Application Note # 4414

Using LabVIEW™ with Galil Controllers

By combining the ease of use of Galil controllers with the simplicity of programming in LabVIEW™, a professional looking graphical user interface can be created in minutes. Galil offers two methods of interfacing to our controllers in LabVIEW. The first one uses the ActiveX toolkit that can be purchased from Galil. It takes advantage of the ActiveX support offered in LabVIEW 5.0 or greater. This package provides the tools needed to get up and running quickly saving costly programming time.

The second method uses direct function calls to Galil library files (.dll). This method can be more flexible, but can also involve much more programming. Some free example VIs (Virtual Instruments) have been provided that provide all the basic functions for our controllers. More advanced functions can be accessed by creating your own VI function calls.

This application note will take the user through the basic setup of a sample VI, including connecting to a Galil controller, sending commands, receiving responses, and closing communication. Part 1 deals with using the ActiveX toolkit, while part 2 uses library function calls. Before you begin, make sure that you have registered and communicated to the controller using the standard Galil software.

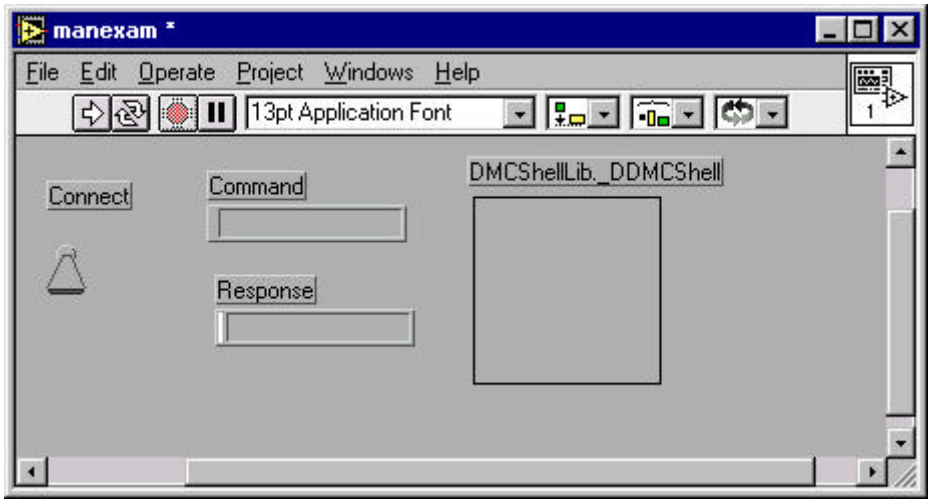
Using the Active-X Tool Kit with LabView

LabView version 5.0 now supports ActiveX (OCX) controls. The standard Galil tool kit can be used without modification. Please note that while properties and methods are supported, 'events' are not. This is a limitation of LabView but should not effect the operation of the tools but it does prevent writing code to respond to error events. The ActiveX tools that you plan to use need to be installed and *registered* as outlined in the ActiveX User Manual.

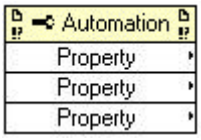
This step by step example demonstrates using the DMCShell, DMCTerminal, and DMCPoll in a LabView application:

- Start with a new LabView project
- Add an ActiveX container, found in the controls palette, to the project
- Select the ActiveX container with the left mouse button and select 'Insert ActiveX Object'. Select DMCShell from the menu.
- Add one Boolean switch from the controls palette and draw it on the project.
- Add one string control and one string indicator and draw it on the project
- Left mouse click on the string control to bring up a menu. Select 'Limit to single line'.
- Create labels on the controls by left clicking the mouse on the control and selecting 'Show-Label' then typing the label text.

The project should look like this:

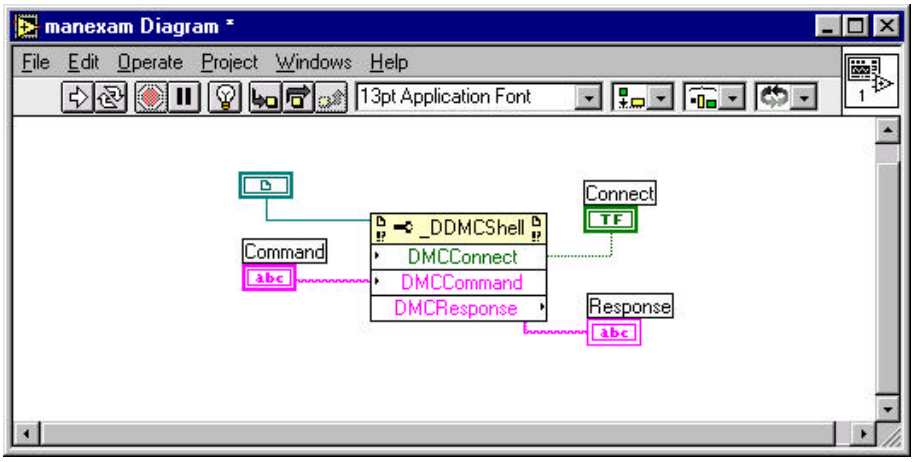


- Show the project diagram by selecting 'Show Diagram' from the Windows Menu.
- Add an ActiveX property node by selecting the Communications icon from the Functions palette then ActiveX from the sub-menu
- Enlarge the icon by dragging a lower corner so that it shows 3 property boxes as shown below



- Select the Connect Wire icon from the Tools palette
- Draw a wire from the DMCShell1 icon to the top left corner of the Automation icon show above.
- Change each property on the Automation icon (which now reads _DMCShell) by placing the mouse pointer over a property box and left click to bring up a menu. Select DMCConnect, DMCCommand, and DMCResponse for the three boxes.
- Left mouse click over the DMCConnect box and select 'Change to Write'. Do the same for DMCCommand.
- Draw a line using the Connect Wire tool from the DMCConnect box to the Switch labeled 'Connect' (a box with TF inside)
- Draw a line from the DMCCommand box to the string control labeled Command.
- Draw a line from the DMCResponse box to the string indicator labeled Response

The diagram should look like this:



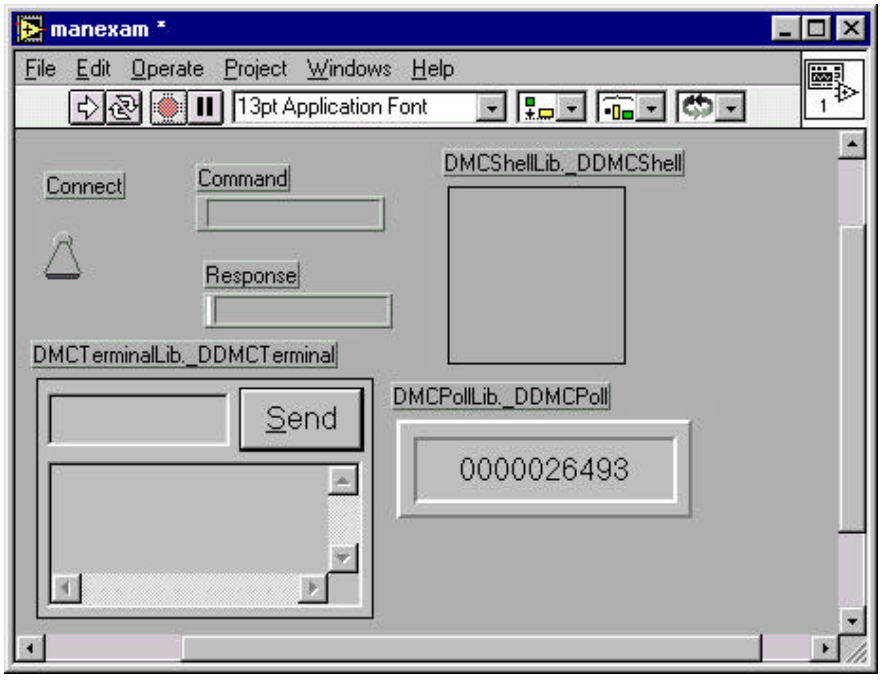
- To run the program click the Run Continuously icon
- From the project window click the Connect switch to start communication with the controller.
- Type a valid Galil command in the Command string control and hit return
- The response should be displayed in the Response string indicator.

Before stopping the program make sure the controller is not connected: hit the connect switch to turn off the connection.

Adding the terminal and polling windows.

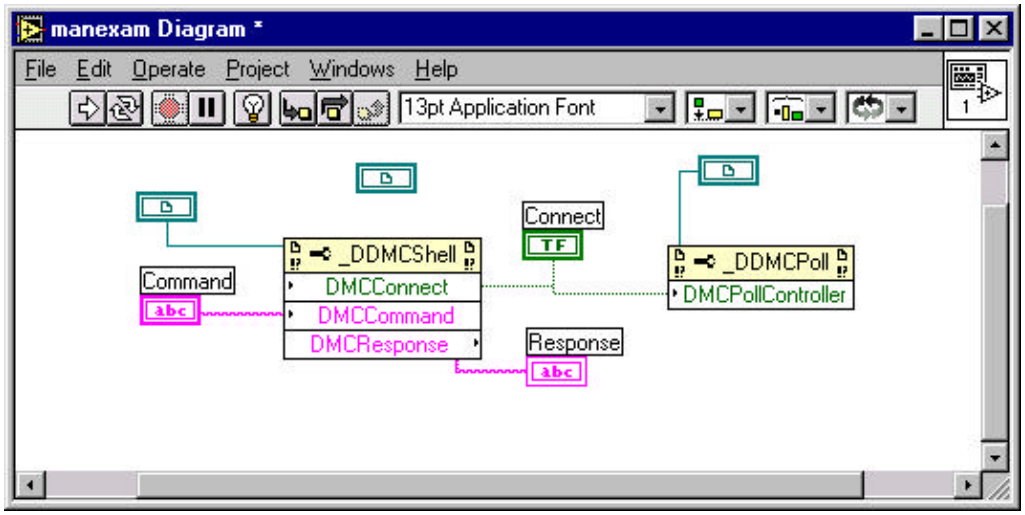
- Add two more ActiveX containers to the project.
- Make one a DMCTerminal, the other a DMCPoll
- Change the properties of the polling window by left clicking on the polling window then selecting DMCPoll Control Object - Properties. Change DMCCommand to TPX. Similarly, you might need to change the Font Color of DMCTerminal to view the text correctly.

The project should look like this:



- Add a Property Node to the diagram and draw a line between the DMCPoll object and the top left corner of the node
- Change the property in the node to DMCPollController then select Change to Write.
- Draw a line from the DMCPollController box to the Connect Switch

The diagram should now look like this:



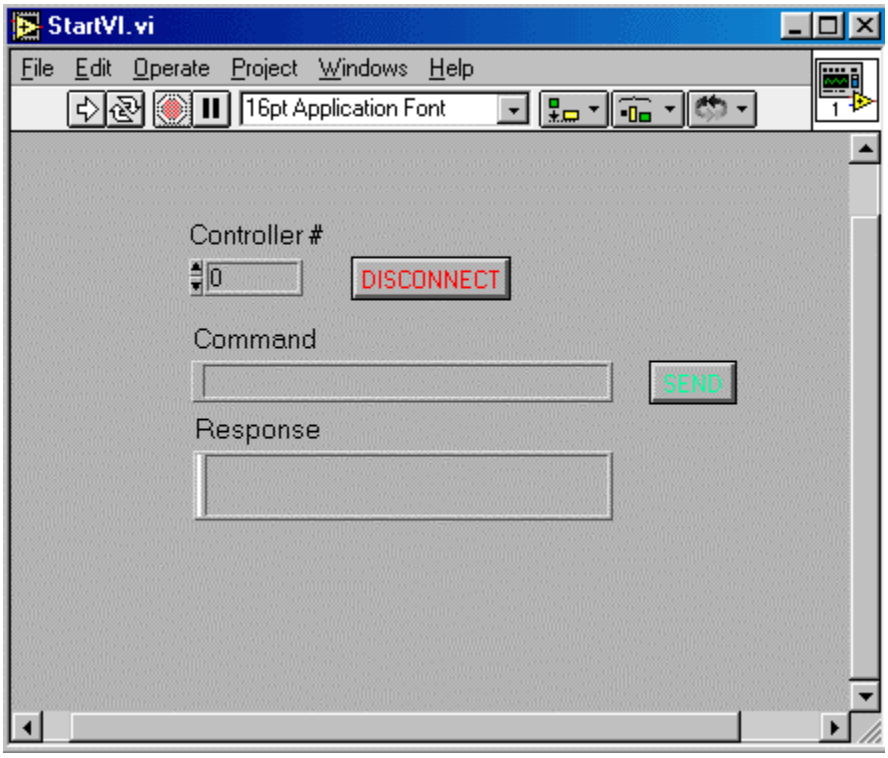
When the program is running the polling window will update. Commands can also be sent though the terminal.

Using Library Function Calls

The other method of programming in the LabVIEW environment is to use sub-VI's that allow you to make function calls from your program directly to the dynamic link libraries (.dlls). A common use for this is to create a custom user interface with buttons that send specific code segments to the controller (ie: Home, Jog, Positional Move, etc…). Other features include being able to quickly and easily graph the controller information to the screen, or save it for comparison later. Here are the basic steps to creating a VI.

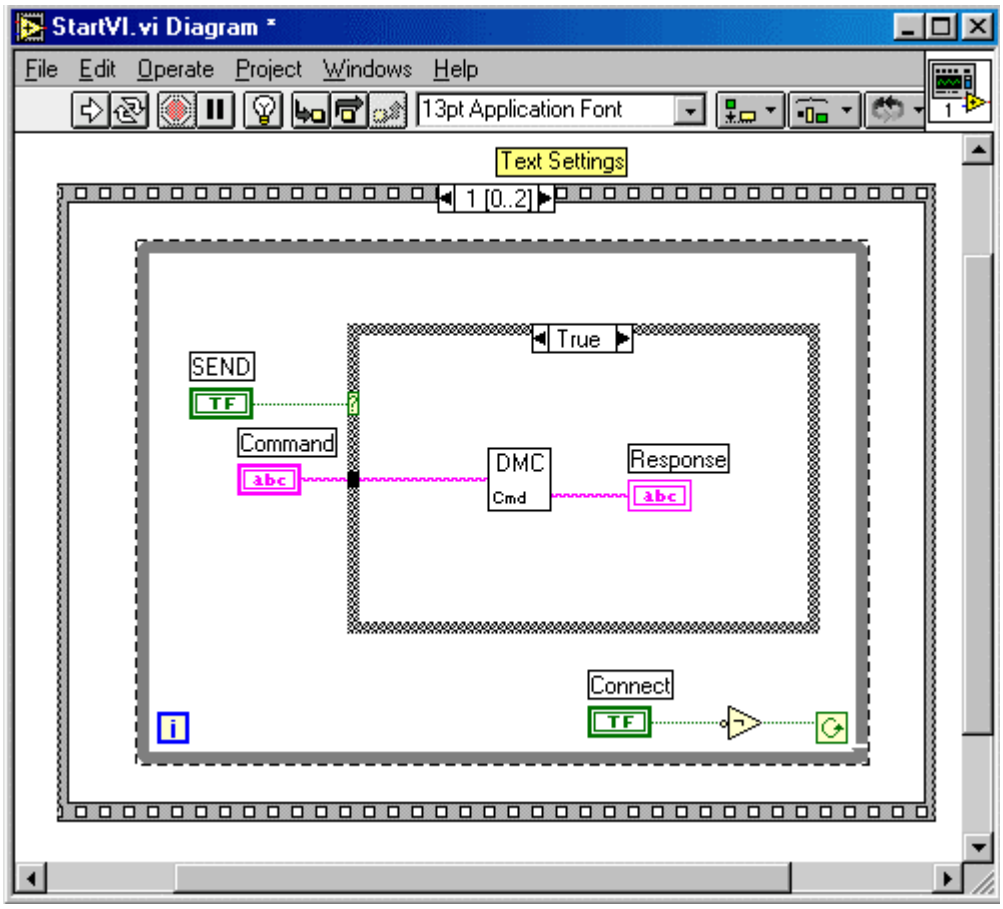
- Download or copy the VIs included on the Galil CD to one of your directories. Start LabVIEW and open a new VI. Go to the Back Panel by selecting Window –> "Show Diagram". Create a Sequence (located under "*Structures*") with three frames by right clicking and selecting "Add Frame After".
- Go to the first frame and click on "Select a VI" from the Functions toolbox. Browse to the directory with the Galil VIs and select "Open.vi" – place this on the first frame of the sequence. Repeat this, but this now browse to Global1.vi and place in your diagram. Put a *Digital Control* on the Front panel with integer representation (I8) and wire this into the "Controller" Global Variable (Global1). (Leave the "Return" of DMCOpen.vi unwired for now).
- Go to the last (3rd) frame of the sequence and insert "close.vi" onto the frame. This ensures that you close the communication channel before exiting the program. No wires need to be attached to this VI.
- In the middle frame, insert "command.vi". This sub-VI allows you to send commands to the controller and receive responses back.
- On the front panel, create a *String Control* and label it "Command". Also Create a *String Indicator* and label it "Response". Create 2 "STOP" Buttons. Label one "Disconnect" and the other "Send".

Your Front Panel should look something like this:



- On the diagram window, create a while loop inside the second frame encompassing everything. Select a boolean "Not" and wire it in line between the Disconnect button and the conditional terminal. This specifies to keep the loop running until the disconnect switch is hit.
- Next, create a "Case" structure and wire the Send Button into the question mark. Place the command.vi and response inside the True case and put the "Command" string control outside it. Wire the Command and Response strings into the appropriate terminals of the command.vi. Leave the False case empty.

Your Diagram should look like this:



Your program is now ready to run. Select the Controller number that corresponds to the number of the controller you want to connect to in the Galil Registry. Click on Run and type in (Upper–Case) commands into the command window. The responses will be returned to your response window. (Note: the ED command will not work in the LabVIEW environment. To download a file, use the sub-VI downbuff.vi or downfile.vi.)

Error checking can be achieved by wiring digital indicators up to the "Return" terminal of each sub-vi. Error codes as well as all possible function calls can be found in the DMCWIN & C/C++ programmer's toolkit. Although Galil only provides a few function calls that have been converted to VI's, the user may create their own by following the same format as those provided.