Applied Example: Simple Adder Program.....	20
6.4) Publishing/Subscribing with other primitive data types.....	22
6.4.1) Ensuring compatible types.....	22
6.4.2) Publishing/Subscribing with custom message data types.....	23
7.) Publish/Subscribe with Labview Clusters.....	27
7.1) Publishing from Labview to ROS2 (cluster).....	27
7.2) Publishing from ROS2 to Labview (cluster).....	31
8.) Publish/Subscribe with Labview Arrays.....	33
8.1) Publishing from ROS2 to Labview (arrays).....	33
8.2) Publishing from Labview to ROS2 (arrays).....	36

## 1.) Introduction

ROS 2 provides a comprehensive suite of advanced robotics tools/algorithms which are really useful for building advanced robots. On the other hand, Labview is really good at data acquisition and National Instruments has a whole series of embedded systems such as the CompactRIO and MyRIO which by default, run on Labview.

Therefore, when building a robot one could use Labview to handle low-level control and ROS2 to handle high level tasks such as navigation etc. In such a situation, passing data between the two platforms is necessary. As ROS 2 uses DDS as its communication protocol, and RTI provides a DDS toolkit for Labview, we are able to send data between the two platforms. Thus, this guide details the steps required to send data between ROS2 and Labview with the RTI DDS toolkit.

## 2.) Installation guide (ROS 2)

### 2.1) Install ROS 2

Firstly, go to the ROS 2 (Foxy) installation page and follow the installation instructions from (add the ROS2 apt repository) to (Installing the python 3 Libraries).

<https://index.ros.org/doc/ros2/Installation/Foxy/Linux-Install-Binary/> I would highly recommend using the linux(ubuntu 20.04) version of ROS 2. We will be using the default fast RTPS DDS implementation.

### 2.2) Source ROS 2 setup.bash & set ROS Domain ID

ROS 2 requires that we source the setup.bash file in order for the terminal to identify ROS 2 commands. Therefore, to save us the hassle of having to source the bash file for every new terminal we open, we could add the source command into our bash file. Regarding ROS Domain ID, it is used to calculate the UDP port for DDS communication, we will set it at 0.

a) Firstly, open a new terminal and key in (sudo gedit ~/.bashrc)

b) Next, copy and paste the following lines into bashrc:

```
(source /opt/ros/foxy/setup.bash)
```

```
(export ROS_DOMAIN_ID=0)
```

### 2.3) ROS 2 Tutorials

If you are new to ROS 2, I would highly recommend following the ROS 2 beginner tutorials (<https://index.ros.org/doc/ros2/Tutorials/>) as the tutorials will allow you to familiarise with crucial stuff like your ros 2 workspace etc. The beginner tutorials would suffice for now.

### 2.4) ROS2 Example programs

The ROS2 example programs are used extensively in this guide. To install them, go to the src folder of your ros2 workspace and key in the following line:

```
(git clone https://github.com/SynapseProgramming/ros2labview_examples.git)
```

Next, compile your ROS2 workspace (colcon build).

### **3.) Installation guide (Labview 2018 RTI DDS Toolkit)**

#### **3.1) Install RTI DDS Toolkit**

Firstly, navigate to section 1.2 of the RTI DDS toolkit getting started guide and follow its instructions. ([https://community.rti.com/static/documentation/dds-toolkit-labview/3.0.0.109/RTI\\_DDS\\_Toolkit\\_GettingStarted.pdf](https://community.rti.com/static/documentation/dds-toolkit-labview/3.0.0.109/RTI_DDS_Toolkit_GettingStarted.pdf))

#### **3.2) Verify installation of RTI DDS Toolkit**

Next, follow the instructions mentioned in section 3.1 to 3.3 of the RTI DDS toolkit getting started guide to verify that the installation of the DDS toolkit is successful.

#### **3.3) Labview example programs**

The Labview example programs are used extensively in this guide. To install them, go to ([https://github.com/SynapseProgramming/ros2labview\\_lvprograms](https://github.com/SynapseProgramming/ros2labview_lvprograms)) and download the file as a zip.

### **4.) Connection between Labview PC and ROS PC**

#### **4.1) LAN Connection**

For optimal network performance, I would highly recommend using a LAN connection and a capable router for communication between the ROS 2 PC and the Labview PC. This would ensure that the network is not a bottleneck. Firstly, connect the ROS 2 and Labview PC to your router using LAN cables. Although not necessary, I would recommend setting up DHCP Reservation as this will ease the process of troubleshooting connection issues(as we would know the ip address of each machine, without having to check them periodically).

a.) Next, navigate to your router's admin page and note down the IP address of the ROS 2 PC and Labview PC.

b.) On the ROS 2 PC, ping the Labview PC by opening a new terminal and key in (ping <ip address here>)

```
roald@potato: ~  
roald@potato: ~ 80x24  
roald@potato:~$ ping 192.168.1.134  
PING 192.168.1.134 (192.168.1.134) 56(84) bytes of data.  
64 bytes from 192.168.1.134: icmp_seq=1 ttl=128 time=0.490 ms  
64 bytes from 192.168.1.134: icmp_seq=2 ttl=128 time=0.458 ms  
64 bytes from 192.168.1.134: icmp_seq=3 ttl=128 time=0.447 ms  
64 bytes from 192.168.1.134: icmp_seq=4 ttl=128 time=0.502 ms  
64 bytes from 192.168.1.134: icmp_seq=5 ttl=128 time=0.481 ms  
64 bytes from 192.168.1.134: icmp_seq=6 ttl=128 time=0.480 ms  
^C  
--- 192.168.1.134 ping statistics ---  
6 packets transmitted, 6 received, 0% packet loss, time 5123ms  
rtt min/avg/max/mdev = 0.447/0.476/0.502/0.018 ms  
roald@potato:~$
```

c.) On the Labview PC, ping the ROS 2 PC by opening up the cmd terminal and key in (ping <ip address here>).

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\GooseHouse>ping 192.168.1.129

Pinging 192.168.1.129 with 32 bytes of data:
Reply from 192.168.1.129: bytes=32 time<1ms TTL=64
Reply from 192.168.1.129: bytes=32 time<1ms TTL=64
Reply from 192.168.1.129: bytes=32 time<1ms TTL=64
Reply from 192.168.1.129: bytes=32 time<1ms TTL=64

Ping statistics for 192.168.1.129:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Users\GooseHouse>
```

d.) If both PC's were able to ping one another, then this implies that a successful connection was established between the two machines. Otherwise, there may be a connection issue.

## 5.) Key points to take note

### 5.1) Naming of ROS2 topics in Labview

In Labview, we must add the prefix of (rt/) to the name of our topic. For example, if our topic has the name of (/chatter) in ROS2, we must name our topic as (rt/chatter) in Labview.

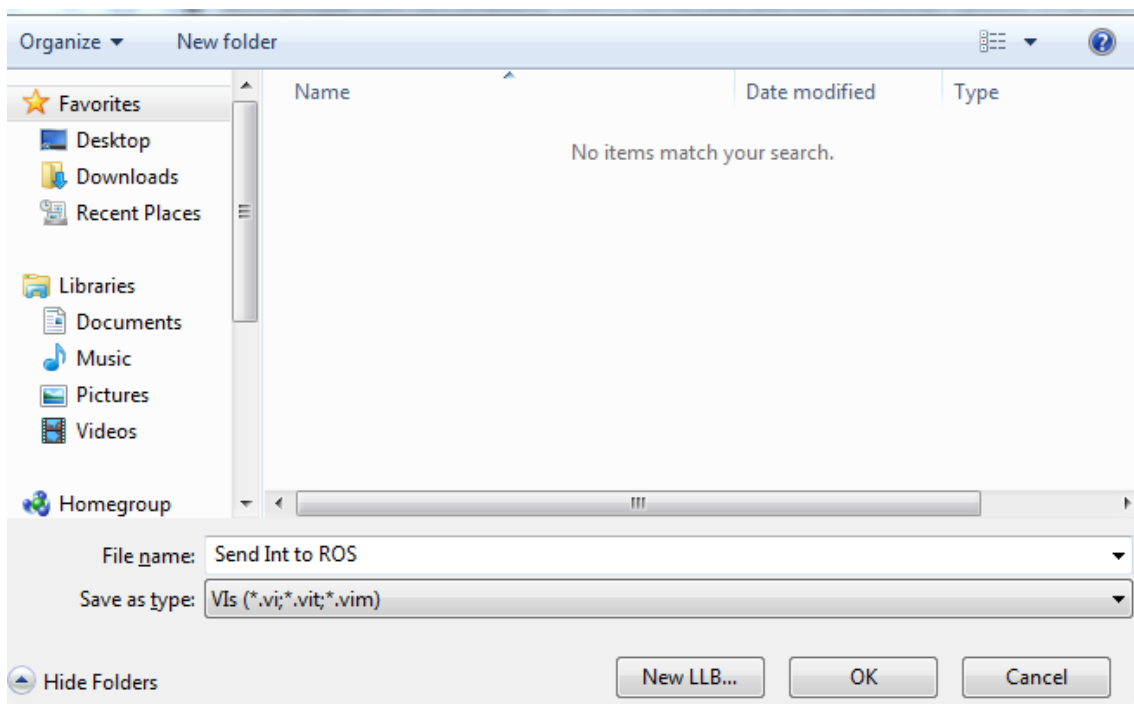
### 5.2) Naming of ROS2 message(.msg) types in Labview

One of the ROS2 message types provided by the std\_msgs package is the Int32 message. The Int32 message appears as (std\_msgs/msg/Int32) in ROS2. However in Labview, we must append extra characters to the message type in order for ROS2 to identify it. Thus, the Int32 message type in Labview would have to be named as (std\_msgs::msg::dds\_::Int32\_). In general, all ROS2 message types in Labview would have to abide by the following naming convention (<package name>::msg::dds\_::<type name>\_).

## 6.) Publish/Subscribe with primitive types

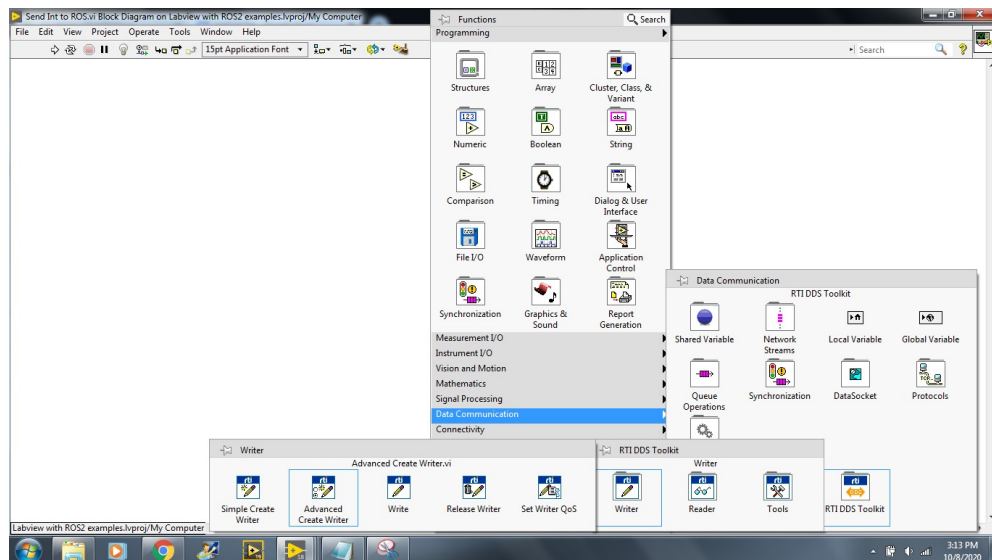
### 6.1) Publishing an integer from Labview to ROS2

a.) Firstly, create a new Vi and name it.

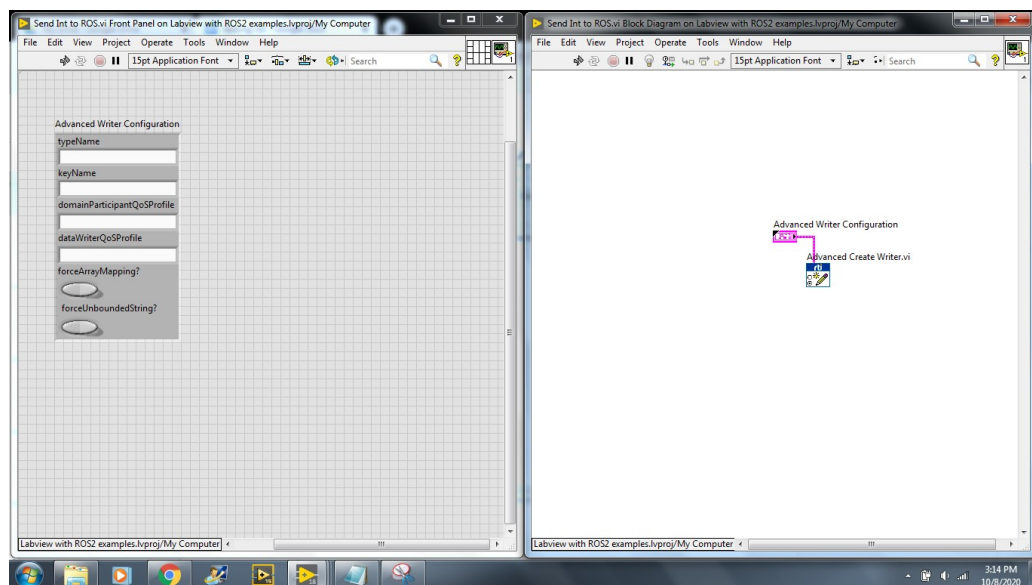


b.) Next, open up the functions palette and select data communication → RTI DDS Toolkit → writer → Advanced Create Writer

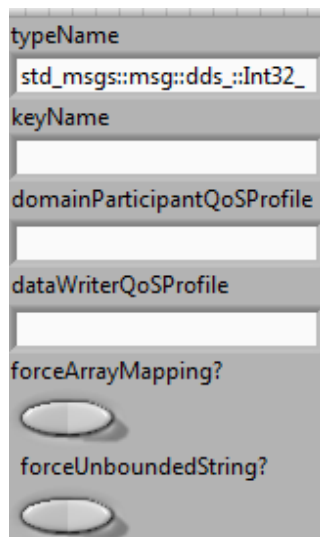




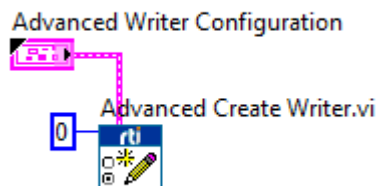
c.) Next, right click on the advanced writer configuration sink and click (create control)



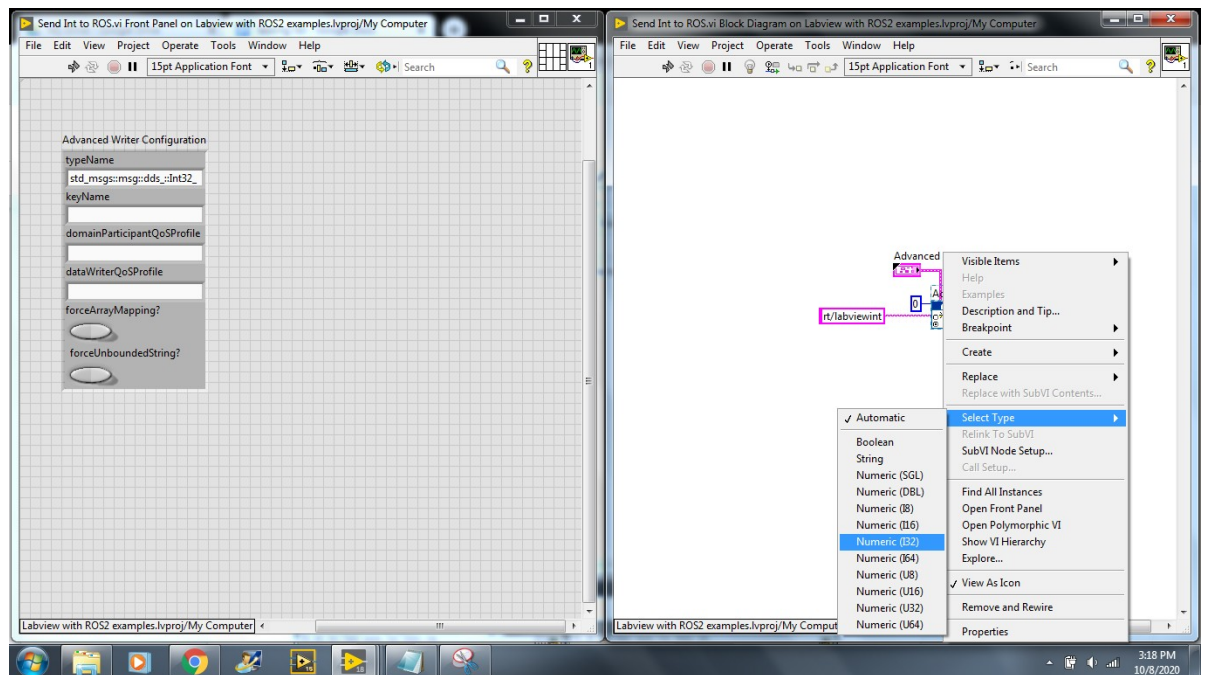
d.) Next, key in the type name of the std\_msgs/msg/Int32 message. (std\_msgs::msg::dds\_::Int32\_). Also, click edit → make current values default



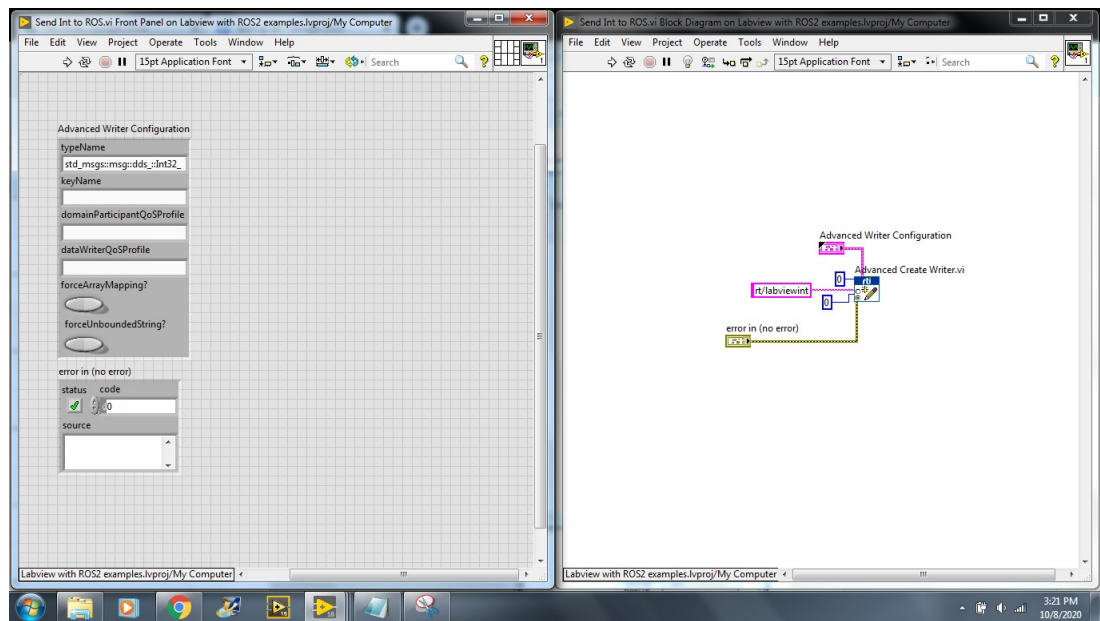
e.) Next, right click on the domain ID sink and create a constant.



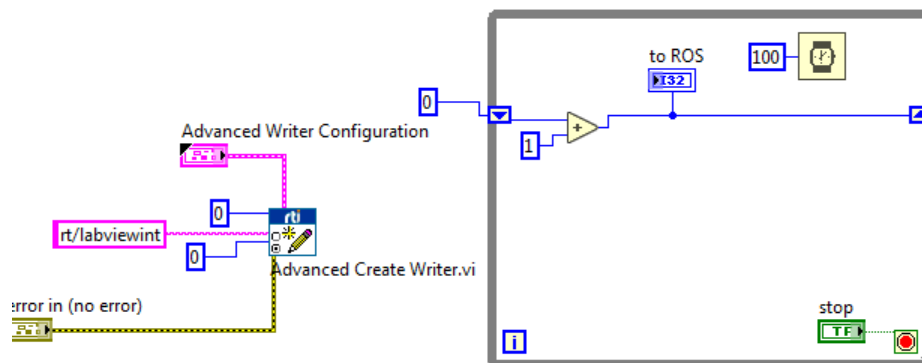
f.) Next, right click on the advanced writer VI → select type → and select Numeric (I32) Also, right click on the (topic name) sink and create a constant. Name the topic (rt/labviewint).



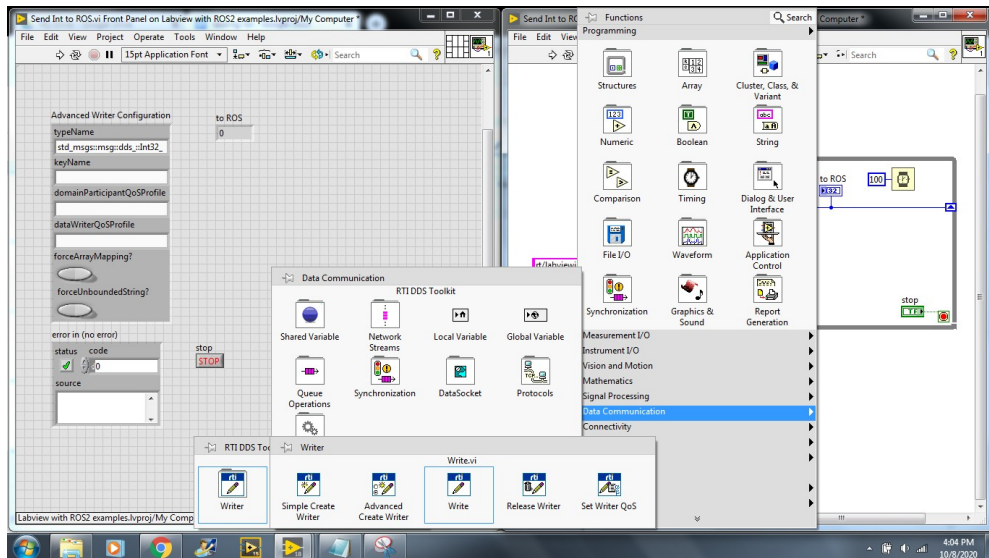
g.) Next, right click on the (error in) sink and create a control. Also, right click on the (data type) sink and create a constant.



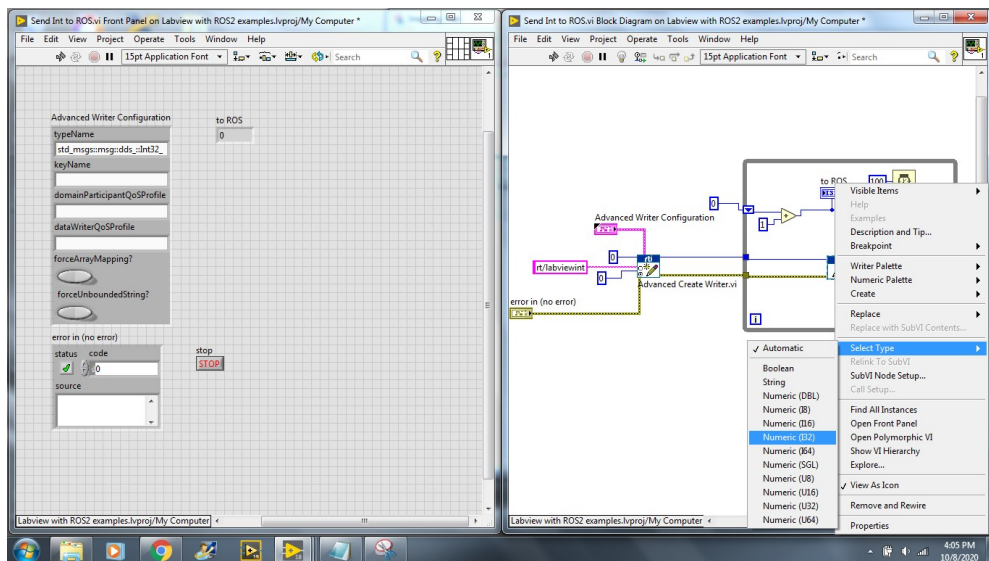
h.) Next, add in a while loop structure and create a simple counter using Int32 as the data type.(to add in a shift register, right click on the while loop and select add shift register)



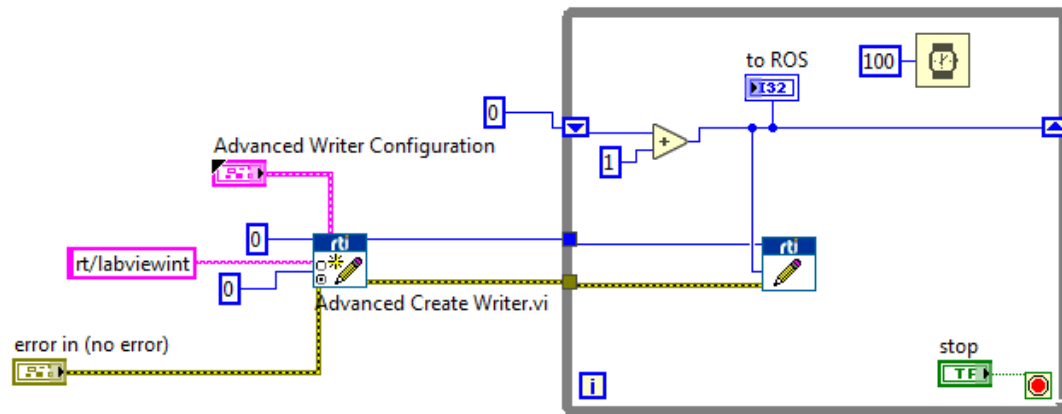
i.) Next, open up the VI palette and select data communication → RTI DDS toolkit → writer → write.



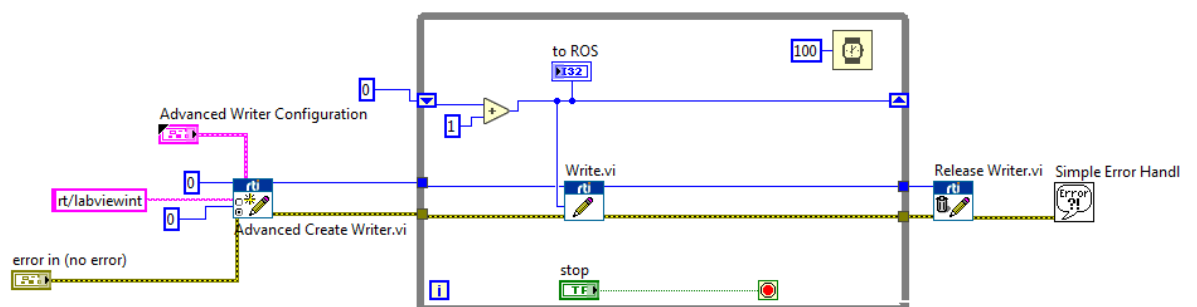
j.) Next, right click on the write VI and click select type → Numeric(I32). Also, wire up the (DDS object ref in) sink and (error in) sink from the advanced create writer VI.



k.) Next, wire up the (data) sink of the write VI to the output of the counter.



l.) Next, right click to open the VI palette, select data communication → RTI DDS toolkit → writer → release writer and wire it up in the following manner. Also, include the simple error handler which can be found in the dialog and user interface palette.



m.) Lastly, add in a OR gate, and a (unbundle by name) Vi to terminate the loop if there is some issue detected.



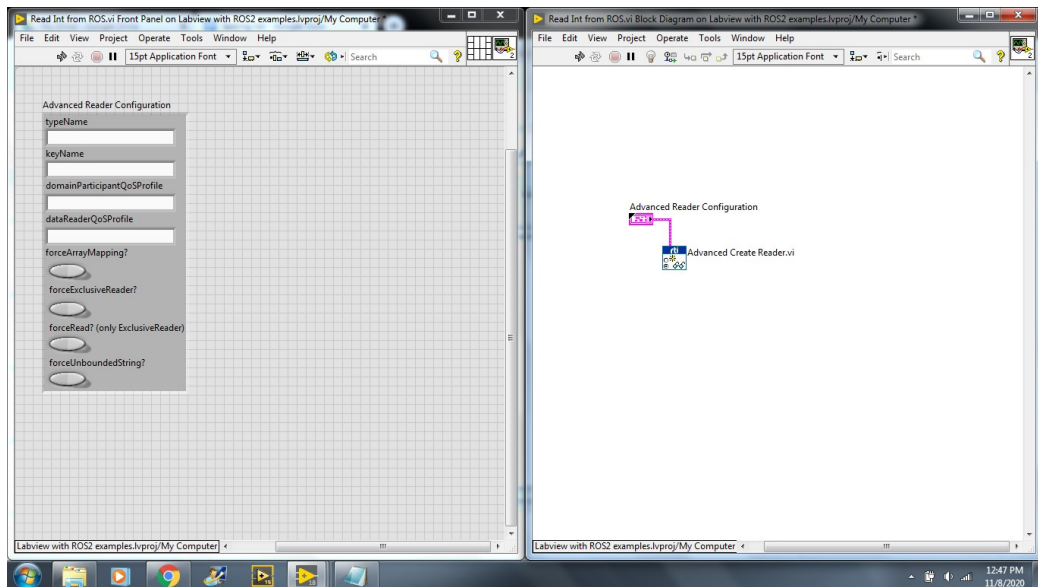
```
roald@potato: ~ 80x24
---
data: 2244
---
data: 2245
---
data: 2246
---
data: 2247
---
data: 2248
---
data: 2249
---
data: 2250
---
data: 2251
---
data: 2252
---
data: 2253
---
data: 2254
---
```

q.) Lastly, open a new terminal and key in the following lines (ros2 run ros2labview\_examples sub\_int32 ) to run the int32 subscriber node.

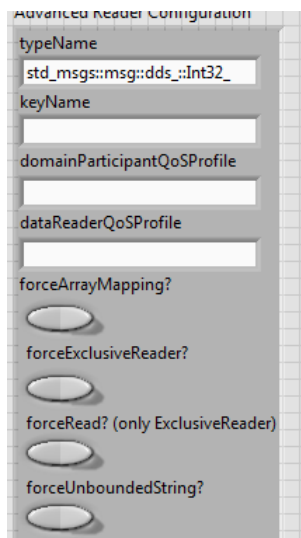
```
roald@potato: ~/ros2_ws/src/roald
[INFO] [1597117644.814814023] [int_sub]: I heard: 124
[INFO] [1597117644.914773491] [int_sub]: I heard: 125
[INFO] [1597117645.014996774] [int_sub]: I heard: 126
[INFO] [1597117645.114791304] [int_sub]: I heard: 127
[INFO] [1597117645.214927295] [int_sub]: I heard: 128
[INFO] [1597117645.314765762] [int_sub]: I heard: 129
[INFO] [1597117645.414845643] [int_sub]: I heard: 130
[INFO] [1597117645.514868409] [int_sub]: I heard: 131
[INFO] [1597117645.614941350] [int_sub]: I heard: 132
[INFO] [1597117645.714936338] [int_sub]: I heard: 133
[INFO] [1597117645.814922606] [int_sub]: I heard: 134
[INFO] [1597117645.914867475] [int_sub]: I heard: 135
[INFO] [1597117646.014944955] [int_sub]: I heard: 136
[INFO] [1597117646.115014573] [int_sub]: I heard: 137
[INFO] [1597117646.215058400] [int_sub]: I heard: 138
[INFO] [1597117646.314888970] [int_sub]: I heard: 139
[INFO] [1597117646.414970198] [int_sub]: I heard: 140
[INFO] [1597117646.514958778] [int_sub]: I heard: 141
[INFO] [1597117646.614954943] [int_sub]: I heard: 142
[INFO] [1597117646.714993999] [int_sub]: I heard: 143
[INFO] [1597117646.815047645] [int_sub]: I heard: 144
[INFO] [1597117646.914848028] [int_sub]: I heard: 145
[INFO] [1597117647.014953608] [int_sub]: I heard: 146
```

## 6.2) Publishing an integer from ROS2 to Labview

- a.) Firstly, create a new VI in your existing project, and add the advance create reader VI. (functions palette → data communication → RTI DDS toolkit → Reader → Advance create reader)
- b.) Next, click on the advanced reader configuration sink, and create a control.

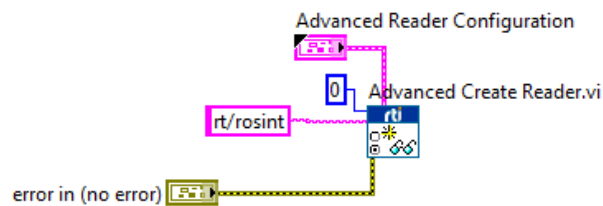


- c.) Next, on the advanced reader configuration tab, key in the type name (`std_msgs::msg::dds_::Int32_`). Once that is done, click on the edit tab and click on make current values default.

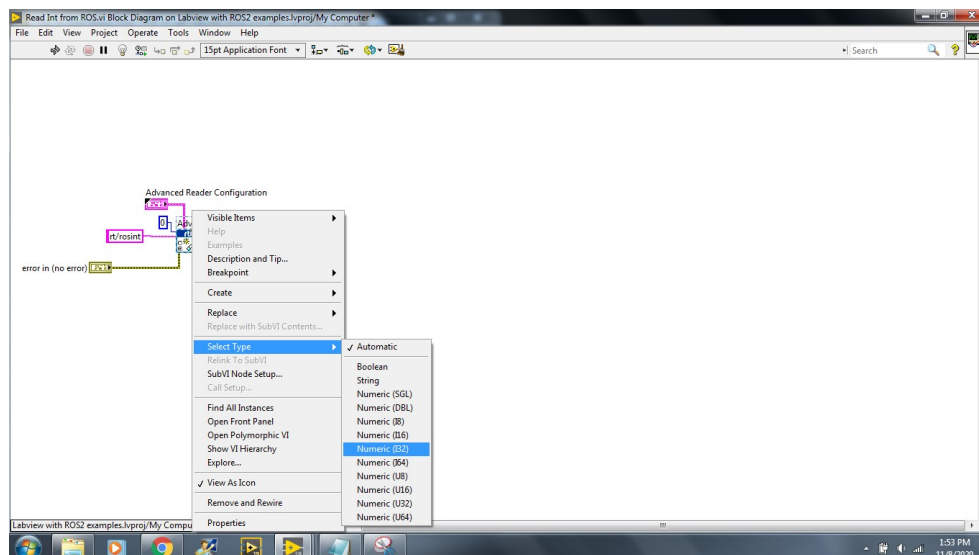




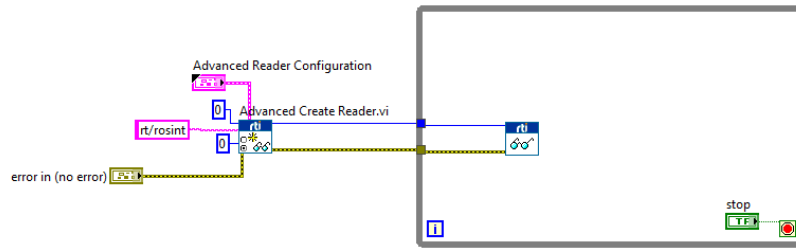
d.) Next, click on the topic name sink and create a constant. Name the constant (rt/rosint). Also, click on the domain ID sink and create a constant. Lastly, click on the error in sink and create a control.



e.) Next, right click on the advanced create writer vi → select type → select numeric(I32). Also, create a constant for the data type sink of the advanced create writer vi.



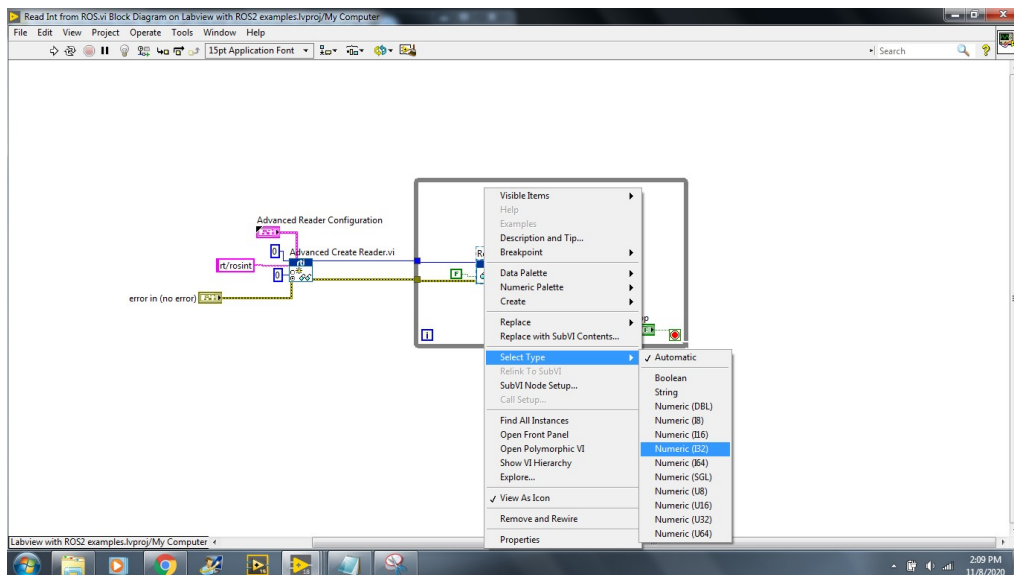
f.) Next, create a while loop and place the read vi inside it. (functions palette → data communication → RTI DDS toolkit → reader → read ). Also, wire up the DDS obj reference in and error in sinks.



g.) Next, wire up a boolean constant to the (only new samples sink). Note that the constant should be a false constant.



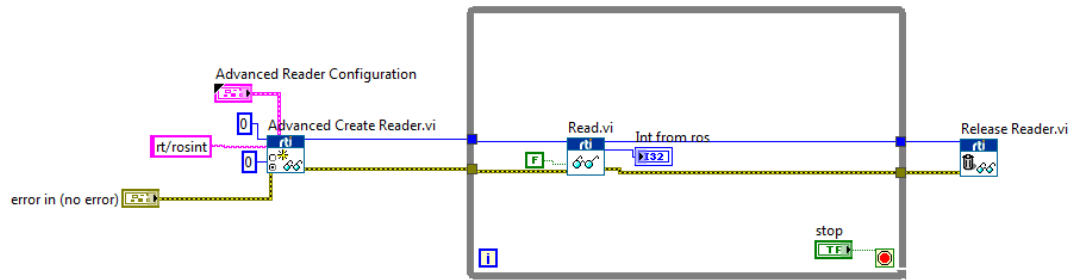
h.) Next, right click on the reader vi and click select type → numeric(I32)



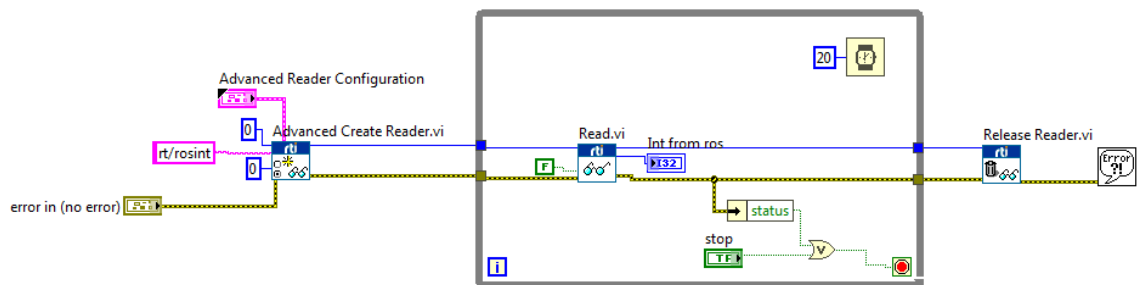
i.) Next, create a indicator for the (data) source.



j.) Next, at the end of the while loop, add the release reader vi(functions palette → data communication → RTI DDS toolkit → reader → release reader ). Connect the error and dds obj reference to the read vi.



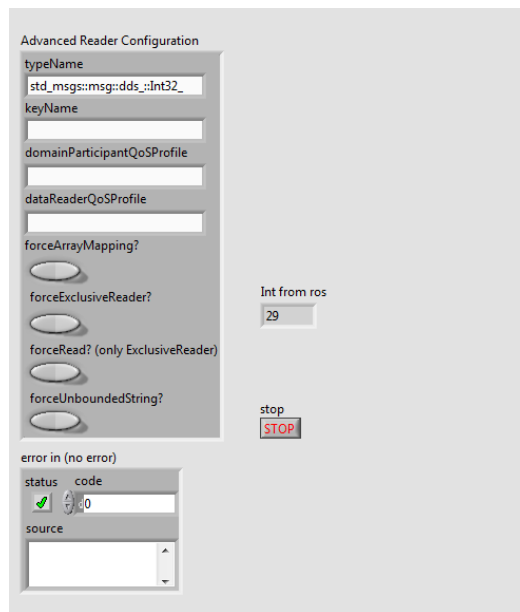
k.) Lastly, add the simple error handler vi and the while loop terminator as well as a 20ms wait block.



l.) We are now complete with our Labview read int program. Next, head on over to the ROS2 PC and open a new terminal, and run the following command (ros2 run ros2labview\_examples pub\_int32)

```
roald@potato: ~ 80x24
roald@potato:~$ ros2 run ros2labview_examples pub_int32
INFO [1597127406.508522107] [ros_publisher]: Publishing: '0'
INFO [1597127407.008584617] [ros_publisher]: Publishing: '1'
INFO [1597127407.508487938] [ros_publisher]: Publishing: '2'
INFO [1597127408.008611303] [ros_publisher]: Publishing: '3'
INFO [1597127408.508546311] [ros_publisher]: Publishing: '4'
INFO [1597127409.008632954] [ros_publisher]: Publishing: '5'
INFO [1597127409.508552397] [ros_publisher]: Publishing: '6'
INFO [1597127410.008527559] [ros_publisher]: Publishing: '7'
INFO [1597127410.508541205] [ros_publisher]: Publishing: '8'
INFO [1597127411.008582338] [ros_publisher]: Publishing: '9'
INFO [1597127411.508614133] [ros_publisher]: Publishing: '10'
INFO [1597127412.008629247] [ros_publisher]: Publishing: '11'
INFO [1597127412.508621609] [ros_publisher]: Publishing: '12'
```

m.) The publisher program will begin publishing an integer to the /rosint topic. Next, run the read int program in Labview.



Note that the integer which was sent from the ros node is now received by our Labview read program.

## 6.3) Applied Example: Simple Adder Program

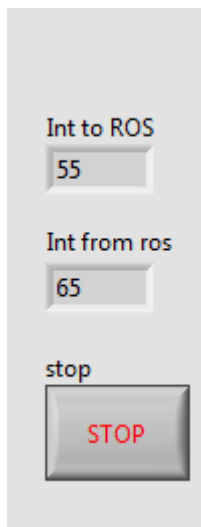
This program demonstrates two-way communication between a Labview program and a ROS2 node. The Labview program would send an integer to the ROS2 program. The ROS2 program would add 10 to the received integer value, and return the sum to the Labview program.

a.) Firstly, run the simple adder program VI, which can be found in the ros2labview\_lvprograms folder.

b.)Next, open a new terminal and key in the following command. (ros2 run ros2labview\_examples add\_int )

```
roald@potato: ~ - 80x24
[INFO] [1597217207.187767325] [ros_addint]: Publishing: '163'
[INFO] [1597217207.288114044] [ros_addint]: Publishing: '164'
[INFO] [1597217207.388009577] [ros_addint]: Publishing: '165'
[INFO] [1597217207.488006296] [ros_addint]: Publishing: '166'
[INFO] [1597217207.588108254] [ros_addint]: Publishing: '167'
[INFO] [1597217207.688007213] [ros_addint]: Publishing: '168'
[INFO] [1597217207.788119527] [ros_addint]: Publishing: '169'
[INFO] [1597217207.888014472] [ros_addint]: Publishing: '170'
[INFO] [1597217207.988032616] [ros_addint]: Publishing: '171'
[INFO] [1597217208.088051087] [ros_addint]: Publishing: '172'
[INFO] [1597217208.188008558] [ros_addint]: Publishing: '173'
[INFO] [1597217208.287976888] [ros_addint]: Publishing: '174'
[INFO] [1597217208.388194535] [ros_addint]: Publishing: '175'
[INFO] [1597217208.487991831] [ros_addint]: Publishing: '176'
[INFO] [1597217208.588116517] [ros_addint]: Publishing: '177'
[INFO] [1597217208.688081621] [ros_addint]: Publishing: '178'
[INFO] [1597217208.788053415] [ros_addint]: Publishing: '179'
[INFO] [1597217208.888055149] [ros_addint]: Publishing: '180'
[INFO] [1597217208.988150035] [ros_addint]: Publishing: '181'
[INFO] [1597217209.088098579] [ros_addint]: Publishing: '182'
[INFO] [1597217209.188211523] [ros_addint]: Publishing: '183'
[INFO] [1597217209.288000394] [ros_addint]: Publishing: '184'
[INFO] [1597217209.387945736] [ros_addint]: Publishing: '185'
```

The ros2 node will subscribe to the topic of (toadd), and send the new number to the (result) topic.



## 6.4) Publishing/Subscribing with other primitive data types

### 6.4.1) Ensuring compatible types

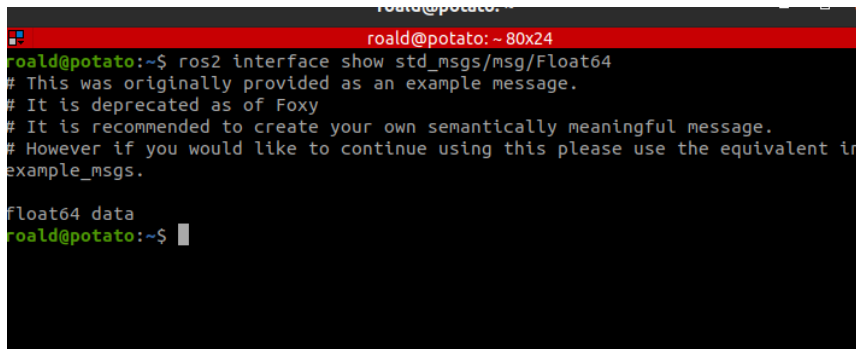
The RTI DDS Labview toolkit supports all of the primitive data types which ROS 2 has to offer. In order to pass data between ROS2 and Labview, we must ensure that the data types declared in the ROS2 message(.msg) file matches the data types specified in the Labview program. You may wish to refer to the legacy interface definition webpage for a more detailed description.

([https://design.ros2.org/articles/legacy\\_interface\\_definition.html](https://design.ros2.org/articles/legacy_interface_definition.html))

### Mapping to IDL types

ROS type	IDL type
bool	boolean
byte	octet
char	uint8
float32	float
float64	double
int8	int8
uint8	uint8
int16	short
uint16	unsigned short
int32	long
uint32	unsigned long
int64	long long
uint64	unsigned long long
string	string

For example, let's take a look at the (std\_msgs/msg/Float64) message definition. Open up a new terminal in the ROS2 PC and key in the following line (ros2 interface show std\_msgs/msg/Float64).



```
roald@potato: ~ 80x24
roald@potato:~$ ros2 interface show std_msgs/msg/Float64
# This was originally provided as an example message.
# It is deprecated as of Foxy
# It is recommended to create your own semantically meaningful message.
# However if you would like to continue using this please use the equivalent in
example_msgs.

float64 data
roald@potato:~$
```

The float64 ROS message has a float64 data type. Thus, the corresponding data type which must be declared in Labview is (double). In your Labview program, right click on the advanced create reader/writer VI → select type, and select (Numeric (DBL)). Ensure that the (Data type) sink has the correct constant attached to it. Also, ensure that the Write/Read vi has the correct type (Numeric (DBL)).

The same process can be extended to other data types (string , bool etc...) However, do note that only one data member can be specified in the ROS message. For multiple data members, please refer to the pub/sub with clusters section.

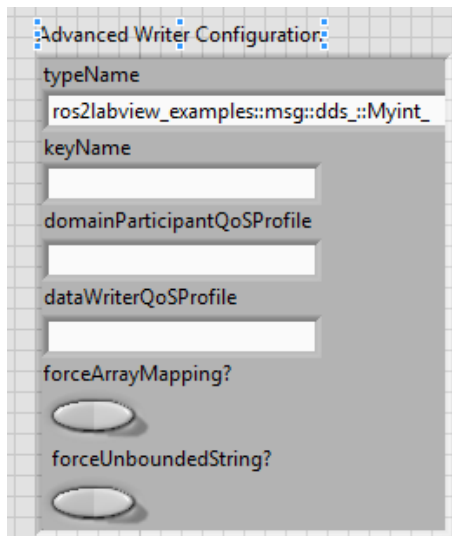
### 6.4.2) Publishing/Subscribing with custom message data types

Subscribing/publishing with custom message data types is no different from subscribing/publishing with the stock message types defined in the std\_msgs package. The only changes which have to be made are to (type name), and to the data type of rti vi's.

a.) Firstly, open the (send int to ros) vi and save it as (send custom int to ros).

b.) Next, open up a new terminal in the ROS2 pc and key in the following command (ros2 interface show ros2labview\_examples/msg/Myint ). (int64 num) Should be displayed in the window. Thus, in Labview we must ensure that our RTI Vi's have the data type of (I64).

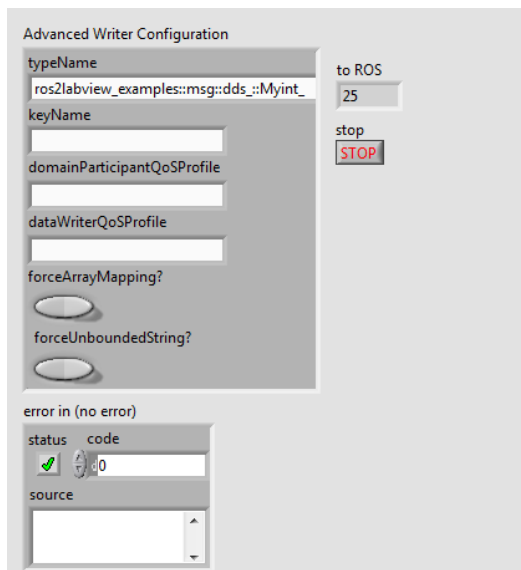
c.) Next, in our Labview program, key in (ros2labview\_examples::msg::dds\_::Myint\_) into (typeName). Also, click edit → make current values default.



d.) Next, right click on the advanced writer vi → select type → numeric I64. Also, change the representation of the data type constant to (I64).

e.) Next, change the topic name to (rt/clabint).

f.) Lastly, right click on the writer vi → select type → numeric I64. Remember to change all of the connected components to have the data type of (I64).

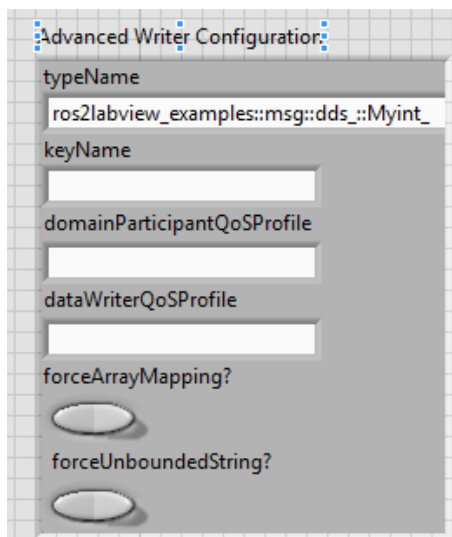


g.) Now, our custom type publisher VI will begin publishing an integer to the (clabint) topic. Open up a new terminal in the ros2 pc and key in the following line (ros2 run ros2labview\_examples sub\_custom\_int ).



```
roald@potato: ~ 80x24
[INFO] [1597235989.207980914] [cint_sub]: I heard: 127
[INFO] [1597235989.307846181] [cint_sub]: I heard: 128
[INFO] [1597235989.407890916] [cint_sub]: I heard: 129
[INFO] [1597235989.507935910] [cint_sub]: I heard: 130
[INFO] [1597235989.607786100] [cint_sub]: I heard: 131
[INFO] [1597235989.707905946] [cint_sub]: I heard: 132
[INFO] [1597235989.807954950] [cint_sub]: I heard: 133
[INFO] [1597235989.907800375] [cint_sub]: I heard: 134
[INFO] [1597235990.007785238] [cint_sub]: I heard: 135
[INFO] [1597235990.107902770] [cint_sub]: I heard: 136
[INFO] [1597235990.207882320] [cint_sub]: I heard: 137
[INFO] [1597235990.307643075] [cint_sub]: I heard: 138
[INFO] [1597235990.407747080] [cint_sub]: I heard: 139
[INFO] [1597235990.507939917] [cint_sub]: I heard: 140
[INFO] [1597235990.607826221] [cint_sub]: I heard: 141
[INFO] [1597235990.707710200] [cint_sub]: I heard: 142
[INFO] [1597235990.807829234] [cint_sub]: I heard: 143
[INFO] [1597235990.907970475] [cint_sub]: I heard: 144
[INFO] [1597235991.007935146] [cint_sub]: I heard: 145
[INFO] [1597235991.107920133] [cint_sub]: I heard: 146
[INFO] [1597235991.207810760] [cint_sub]: I heard: 147
[INFO] [1597235991.307816262] [cint_sub]: I heard: 148
[INFO] [1597235991.407857935] [cint_sub]: I heard: 149
```

- h.) The same steps could be applied to create our custom type reader program. Firstly, open the (read int from ros) vi and save it as (read custom int from ros).
- i.) Next, right click on the advanced reader vi → select type → numeric I64. Also, change the representation of the data type constant to (I64).
- j.) Next, in our Labview program, key in (ros2labview\_examples::msg::dds\_::Myint\_) into (typeName). Also, click edit → make current values default.

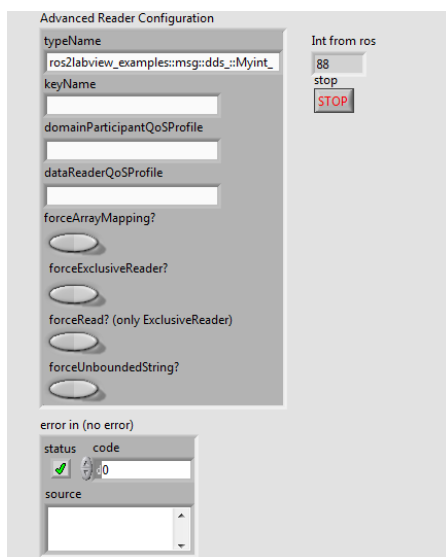


- k.) Next, right click on the reader vi → select type → numeric I64. Remember to change all of the connected components to have the data type of (I64).

l.) On the ros2 pc, open a new window and key in the following line (ros2 run ros2labview\_examples pub\_custom\_int ). A ros2 publisher will begin publishing to the rosinet topic.

```
roald@potato: ~ 80x24
roald@potato:~$ ros2 run ros2labview_examples pub_custom_int
[INFO] [1597237152.085075052] [ros_publisher]: Publishing: '0'
[INFO] [1597237152.584996325] [ros_publisher]: Publishing: '1'
[INFO] [1597237153.084941419] [ros_publisher]: Publishing: '2'
[INFO] [1597237153.585032072] [ros_publisher]: Publishing: '3'
[INFO] [1597237154.084935485] [ros_publisher]: Publishing: '4'
[INFO] [1597237154.584925617] [ros_publisher]: Publishing: '5'
[INFO] [1597237155.084970813] [ros_publisher]: Publishing: '6'
[INFO] [1597237155.584972230] [ros_publisher]: Publishing: '7'
[INFO] [1597237156.084942129] [ros_publisher]: Publishing: '8'
[INFO] [1597237156.585028022] [ros_publisher]: Publishing: '9'
[INFO] [1597237157.084957863] [ros_publisher]: Publishing: '10'
[INFO] [1597237157.584978272] [ros_publisher]: Publishing: '11'
[INFO] [1597237158.084983033] [ros_publisher]: Publishing: '12'
[INFO] [1597237158.584959781] [ros_publisher]: Publishing: '13'
[INFO] [1597237159.084946457] [ros_publisher]: Publishing: '14'
[INFO] [1597237159.584944699] [ros_publisher]: Publishing: '15'
[INFO] [1597237160.084976176] [ros_publisher]: Publishing: '16'
[INFO] [1597237160.584948706] [ros_publisher]: Publishing: '17'
[INFO] [1597237161.084982151] [ros_publisher]: Publishing: '18'
```

m.) Lastly, run the read custom int Labview program.



The Labview program should receive the number published by the ros node.

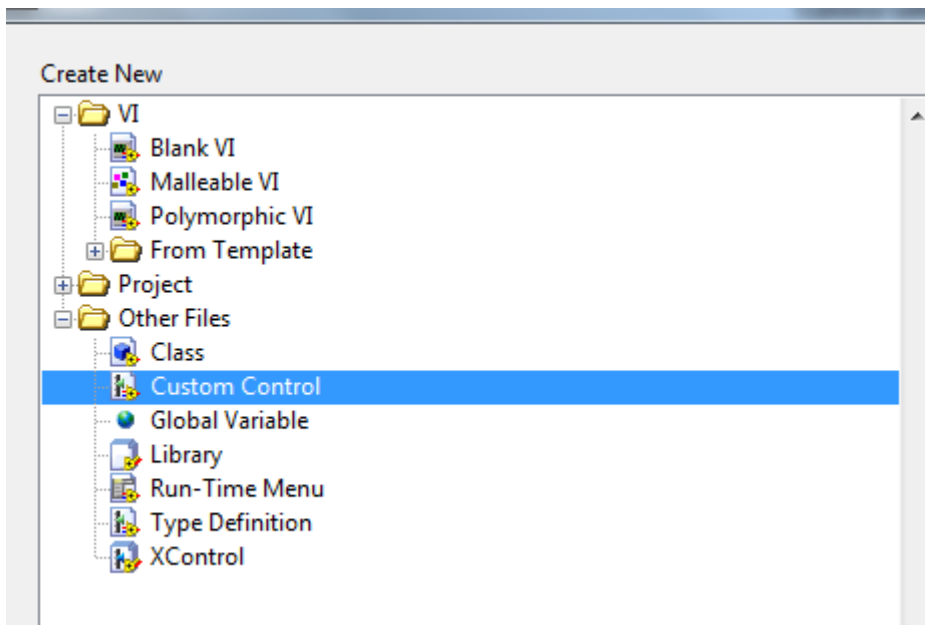
## 7.) Publish/Subscribe with Labview Clusters

In some circumstances, publishing a single variable on a topic may not be sufficient. For example, the ros (cmd\_vel) topic which is reserved for speed commands requires at least two variables (linear velocity and angular velocity) or more. We could publish/ subscribe with Labview clusters to send multiple variables (of different types if needed) on the same topic.

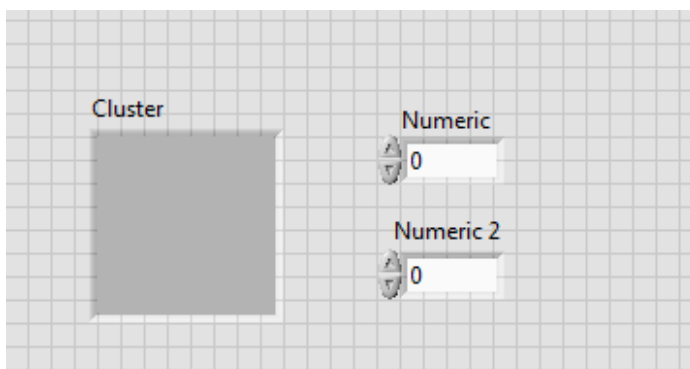
### 7.1) Publishing from Labview to ROS2 (cluster)

a.) Firstly, open up a new terminal on the ros pc and key in the following command (ros2 interface show ros2labview\_examples/msg/Clusterdemo). In this custom message type, we have two data members (int64 x) and (int64 y). Thus, we would have to re-create this type in Labview.

b.) Next, in your Labview project, create a new custom control file.

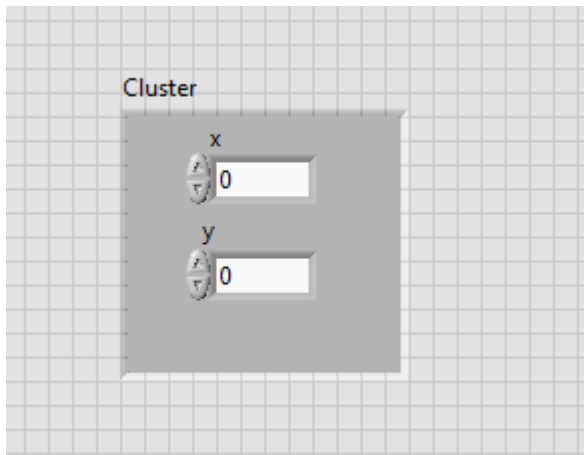


c.) Next, place down a cluster, and two numeric controls.



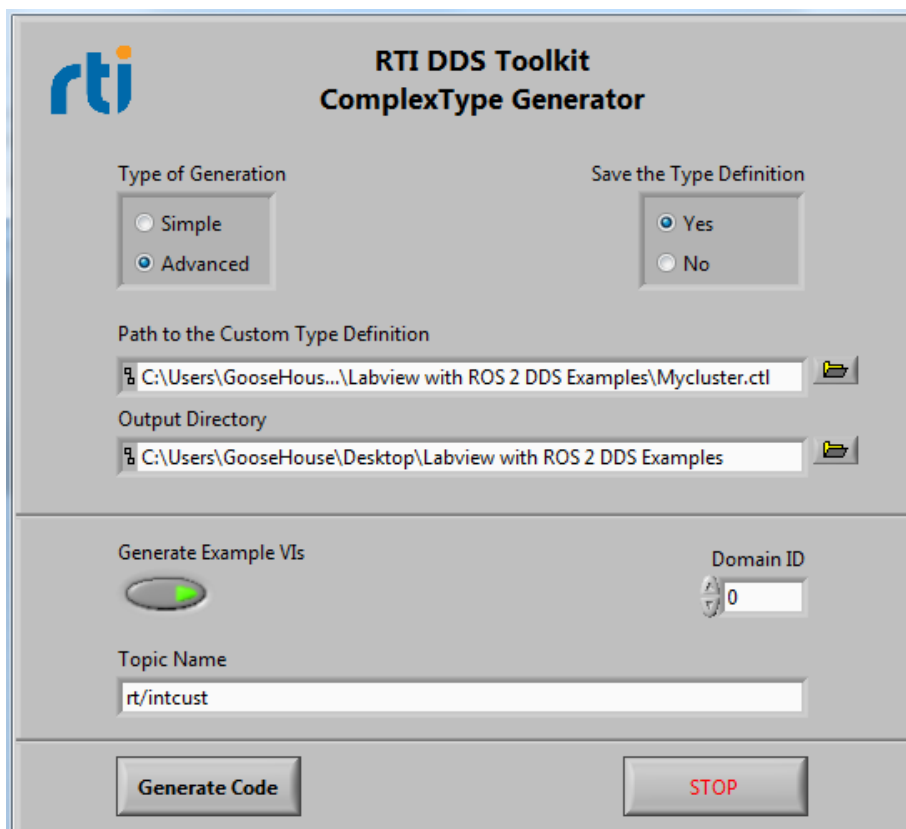
d.) Next, right click on the numeric controls → representation → select (I64) for both numeric controls.

e.) Next, rename the numeric controls as x and y, and place them in the cluster.













f.) Next, save the control as (Mycluster).

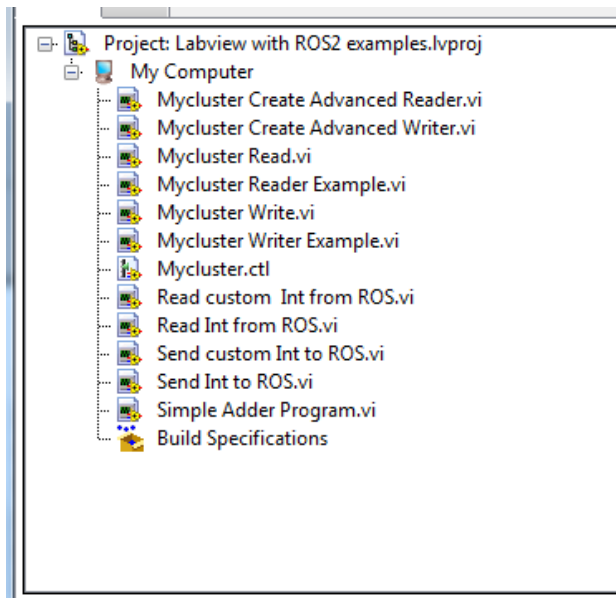
g.) Next, go to your labview project and click tools → RTI DDS toolkit → RTI DDS complex type generator. Key in the following details into the complex type generator (please specify your own path to the control file and your project folder)



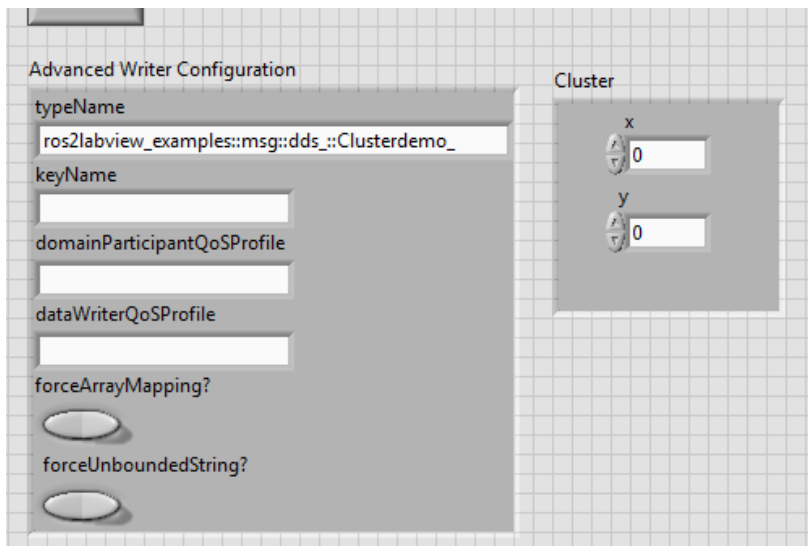
h.) Next, click generate code. The following files will be generated in your project folder.

	Labview with ROS2 examples.aliases	13/8/2020 1:16 PM	ALIASES File	1 KB
	Labview with ROS2 examples.lvpls	13/8/2020 1:16 PM	LVLPS File	1 KB
	Labview with ROS2 examples	13/8/2020 1:16 PM	LabVIEW Project	6 KB
	Mycluster Create Advanced Reader	13/8/2020 1:22 PM	LabVIEW Instrume...	31 KB
	Mycluster Create Advanced Writer	13/8/2020 1:22 PM	LabVIEW Instrume...	31 KB
	Mycluster Read	13/8/2020 1:22 PM	LabVIEW Instrume...	28 KB
	Mycluster Reader Example	13/8/2020 1:22 PM	LabVIEW Instrume...	32 KB
	Mycluster Write	13/8/2020 1:22 PM	LabVIEW Instrume...	23 KB
	Mycluster Writer Example	13/8/2020 1:22 PM	LabVIEW Instrume...	35 KB
	Mycluster	13/8/2020 1:16 PM	LabVIEW Control	5 KB

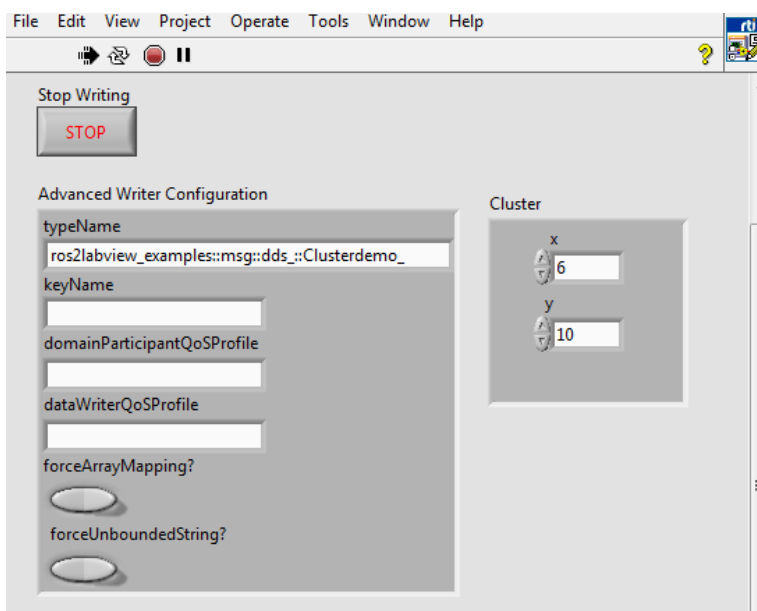
i.) Next, include these files into your Labview project.



j.) Next, open up the (Mycluster Writer Example) vi and key in the following type name. Click edit → make current values default.



k.) We are now done with the labview publisher program. Run the publisher program, and set any value to x and y.



l.) Next, on the ros2 pc, open up a new terminal and key in the following lines (ros2 run ros2labview\_examples sub\_cluster)

```

[INFO] [1597297866.138850335] [cluster_sub]: I heard: 6 and 10
[INFO] [1597297866.238809649] [cluster_sub]: I heard: 6 and 10
[INFO] [1597297866.338722180] [cluster_sub]: I heard: 6 and 10
[INFO] [1597297866.438876341] [cluster_sub]: I heard: 6 and 10
[INFO] [1597297866.538796015] [cluster_sub]: I heard: 6 and 10
[INFO] [1597297866.638861034] [cluster_sub]: I heard: 6 and 10
[INFO] [1597297866.738709598] [cluster_sub]: I heard: 6 and 10
[INFO] [1597297866.838632467] [cluster_sub]: I heard: 6 and 10
[INFO] [1597297866.938643039] [cluster_sub]: I heard: 6 and 10
[INFO] [1597297867.038799820] [cluster_sub]: I heard: 6 and 10
[INFO] [1597297867.138803162] [cluster_sub]: I heard: 6 and 10
[INFO] [1597297867.238822268] [cluster_sub]: I heard: 6 and 10
[INFO] [1597297867.338837708] [cluster_sub]: I heard: 6 and 10
[INFO] [1597297867.438895463] [cluster_sub]: I heard: 6 and 10
[INFO] [1597297867.538806444] [cluster_sub]: I heard: 6 and 10
[INFO] [1597297867.638814324] [cluster_sub]: I heard: 6 and 10
[INFO] [1597297867.738906135] [cluster_sub]: I heard: 6 and 10
[INFO] [1597297867.838882482] [cluster_sub]: I heard: 6 and 10
[INFO] [1597297867.938799071] [cluster_sub]: I heard: 6 and 10
[INFO] [1597297868.038884551] [cluster_sub]: I heard: 6 and 10
[INFO] [1597297868.138881847] [cluster_sub]: I heard: 6 and 10
[INFO] [1597297868.238860155] [cluster_sub]: I heard: 6 and 10
[INFO] [1597297868.338918380] [cluster_sub]: I heard: 6 and 10

```

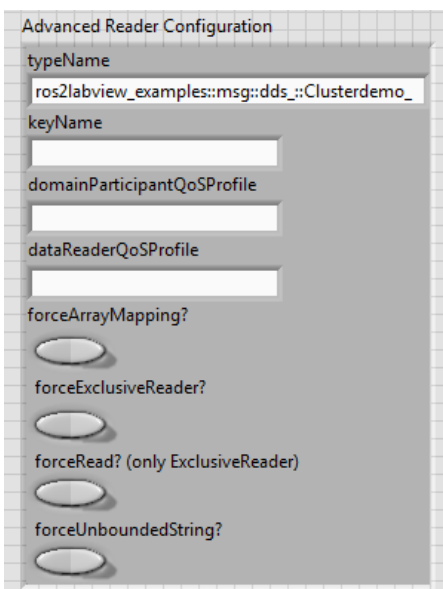
The ros node has received the data being sent from our Labview publisher program.

## 7.2) Publishing from ROS2 to Labview (cluster)

The steps required to create our Labview subscriber program is similar to the steps required to create our Labview publisher program. In fact, our subscriber program has already been created, if you have completed section 7.1.

a.) Firstly, open the (Mycluster reader example) Vi.

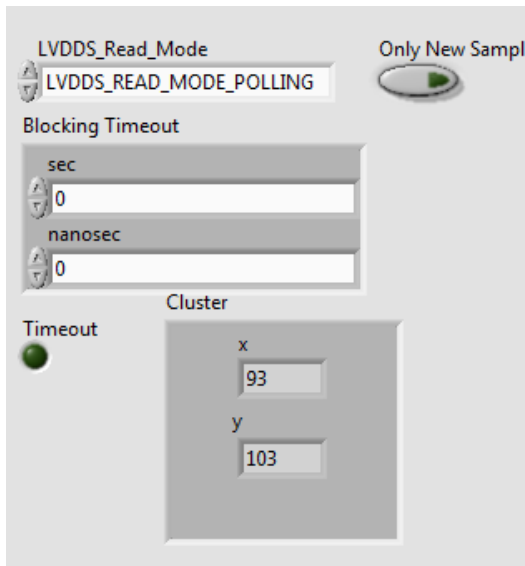
b.) Next, key in the following type name, and set all current values as default.



c.) Next, open up a new terminal in the ROS2 pc and key in the following command to run the publisher node (ros2 run ros2labview\_examples pub\_cluster).

```
roald@potato: ~ 80x24
roald@potato:~$ ros2 run ros2labview_examples pub_cluster
[INFO] [1597301138.034560866] [ros_publisher]: Publishing: 0 and 10
[INFO] [1597301138.534473421] [ros_publisher]: Publishing: 1 and 11
[INFO] [1597301139.034550764] [ros_publisher]: Publishing: 2 and 12
[INFO] [1597301139.534511893] [ros_publisher]: Publishing: 3 and 13
[INFO] [1597301140.034498802] [ros_publisher]: Publishing: 4 and 14
[INFO] [1597301140.534513907] [ros_publisher]: Publishing: 5 and 15
[INFO] [1597301141.034527489] [ros_publisher]: Publishing: 6 and 16
[INFO] [1597301141.534521917] [ros_publisher]: Publishing: 7 and 17
[INFO] [1597301142.034469752] [ros_publisher]: Publishing: 8 and 18
[INFO] [1597301142.534548174] [ros_publisher]: Publishing: 9 and 19
[INFO] [1597301143.034519824] [ros_publisher]: Publishing: 10 and 20
[INFO] [1597301143.534501122] [ros_publisher]: Publishing: 11 and 21
[INFO] [1597301144.034520227] [ros_publisher]: Publishing: 12 and 22
[INFO] [1597301144.534481279] [ros_publisher]: Publishing: 13 and 23
[INFO] [1597301145.034609783] [ros_publisher]: Publishing: 14 and 24
```

d.) Next, run our Labview subscriber program. Note that we have received the integers which was sent from our ros node.





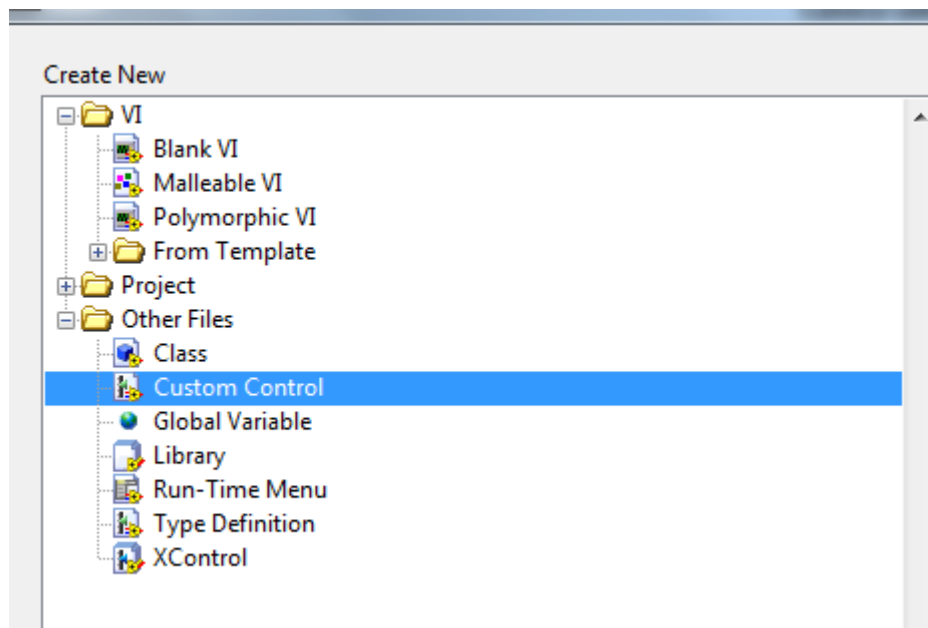
## 8.) Publish/Subscribe with Labview Arrays

### 8.1) Publishing from ROS2 to Labview (arrays)

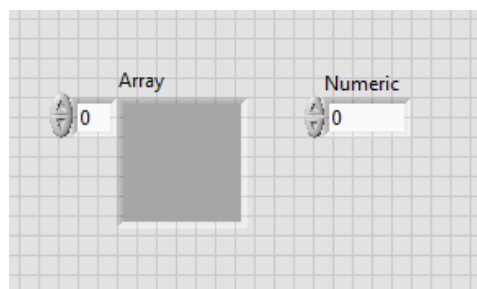
a.) Firstly, open up a new terminal in the ROS pc and key in the following command(`ros2 interface show ros2labview_examples/msg/Intarray`). (`int64[<=3]` num) is displayed.

The labview RTI dds toolkit publishes/subscribes arrays as bounded sequences. Therefore, in our ros message definition, we would have to specify our data members as bounded arrays ie. (`int32[<=5]`).

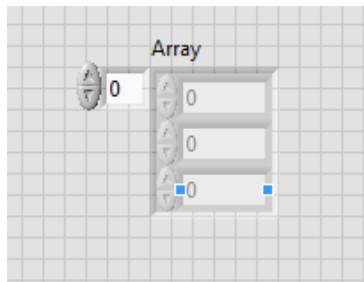
b.) Next, in your Labview project, create a new custom control file.



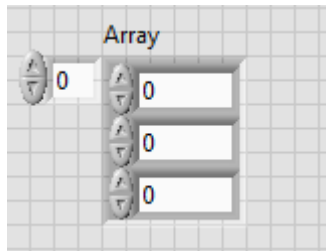
c.) Next, place an array, and a numeric control. Change the representation of the numeric control to (I64)



d.) Next, place the numeric control into the array.

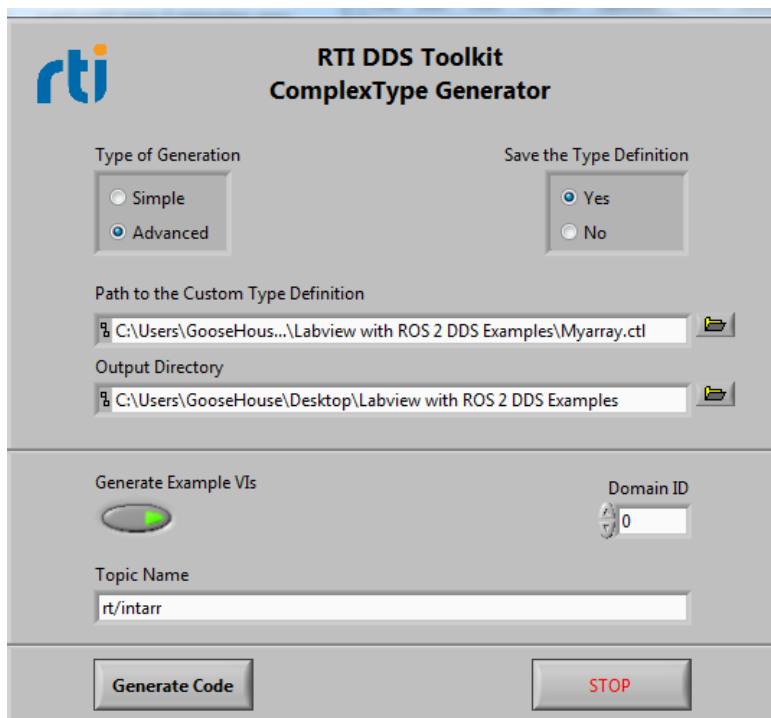


e.) Next, initialize the three elements in the array by entering 0 into each element. Note that the number of elements that we initialize in the Labview array must match the number of elements specified in the ros message(int64[<=3]).

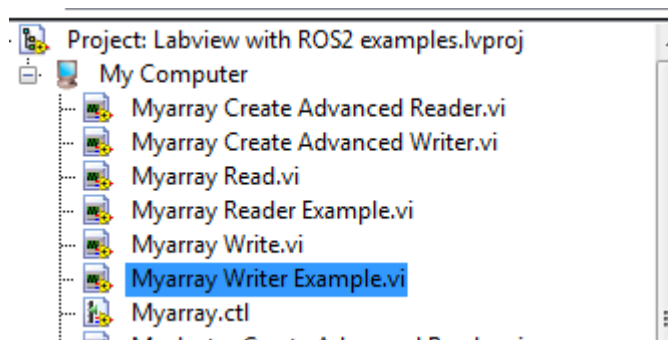


f.) Next, click edit → make current values default. Also, save the control file as (Myarray).

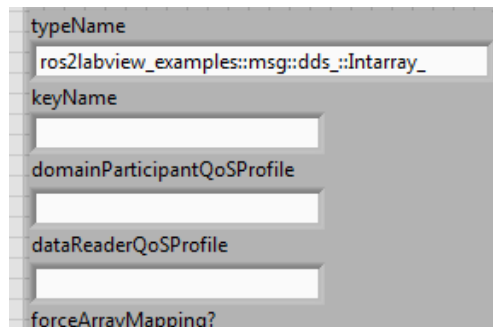
g.) Next, go to your labview project and click tools → RTI DDS toolkit → RTI DDS complex type generator. Key in the following details into the complex type generator(please specify your own path to the control file and your project folder)



h.) Once the generate code button is pressed, the following VI's will be created in your project folder. Include these files in your Labview project.



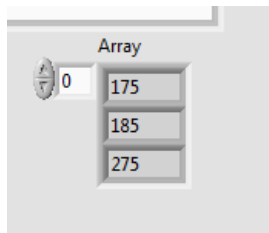
i.) Next, open the Myarray Reader Example vi and key in the following type into (typeName). Also, click edit → make current values default.



j.) Next, open up a new terminal in the ROS pc and key in the following command(ros2 run ros2labview\_examples pub\_array). The ros array publisher will start running.

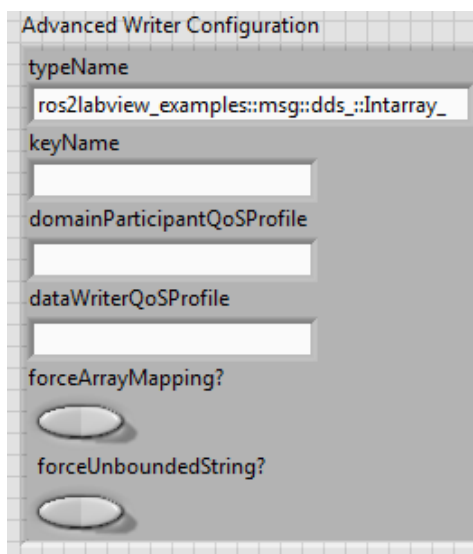
```
roald@potato: ~$ ros2 run ros2labview_examples pub_array
[INFO] [1597313656.862817716] [ros_publisher]: Publishing: 0 10 100
[INFO] [1597313657.362797381] [ros_publisher]: Publishing: 1 11 101
[INFO] [1597313657.862791138] [ros_publisher]: Publishing: 2 12 102
[INFO] [1597313658.362761165] [ros_publisher]: Publishing: 3 13 103
[INFO] [1597313658.862807459] [ros_publisher]: Publishing: 4 14 104
[INFO] [1597313659.362700082] [ros_publisher]: Publishing: 5 15 105
[INFO] [1597313659.862702632] [ros_publisher]: Publishing: 6 16 106
[INFO] [1597313660.362688937] [ros_publisher]: Publishing: 7 17 107
[INFO] [1597313660.862812254] [ros_publisher]: Publishing: 8 18 108
[INFO] [1597313661.362756172] [ros_publisher]: Publishing: 9 19 109
[INFO] [1597313661.862704766] [ros_publisher]: Publishing: 10 20 110
[INFO] [1597313662.362740669] [ros_publisher]: Publishing: 11 21 111
[INFO] [1597313662.862796024] [ros_publisher]: Publishing: 12 22 112
[INFO] [1597313663.362768949] [ros_publisher]: Publishing: 13 23 113
[INFO] [1597313663.862787346] [ros_publisher]: Publishing: 14 24 114
[INFO] [1597313664.362757709] [ros_publisher]: Publishing: 15 25 115
[INFO] [1597313664.862771217] [ros_publisher]: Publishing: 16 26 116
[INFO] [1597313665.362758984] [ros_publisher]: Publishing: 17 27 117
[INFO] [1597313665.862777651] [ros_publisher]: Publishing: 18 28 118
[INFO] [1597313666.362785087] [ros_publisher]: Publishing: 19 29 119
```

k.) Lastly, run the Myarray Reader example Vi. It should display the data sent from the ros node.



## 8.2) Publishing from Labview to ROS2 (arrays)

- a.) Firstly, open up (Myarray writer example) VI, which should have been created once you have completed section 8.1.
- b.) Next, key in the following type into (typeName), and click edit → make current values default.



- c.) Next, on the ROS pc, open a new terminal and key in the following command (ros2 run ros2labview\_examples sub\_array)
- d.) Next, run the (Myarray writer example) VI and specify some values in the array. The ros node should receive data sent from labview.

```

INFO] [1597318977.277309429] [array_sub]: I heard: 7 5 6
INFO] [1597318977.377354906] [array_sub]: I heard: 7 5 6
INFO] [1597318977.477370360] [array_sub]: I heard: 7 5 6
INFO] [1597318977.577373050] [array_sub]: I heard: 7 5 6
INFO] [1597318977.677410872] [array_sub]: I heard: 7 5 6
INFO] [1597318977.777445866] [array_sub]: I heard: 7 5 6
INFO] [1597318977.877158254] [array_sub]: I heard: 7 5 6
INFO] [1597318977.977361199] [array_sub]: I heard: 7 5 6
INFO] [1597318978.077424597] [array_sub]: I heard: 7 5 6
INFO] [1597318978.177416543] [array_sub]: I heard: 7 5 6
INFO] [1597318978.277440065] [array_sub]: I heard: 7 5 6
INFO] [1597318978.377367889] [array_sub]: I heard: 7 5 6
INFO] [1597318978.477387722] [array_sub]: I heard: 7 5 6
INFO] [1597318978.577411610] [array_sub]: I heard: 7 5 6
INFO] [1597318978.677497143] [array_sub]: I heard: 7 5 6
INFO] [1597318978.777582846] [array_sub]: I heard: 7 5 6
INFO] [1597318978.877410741] [array_sub]: I heard: 7 5 6
INFO] [1597318978.977210039] [array_sub]: I heard: 7 5 6
INFO] [1597318979.077309654] [array_sub]: I heard: 7 5 6
INFO] [1597318979.177490993] [array_sub]: I heard: 7 5 6
INFO] [1597318979.277618905] [array_sub]: I heard: 7 5 6
INFO] [1597318979.377346777] [array_sub]: I heard: 7 5 6
INFO] [1597318979.477512194] [array_sub]: I heard: 7 5 6

```