

Using a LabVIEW dashboard with a Java/C++ robot

Joe Ross
Team 330

1/11/2010

Table of Contents

Introduction.....	2
Important Notes.....	2
Using the default dashboard.....	2
Adding your own data to the dashboard.....	4
Modifying LabVIEW dashboard.....	4
Creating Dashboard Datatype.ctl.....	5
Displaying the custom data.....	7
Dashboard Datatype order.....	9
Building Executable.....	10
Java code.....	11
Running the program.....	12
Additional Topics.....	13
Graphs.....	13
Types of dashboard data.....	13
Fitting more things on the screen.....	14

Introduction

This document was written by a LabVIEW programmer in an attempt to demystify LabVIEW enough that a Java/C++ programmer can integrate a Java/C++ robot with a LabVIEW dashboard. This document covers Java explicitly, but because of the similarities of the libraries, it should be easy to implement the same things in C/C++. It covers Dashboard data only, not the camera. There should be examples in both C++ and Java for sending camera data (I have not tried it).

If your programmers are familiar with GUI programming and network communication you may wish to write a dashboard in the language of your choice. That is beyond the scope of this document, but the following information may help you along the right path.

<http://www.chiefdelphi.com/forums/showthread.php?t=72874>

Team 1073's presentation (should be posted soon on their website)

<http://www.chiefdelphi.com/forums/showthread.php?t=75261>

LabVIEW is a graphical programming language that is designed to make it easy for engineers to write programs. It makes building GUIs easy (as well as doing a lot of complex data analysis). The default dashboard that is installed on the classmate is built in LabVIEW, and the source is included. It is not that hard to modify to add your own data, but can be daunting to someone who's never used LabVIEW before. This document will walk you through modifying the LabVIEW dashboard to add custom data and modifying your Java program to send both the default data and your custom data.

Important Notes

This document was written based on the last beta update of both LabVIEW and Java. However, the released versions differ significantly. Fixing this document for all the changes is a bigger task than I can take on during the build season, so I am releasing this as is. I did make a few quick updates on things I noticed and were easy to change, but for the most part it is left as an exercise to the reader to make the appropriate changes.

Using the default dashboard

Since the classmate has a built in dashboard, the first step is to get the default dashboard data populated.

There is an example project that sends data to the default dashboard. In Netbeans create a new example dashboard project as shown in Figure 1.

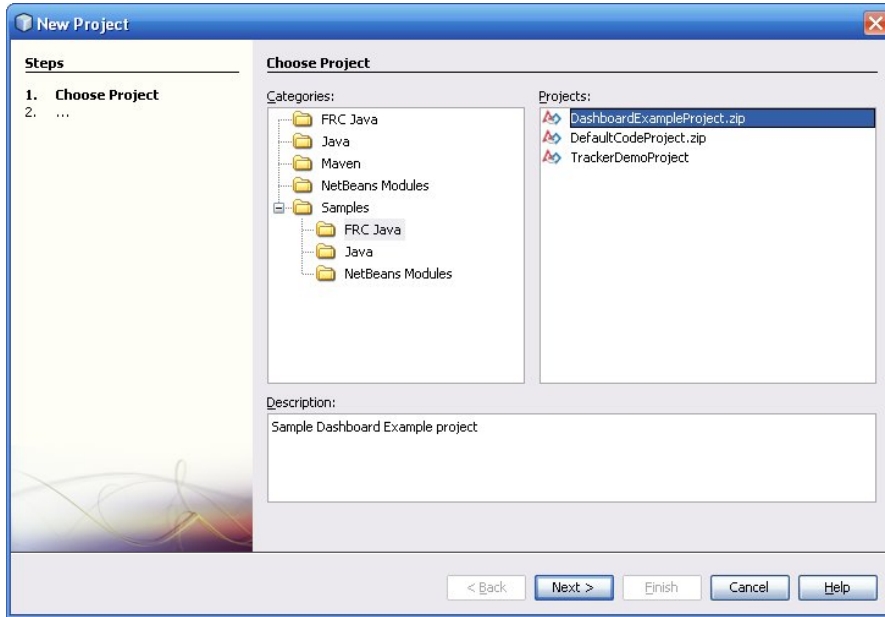


Figure 1: Dashboard Example Project

The project is a bare bones IterativeRobot project. It defines a function called `sendDashboardData` which contains the code needed to collect and send the default dashboard data. All the example program does is send the data in `teleopPeriodic`. You can load this project into your robot directly, but it isn't that interesting, as you can only see the inputs on the robot change, and you can't control anything. However, you can copy the `sendDashboardData` function to your own code, and call it periodically.

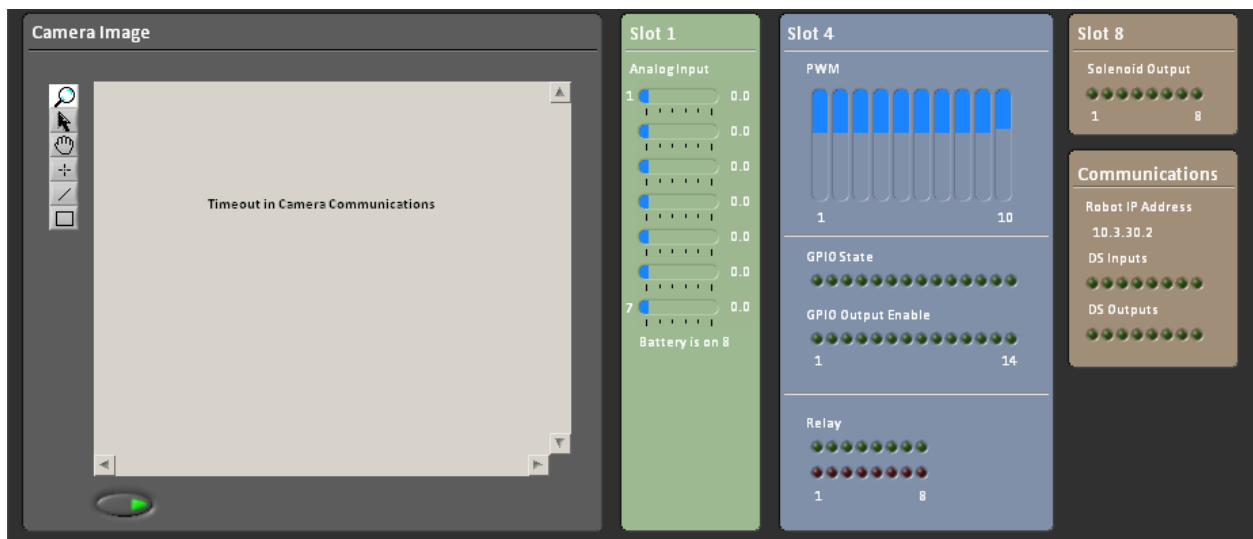


Figure 2: Default Dashboard

This covers the I/O data on the right of the default dashboard as shown in Figure 2.

Adding your own data to the dashboard

There are 3 steps to adding your own data to the dashboard. First, you define the data you want to send in LabVIEW. Then you write the code to send the data using Java/C++. Finally, you can choose how to display the data in LabVIEW.

For this example, we'll send one of each data type that the dashboard protocol supports. The data types are listed below.

Name	Description
Byte	8 bit integer
Short	16 bit integer
Int	32 bit integer
Float	32 bit floating point number
Double	64 bit floating point number
Boolean	1 bit value packed into a byte
String	Null terminated array of bytes, or a array of bytes with a specified length
Array	An array can contain any of the other data types
Cluster	Equivalent to a struct in C

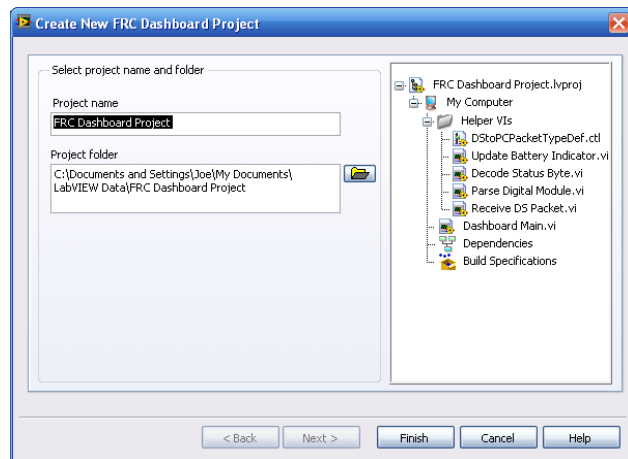
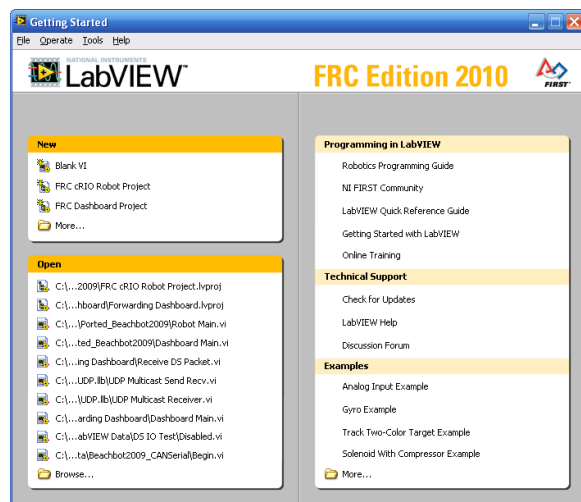


Figure 3: LabVIEW Getting Started Window Figure 4: Creating a Dashboard project

Modifying LabVIEW dashboard

To start, open LabVIEW. When it gets to the Getting Started Window as shown in Figure 3, choose New FRC Dashboard project from the upper left box. In the create new project window (Figure 4), enter a new name in the project name box and change the folder, if desired.

Press finish and wait for LabVIEW to finish configuring the project. Expand the Helper Vis folder in the Project Explorer so it looks like Figure Error: Reference source not found. Double Click on Dashboard Main.vi and you'll get a window that looks like Figure 2.

A LabVIEW VI (Virtual Instrument) contains both the GUI (Front Panel) and the code (Block Diagram). When you open a VI, the first thing that opens is the front panel. To switch to the Block Diagram, go to the Windows menu and select Show Block Diagram (or press CTRL → E). The block diagram looks like figure 5.

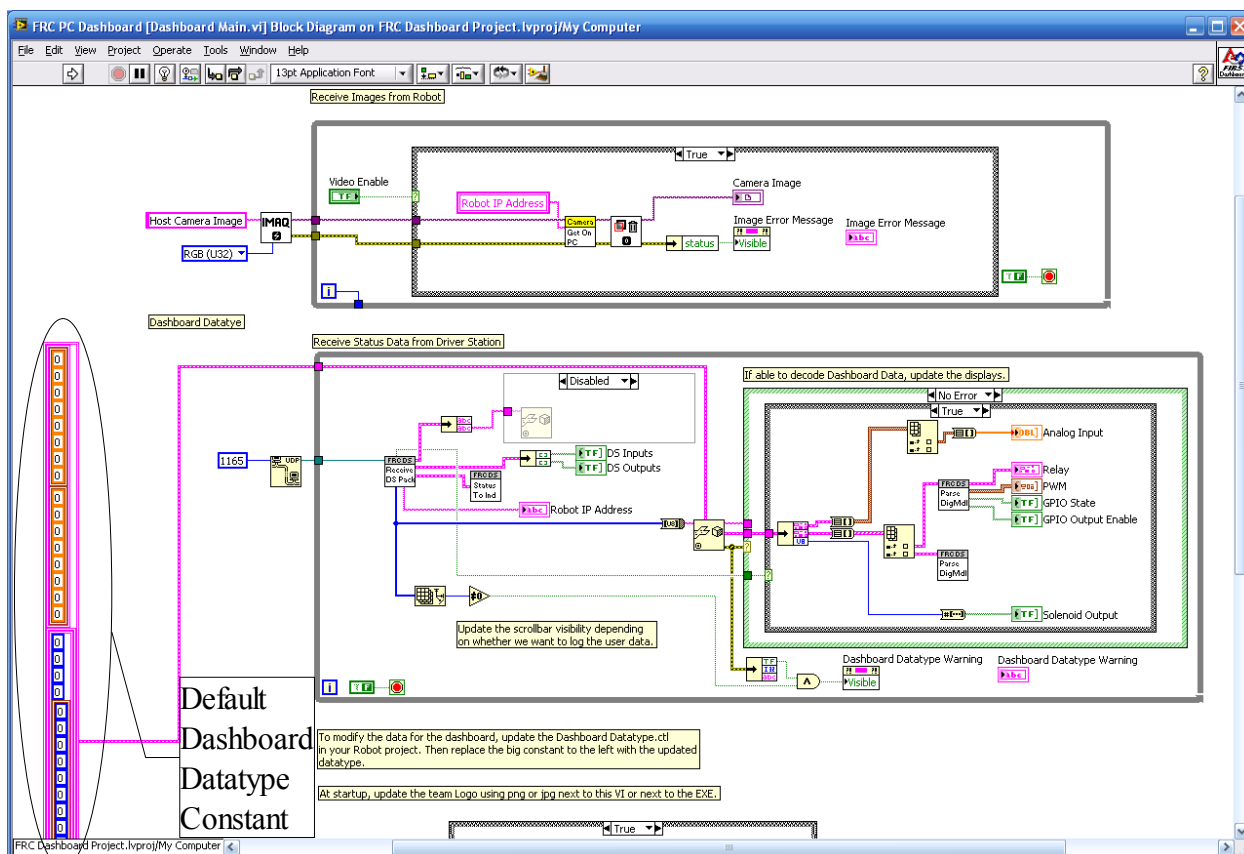


Figure 5: Block Diagram of Dashboard Main.vi

Creating Dashboard Datatype.cti

By default, the dashboard uses a constant to define the order of the dashboard data. We want to make it easier to modify, so we will turn it into a typedef. Click on the pink border of the Dashboard Datatype Constant (as identified in Figure 5). Its easiest to do get it at the top right of the constant. Copy the constant (Ctrl-C or Edit → Copy).

Switch to the Project Explorer (Figure Error: Reference source not found). Right Click on the Helper VIs folder and select New->Control). A new window is created with your new control. Select Edit->Paste to paste the copied constant into the new control. On the toolbar, change the pulldown from Control to Strict Type Def. The window should look like Figure 6.

Save the new control, giving it the filename Dashboard Datatype.ctl.

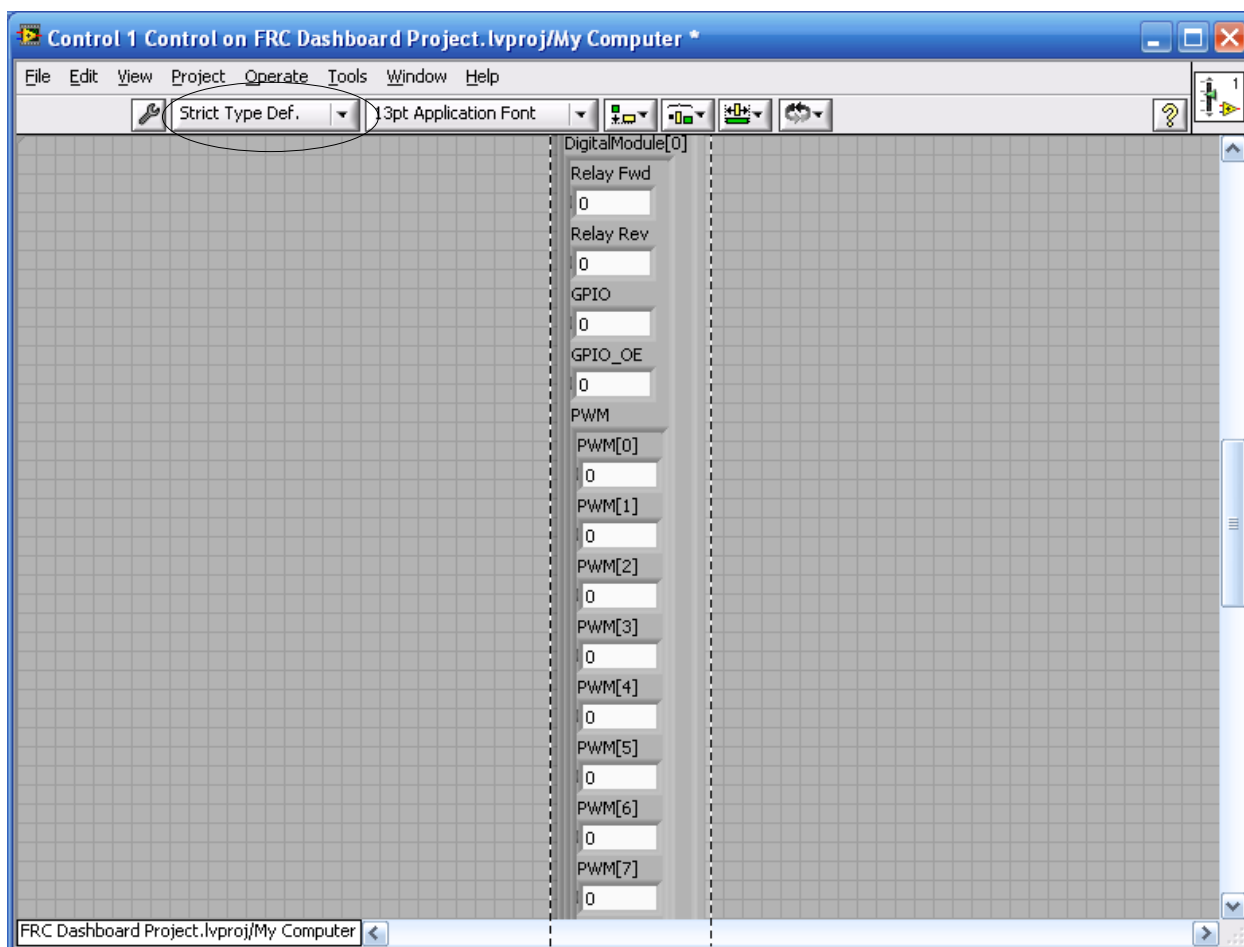


Figure 6: Dashboard Datatype Control

Change back to the Dashboard Main.vi. Select the same constant as before and delete it. Notice that the existing wires from the constant will turn into a dashed line. Position the Project Explorer so you can see it also. In the Project Explorer, select Dashboard Datatype.ctl and drag it to the Dashboard Main.vi near where the old constant used to be. It will create a new constant that is linked to the Dashboard Datatype.ctl typedef. When you mouse over the constant, a small pink pigtail will appear. If you put the mouse over the pigtail, the cursor will turn into a wire spool. While it is a spool, left click, and then move the cursor to the dashed line. When the dashed line starts flashing, left click again to finish drawing the wire. The wire should turn pink. Most likely, there will still be some part of the old line. Press Ctrl-B to clean that up. The Block Diagram should once again look like Figure 5.

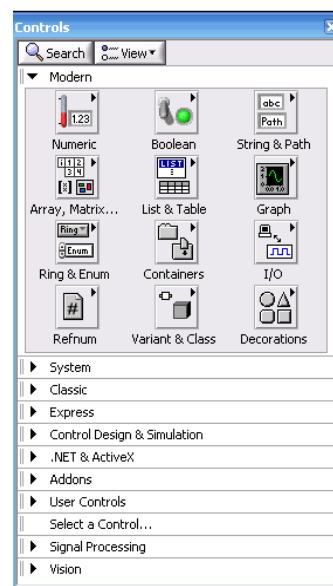


Figure 7: LabVIEW Palette Category & Text

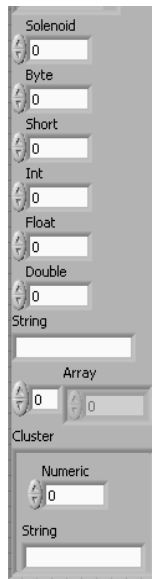
The next step is to start adding new data to Dashboard Datatype.ctl. Open Dashboard Datatype.ctl (See Figure 6). Right click on an empty area of the screen and a menu of items you can add will pop up. Click the push-pin in the upper left corner to keep the palette visible. Depending on how LabVIEW is configured, it may look like Figure 7 or may look slightly different. To make it easier to follow, if the palette does not already look like Figure 7, click on the View button and navigate to View this Palette as → Category (Icons & Text).

On the front panel, Scroll down to the bottom of the cluster. In the palette, navigate to Modern → Numeric → Numeric Control and click to choose the Numeric Control. Click again in the empty space to the right of the box labeled solenoid to add the Numeric Control. It will automatically move to the bottom of the cluster and be named Numeric. Double click on the name Numeric to select the text and replace it with Byte.

LabVIEW treats most type of numeric data types the same, so you can use the same displays for both an integer and a floating point value. To tell LabVIEW that the numeric control we just added should be an 8 bit integer, right click on the indicator and choose Representation → U8 (for unsigned 8 bit integer). You can also use S8 for signed 8 bit integer, just make sure LabVIEW and your Java/C++ code match signed or unsigned.

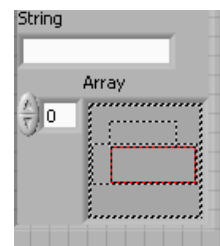
Add 4 more numeric controls and name them Short, Int, Float, and Double. Change the representation of the indicators to U16, U32, SGL, and DBL respectively.

Next we'll add a String. In the palette, navigate to Modern → String & Path → String Control. Add the String Control to the cluster, and it will be called String.



*Figure 9:
Finished
Dashboard
Datatype*

Next we'll add an array. The first thing to do is add an array box. Navigate to Modern → Array, Matrix, & Cluster → Array. This drops the Array container, but LabVIEW needs to know what data type to put in the array. Like most other languages, you can have an array of numbers, or strings, or just about any other data type. For this example, we'll use numbers. Drag a numeric control into the empty area inside the array (See Figure 8). This sets the type of the array.



*Figure 8: Adding
Array Type*

Another data type that can hold multiple items is a cluster. Unlike the array, it can hold different types of data, like a struct in C. In the Palette, navigate to Modern → Array, Matrix, & Cluster → Cluster. Like the Array, you drag and drop elements inside the cluster to add them. Add a Numeric Control and a String Control. You may have to resize the cluster to get everything to fit nicely.

Displaying the custom data

After everything is added, it should look like Figure 9. You can close Dashboard Datatype.ctl (be sure to save it).

Open Dashboard Main.vi again and compare how Figure 10 looks compared to Figure 5. You'll notice that there are several more items in the Unbundle function on the left. Each of the new items correspond to the items you added to the Dashboard Datatype.

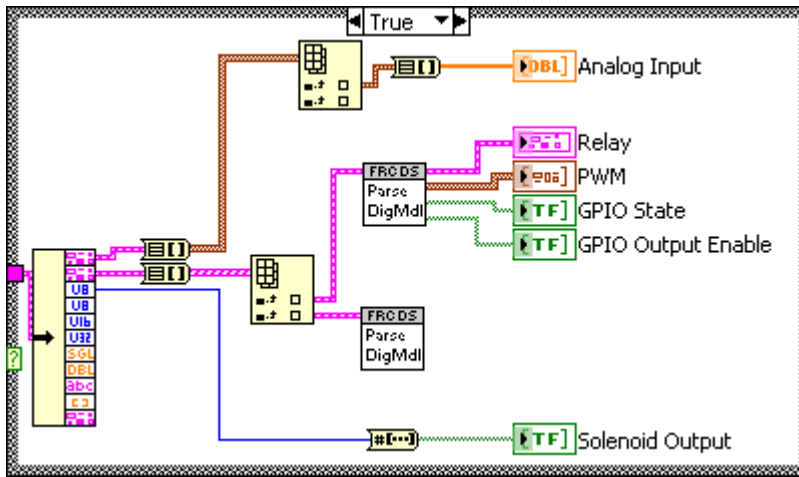


Figure 10: Expanded Dashboard Datatype

The next thing we'll do is display the new data. First thing is to make some more room on the block diagram. Grab the border of the case structure in Figure 10 and drag it down to create more room. After there is room, right click on each unconnected piece of data and choose create → Indicator. This will create an indicator that appears on the front panel. Sometimes, the indicator will end up on top of your other code. Grab it and drag it to an empty spot. Repeat this for the other items. When you're done, it should look something like Figure 11.

Adding those indicators to the block diagram also added them to the front panel. Switch to the front panel and scroll around to look for them. If you can't find them, switch back to the block diagram and double click on one of the indicators that you added. It will switch back to the front panel and highlight the indicator. Take all the of the indicators and drag them to the empty space to the right of the default dashboard. Each of the indicators has a label, but it is black text on a black background and can be hard to see. To change the background color, go to View → Tools Palette. The tools palette will pop up (Figure 13). Click on the paint brush at the bottom and then click on an empty area of the front panel. That will turn the

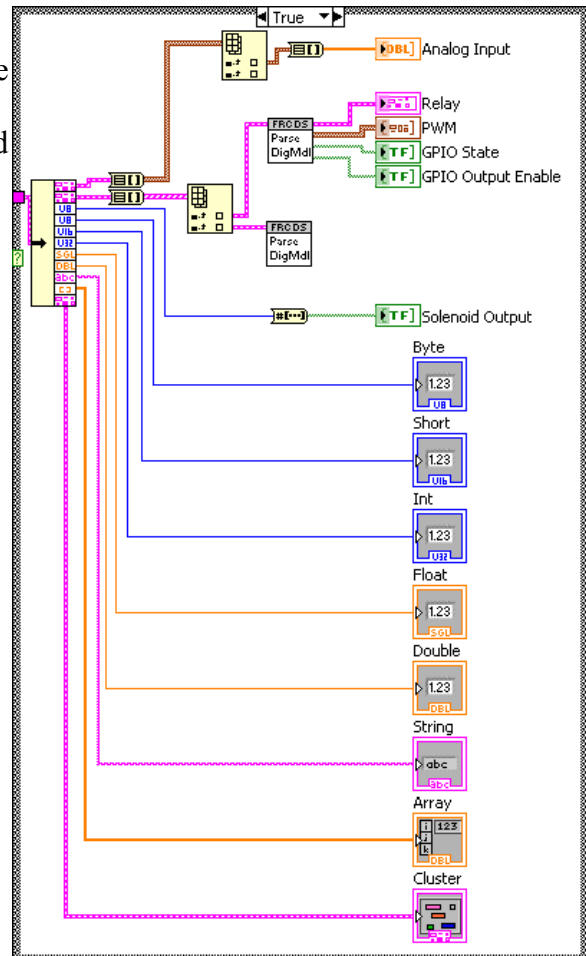


Figure 11: Indicators for added data

and change the type to “Adapt to Type” (See Figure 16). Draw a wire from the input of the unbundle function to the white box on the bottom left of the Call Library Function Node (See Figure 15). Right click on the Call Library Function Node and select Generate C Code and save the file wherever you like. You can review that file and see the order that LabVIEW placed everything in.

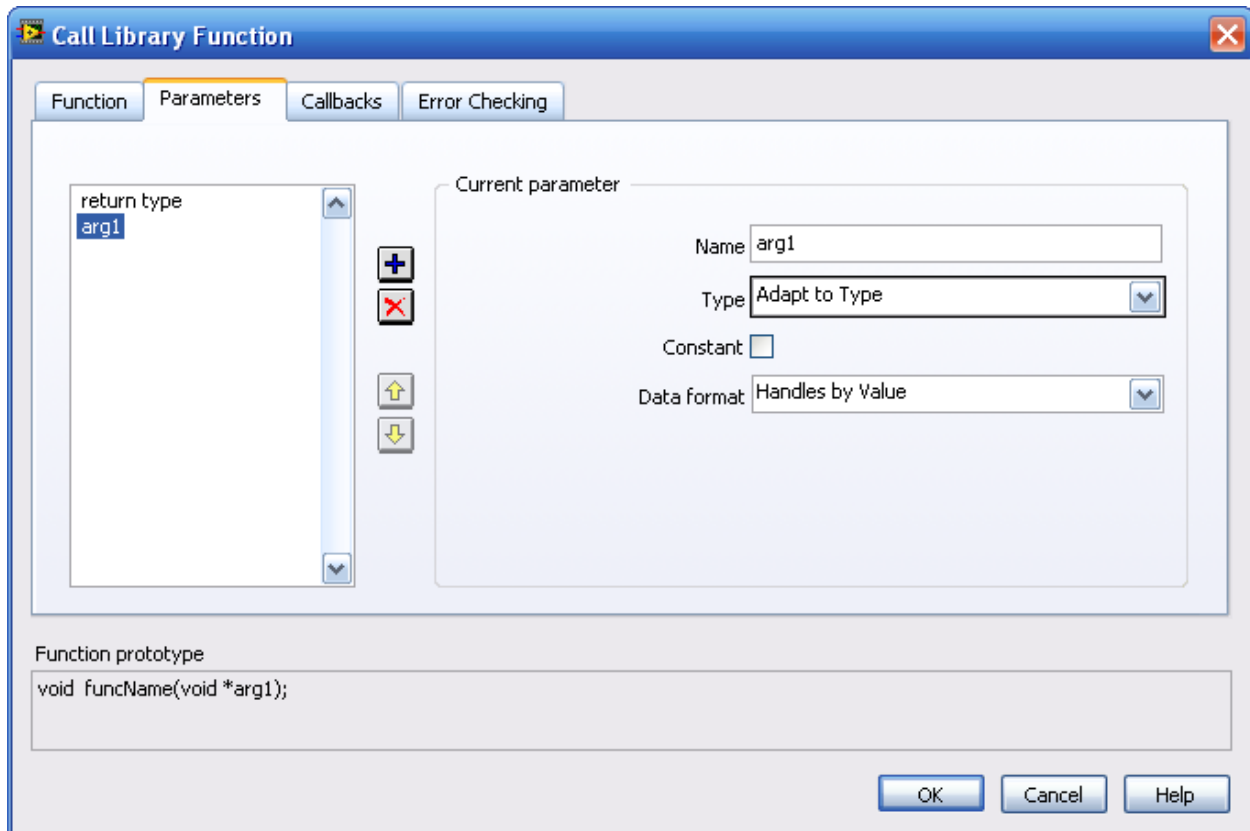


Figure 16: Configuring the Call Library Function Node

Building Executable

LabVIEW can build an executable that you can load on the classmate in place of the default dashboard executable. It will run on a computer that has Labview installed, or that has the [LabVIEW Runtime Engine](#) installed.

In the LabVIEW Project Explorer, expand Build Specifications and right click on FRC PC Dashboard and select Build. By Default, it will create a builds directory one directory higher then your project. For example, if your LabVIEW project is in <stuff>\Dashboard, the exe would be in <stuff>\builds\Dashboard\FRC PC Dashboard.

You can move the exe to the classmate. To avoid overwriting the default dashboard, you can put the executable anywhere you want. To point to your new executable, edit C:\Documents

and Settings\All Users\Documents\FRC DS Data Storage.ini and make the line starting with DashboardCmdLine= point to your executable.

Java code

Now that the LabVIEW dashboard has been set up, it's time to write the java code that will add the custom data. As a starting point, open the Dashboard Example (or your own code, if you added the Dashboard Example code to it).

In the updateDashboard function, scroll to the bottom and find where it sets the solenoid data. As that is the last thing in the default dashboard, that is where we are going to add code for our new data. To simplify the example, we are just going to send constants. However, to be usefull, you should add variables from your own code. Adding the numeric data is fairly easy, just call add<variable type>. If you recall, the first 5 numeric values were of type byte, short, int, float, and double. The code to send data is shown in Code 1. Note that since Java's default data types are int and double, the constants must be cast to the appropriate data type to avoid a warning.

```
lowDashData.addByte((byte)1);  
lowDashData.addShort((short)2);  
lowDashData.addInt(3);  
lowDashData.addFloat((float)4.5);  
lowDashData.addDouble(6.7);
```

Code 1: Numeric Data Types

Adding a string is also fairly simple, just addString.

Adding an array is a little more complex. The first part of a LabVIEW array is the length of the array. Rather than making you keep track of that, the dashboard class does it automatically. However, that means you have to help it by telling it when the array starts and ends. Before the first data, you need to call addArray and after the last piece of the array, you need to call finalizeArray. In between those, you call the appropriate add command for the data type of your array. See Code 2.

```
lowDashData.addArray();
{
    for(int i=0;i<4;i++)
        lowDashData.addDouble(i);
}
lowDashData.finalizeArray();
```

Code 2: Array

Adding a cluster is basically the same as an array. You start with `addCluster` and finish with `finalizeCluster`. In the middle, you add the datatypes that you defined in LabVIEW. See Code 3 for the cluster we defined earlier, containing a double and a string.

```
lowDashData.addCluster();
{
    lowDashData.addDouble(330);
    lowDashData.addString("Beta Test");
}
lowDashData.finalizeCluster();
```

Code 3: Cluster

Running the program

Download the java program and run the LabVIEW dashboard, and you'll see all the data become populated once you switch to teleoperated enabled mode. See Figure 17.

Congratulations, you now should know enough to make a simple LabVIEW dashboard and communicate with it with Java or C++ code.

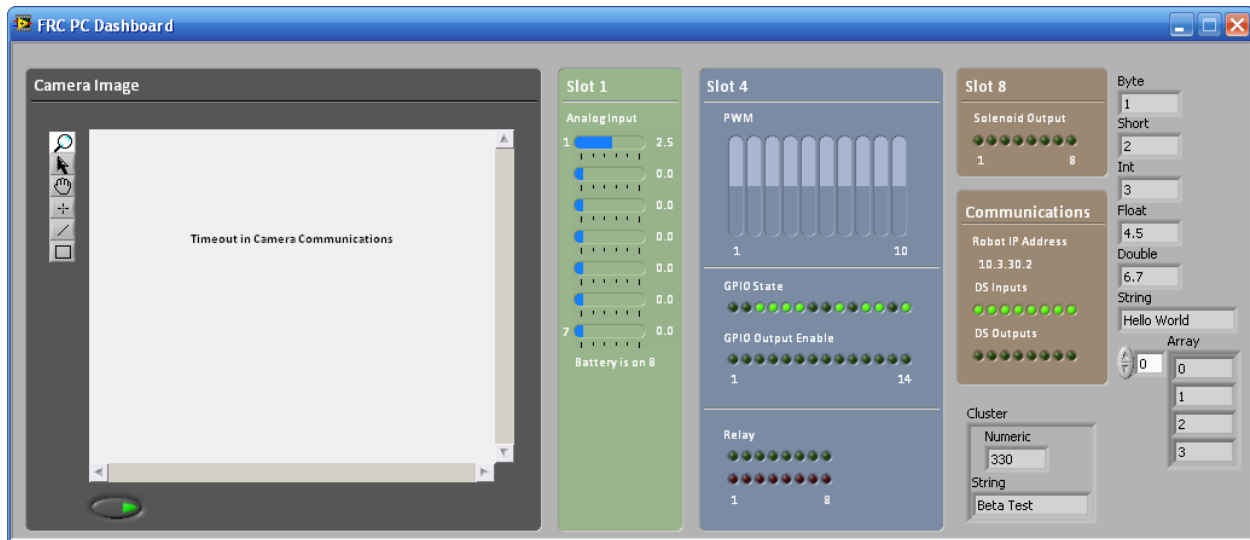


Figure 17: Finished dashboard with data populated

Additional Topics

Here are some hints on some additional things that you can do with the dashboard.

Graphs

LabVIEW makes it easy to graph data also. You can turn any numeric indicator into a graph by right clicking on the indicator and choosing Replace → Graph → Waveform Chart. You can right click on the chart and choose properties to change all kinds of things.

Types of dashboard data

The Dashboard protocol has 984 bytes that are available to teams. This is broken up into high priority data, error strings, and low priority data. These can be any size, as long as the total is 984 bytes or less. In order to make sure the error strings don't preempt your data, it might make sense to send your data as high priority.

The data sent in this example is all in the low priority dashboard data. The 2010 dashboard uses some high priority data (the game specific stuff) and some low priority data (the I/O). The dashboard example only uses low priority, but the Tracker example uses both.

To decode the high priority data in LabVIEW, you would do similar steps for the 2010 game specific IO, turn it into a typedef, etc. The data that is currently there is output to an indicator that is off the screen. You can add a unbundle hanging off that indicator to access the data. If you delete the indicator, some of the camera data

Fitting more things on the screen

The classmate has a pretty small screen for the front panel. One way to display data differently is to use a tab control. Right click on the front panel and navigate to Modern → Containers → Tab Control. Draw the tab control just smaller than the size of the screen. You can then move your displays into the various tabs, and add more tabs if necessary.

If you aren't using the classmate for your dashboard, you can simply make the window bigger.

It's unlikely you'll use all the I/O available to you, so you can also save screen space by deleting the I/O you don't want (you may need to ungroup things on the front panel before you can delete individual items). You can also stop sending them to save more room for the data you want.