LabVIEW 8.5

New Features Exercises | Customer Hands-On

NATIONAL INSTRUMENTS

Table of Contents

Setup Instructions

1. If a ‘Hands-On Exercises’ folder has already been extracted to you desktop, delete it to make sure you aren’t using a previous users files.

2. Extract the contents of ‘Hands-On Exercises.Zip’ onto your desktop.

3. The files necessary to complete the exercises will be located in the ‘Hands-On Exercises’ directory on your desktop.

.

Exercise 1 Enhanced Project Window

Goal

The new functionality available within the Project Explorer is designed to help manage large application development by introducing features that prevent cross-linking files and enable advanced control of project hierarchies and dependencies.

Scenario

Top-level VIs can accidentally call the incorrect sub-VI-. Applications that are saved in multiple locations as a result of archiving, backup up or division of work are the most common culprit and can lead to use of incorrect code and broken applications.

Description

If used correctly, the tools provided in the enhanced Project in LabVIEW 8.5 can reduce the chance of incorrect linking and also helps resolve conflicts if they do occur.

Concepts Covered

What scenarios can lead to cross-linked files?

How can you determine the location of files on a disk?

Is there a way to move or rename files from within the Project explorer?

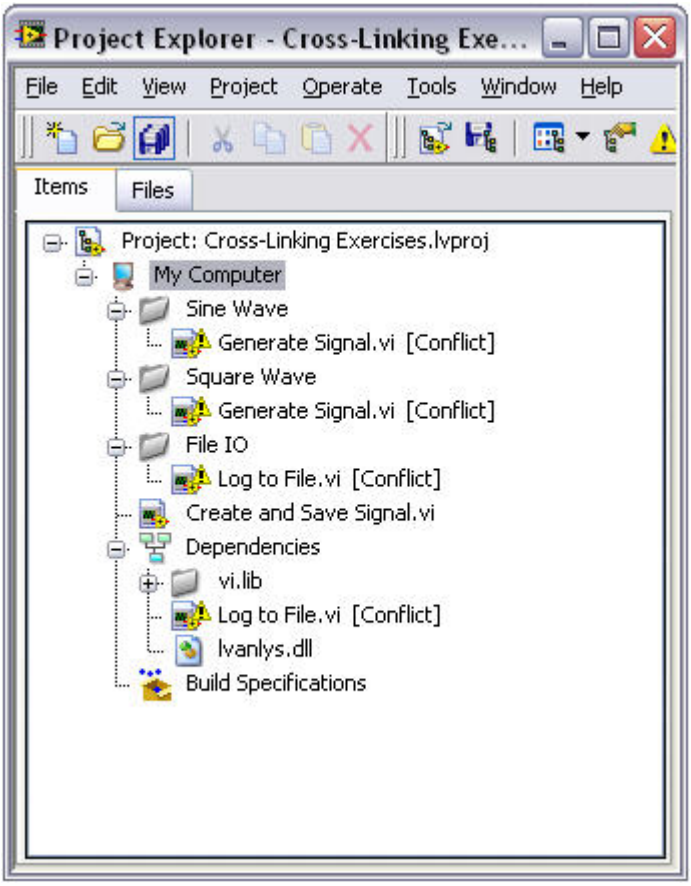
How does the dependencies section distinguish between user subVIs and LabVIEW subVIs?

What does LabVIEW do when it detects a conflict and what methods can be used to resolve it?

What additional functionality has been added to the project?

Part 1: Conflicts and Cross-Linking

1) Exploring a Project With Conflicts
   a) Open `Hands-On Exercises\File and Project Management\Cross-Linking Exercises.lvproj`
   b) Expand the folders to see the various VIs that LabVIEW indicates have potential conflicts. These are highlighted with a yellow exclamation mark next to the name.



   c) Two different conflicts exist
      i) Multiple files with the name `Generate Signals.vi` are referenced in the project
      ii) A top-level VI is calling a subVI outside the project, `Log to File.vi`, which shares a name with a file already contained in the project
   d) Double click `Generate Signal.vi` in the *Sine Wave* folder
   e) Open the block diagram and observe that this VI generates a Sine Wave
   f) Close `Generate Signal.vi`

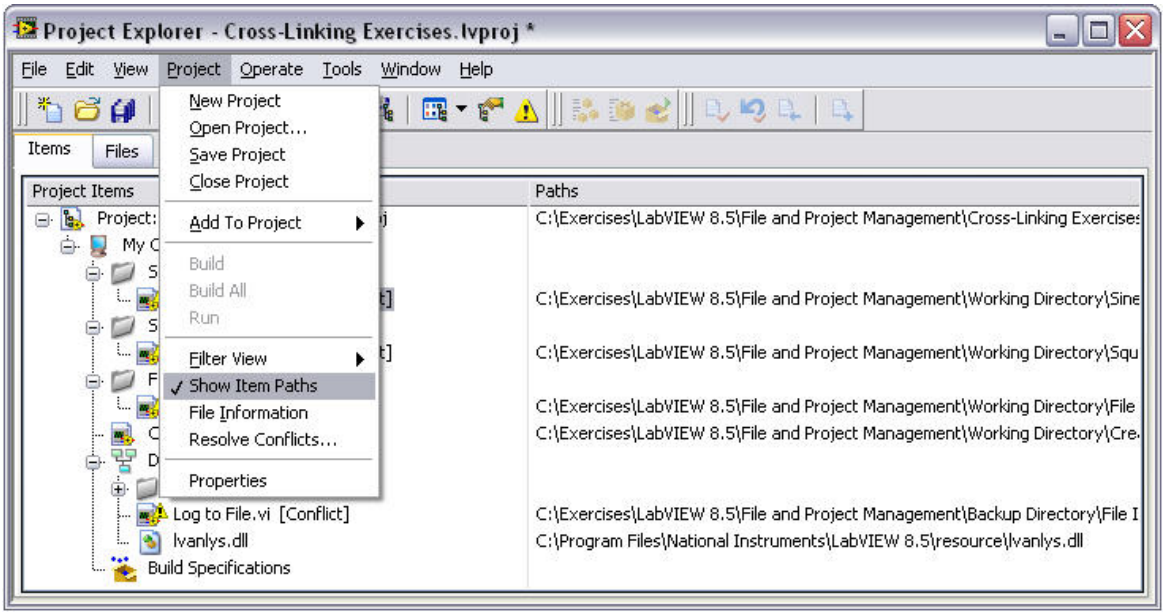
g) Double click `Generate Signal.vi` in the *Square Wave* Folder

h) Open the block diagram and observe that this VI is different because it generates a Square Wave

i) Close `Generate Signal.vi`

2) Examine the new column for viewing the path of a file on the computer's hard disk. The first step when attempting to resolve conflicts that exist as a result of cross linking is often to view the file paths of the files referenced in the project.

a) Turn on Show Item Paths in the Project menu to see the paths of the files by selecting **Project >> Show Item Paths**



b) This redesigned interface clearly shows where files are located in a second column.

3) If one of the two VIs isn't used anywhere else in the application, than the conflict can be easily resolved by removing the unused code. However, it is still important to consider whether or not the code will eventually be used.

a) Ensure that you are still in the Items view

b) Right click on the first `Generate Signal.vi` and select **Find » Callers**

c) LabVIEW will highlight any VIs that call as a subVI

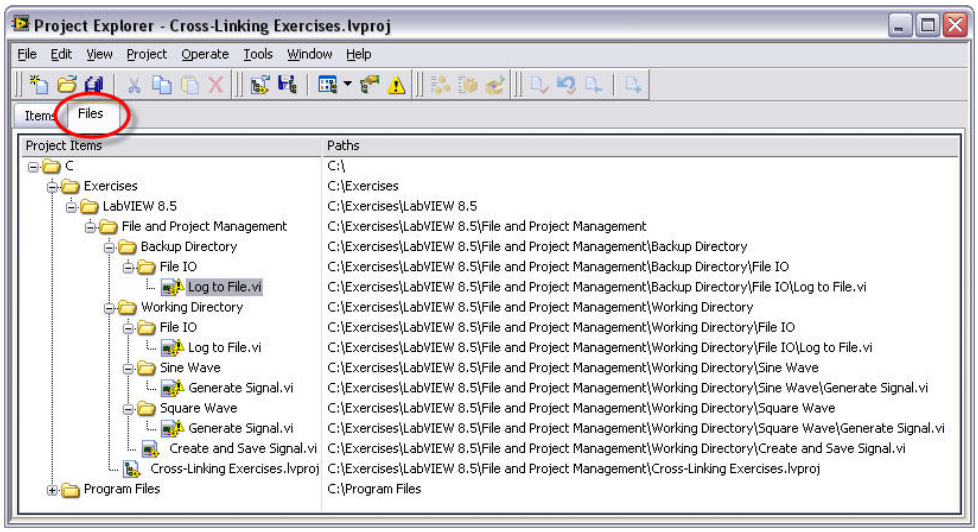
d) Repeat this process for the second `Generate Signal.vi`

e) The second copy is not called by any VIs; however, it may be inadvisable to delete since it appears to have different functionality

f) The appropriate action would be to rename one or both files

4) Manually renaming files is a frequent solution when cross-linking occurs. When renaming a file it is important to ensure that any references to it are preserved by performing this operation from within the LabVIEW project window.

a) Click on the **Files** Tab to see the actual hierarchy of files on your computer's hard drive.

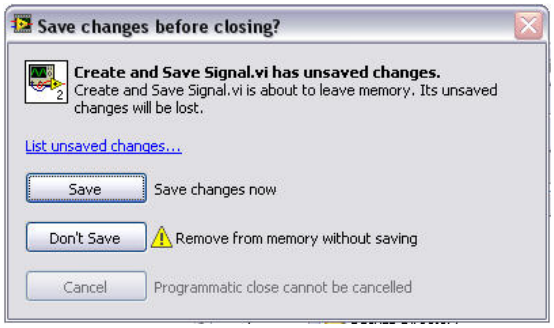


b) Navigate to the locations of both files named `Generate Signals.vi`

c) Right click on `Generate Signals.vi` in the Sine Wave folder and select **Rename…**

d) Type the name `Generate Sine Wave.vi` and click **OK**

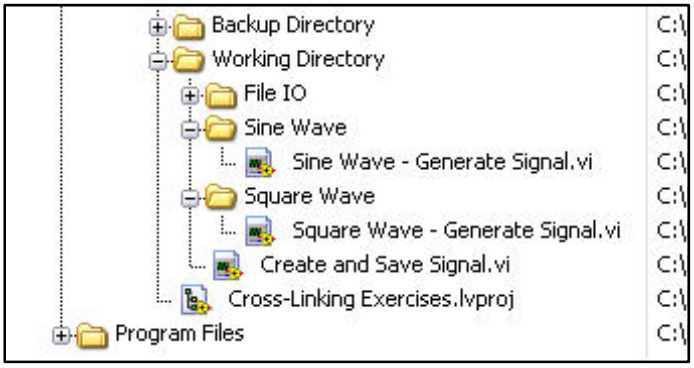
e) LabVIEW prompts you to save the changes made to the calling VI, `Create and Save Signal.vi`, in order to preserve links. Click **Save**.



f) Right click on `Generate Signals.vi` in the Square Wave folder and select **Rename…**

g) Type the name `Generate Square Wave.vi` and click **OK**

h) Click **Save** when the dialog appears.

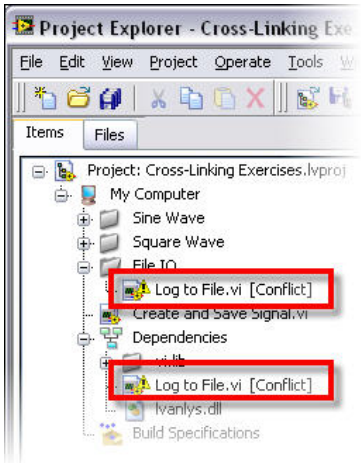


i) Save the modifications to the project by selecting **File >> Save**

5) Explore advanced methods for automatically detecting and resolving cross-linking.

a) `Create and Save Signal.vi` references a file that is not currently in the project, `Log to File.vi`, which we can determine due to the appearance of the file under dependencies. However, a different copy of this file exists within the project within the *File IO* folder.

b) Expand the dependencies and observe that the second copy of `Log to File.vi` is listed here since it is called by a file within the Project



c) The **Dependencies** section of the project has been modified in LabVIEW 8.5 so that it separates user created subVIs from VIs within vi.lib. Now only the contents of libraries referenced within a project will appear within the dependencies section.

d) LabVIEW will prompt you to resolve conflicts if you select any of the following

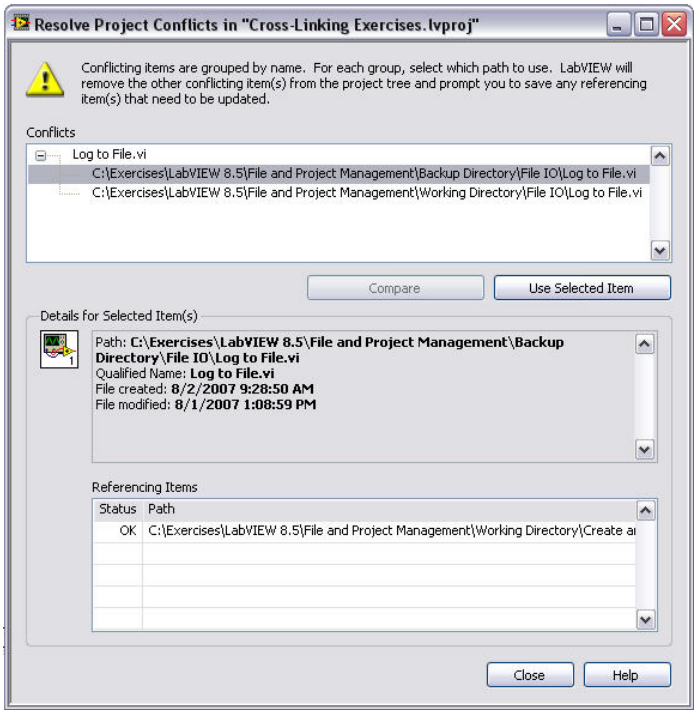
i) Click the **Resolve Conflicts** icon on the top menu

ii) Attempt to open a top level VI that is calling a subVI that has a conflict

iii) Right click a VI with a conflict and select **Resolve Conflcits**

e) Click the yellow icon on the toolbar to resolve conflicts (Option i)

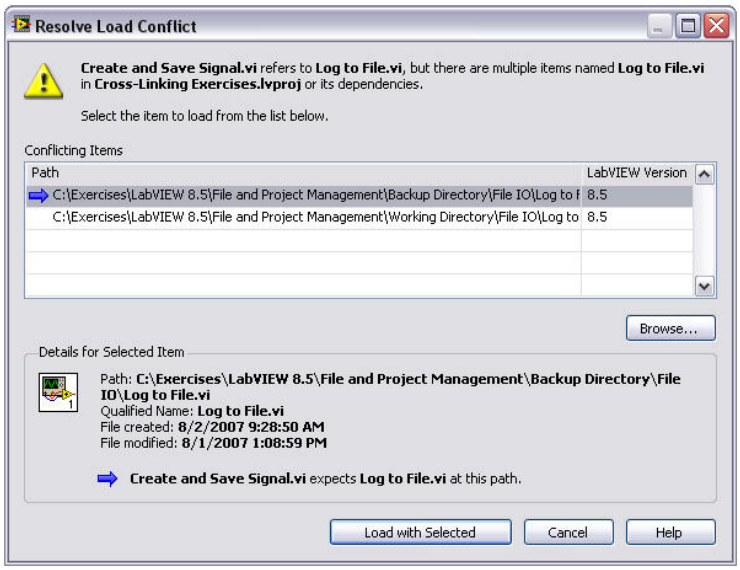
f) The following dialog prompt appears, enumerating all of the conflicts that exist within the Project



g) Close this dialog without selecting an item

h) Open `Create and Save Signal.vi` (Option ii)

i) The following dialog prompt appears, asking you to select which subVI the caller will reference

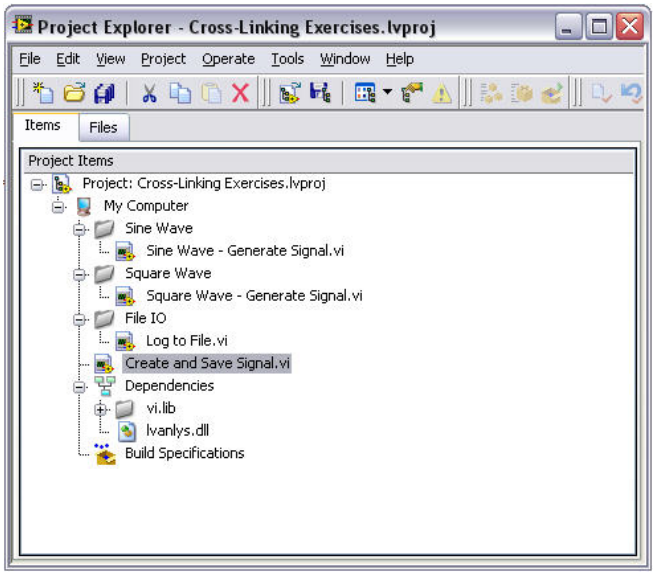


j) Before you can view the VI you must select which of the two conflicting `Log to File.vi` programs should be called by `Create and Save Signal.vi`

k) Select the VI from the *Working Directory* and click **Load with Selected**

Note: To determine which VI resides in the *Working Directory* you'll need to examine the file paths of the two copies of `Log to File.vi`. If the path names are longer than the window allows you to see, hover over them with your mouse in order to display the rest of the path.

l) A warning dialog will appear afterwards informing you that the subVI was switched to a different location.



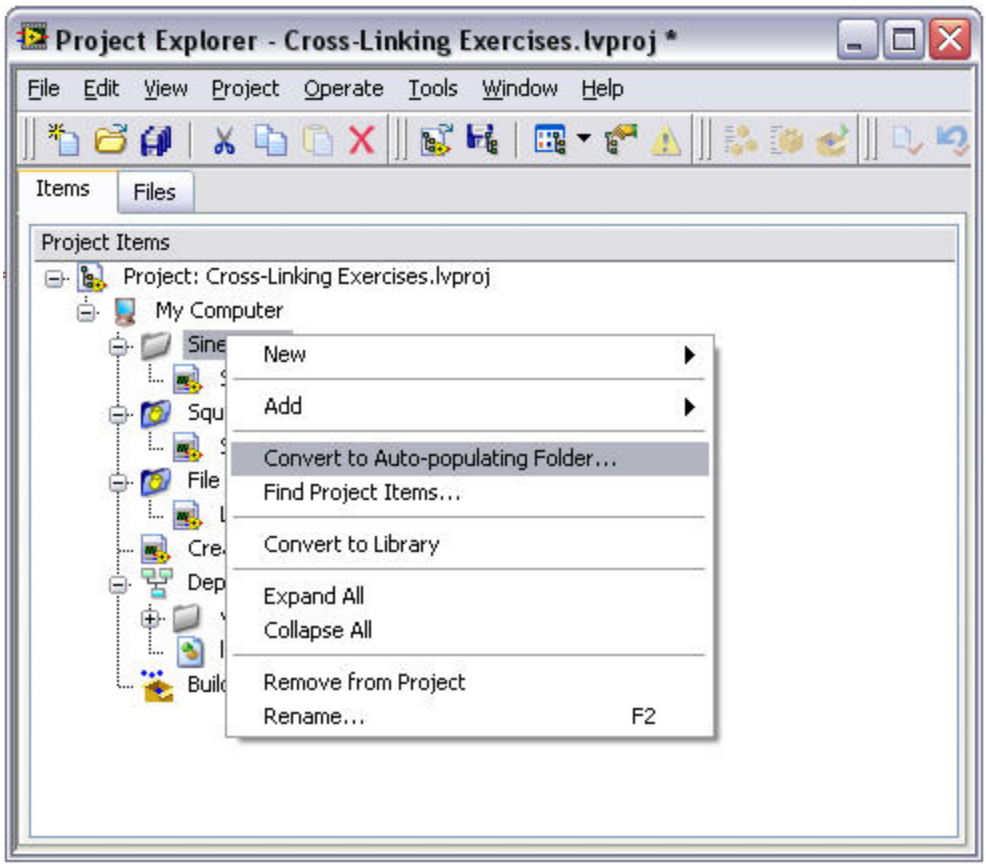
m) All conflicts should be resolved at this point

Note: Conflict resolution may not be possible using this dialog for VIs within folders set to Auto-populate, which is discussed further in the next section. LabVIEW will not remove files from an auto-populate folder since the folder is *always* a direct representation of the contents of the hard drive. Resolution of conflicts within auto-populate folders may require manually renaming or moving files.

Part 2: Auto-Populate Project Folders

1) Auto-Populate is a new mode for folders in LabVIEW 8.5 that simplifies the process of organizing and sorting the contents of the Project. When Auto-Populate is turned on for a specific folder, the contents of the folder reflect the hierarchy on disk. Folders that are set to Auto-Populate are distinguished by a unique folder icon.

   a) Right click on the *Sine Wave* folder in the project and select **Convert to Auto-Populating Folder…**



   b) Navigate to the *Sine Wave* folder located at `Hands-On Exercises\File and Project Management\Working Directory\Sine Wave`

   c) Click **Current Folder**

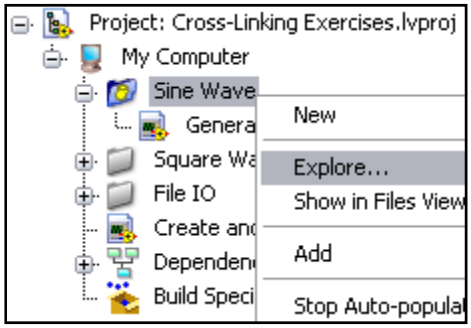
   d) Observe that the folder icon changes to indicate that the folder is now set to Auto-Populate

   e) Click on the **Files** tab

Note: Any modifications to location of files contained within folders set to Auto-Populate will immediately be reflected in the Items view. You cannot move files from within the Items view, since this is no longer independent of actual file location.

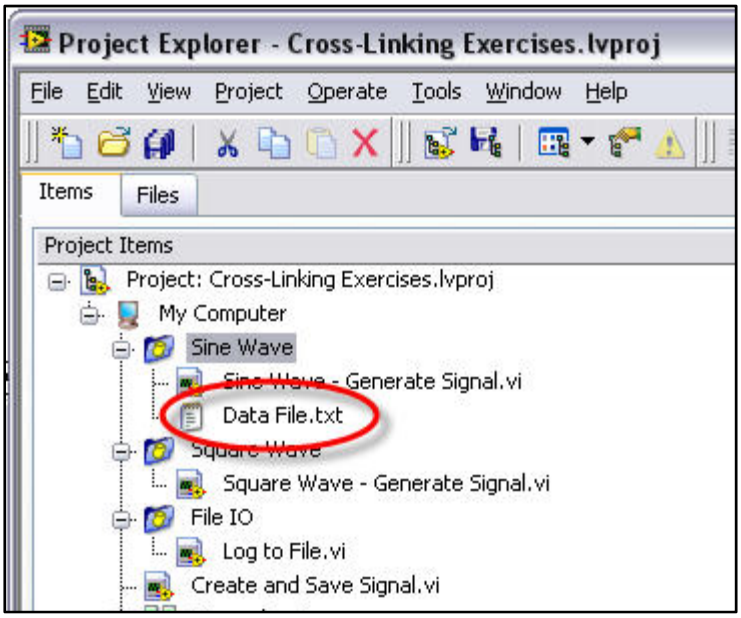
f) Find the *Sine Wave* folder on your computer's disk hierarchy within the Files View by exploring the folders as shown below

g) Right click on the folder and select **Explore**



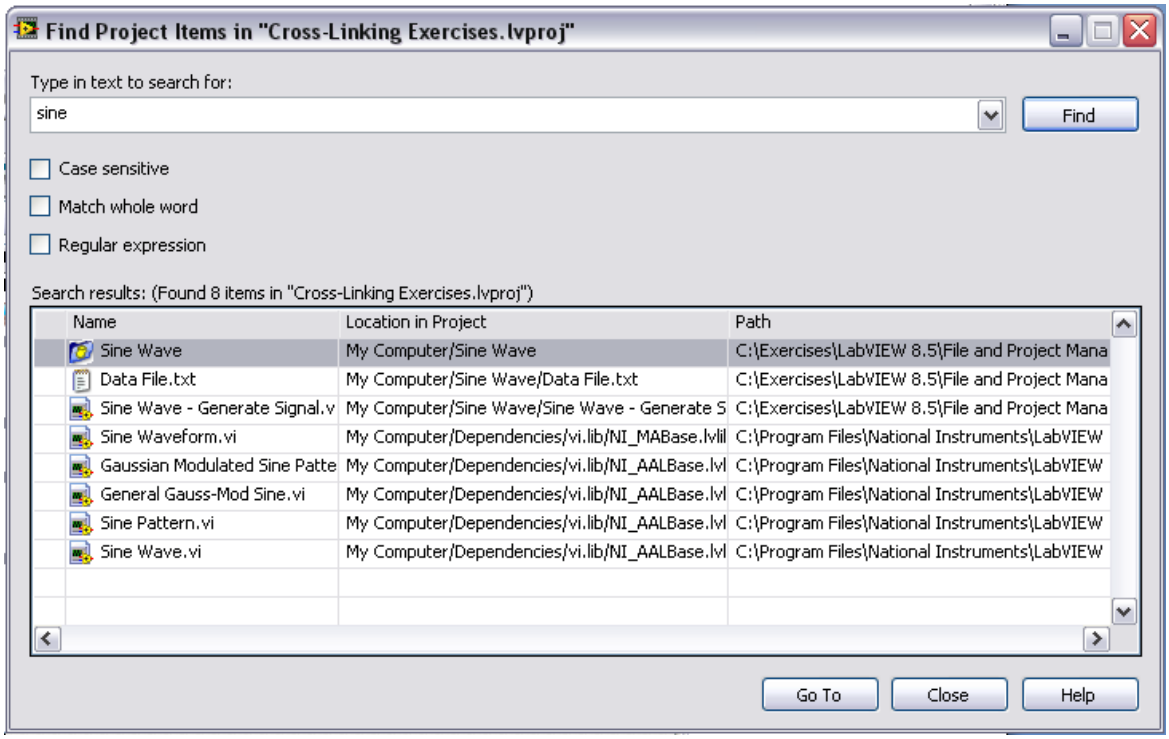
h) Once the explorer opens, create a new text document, type the name `DataFile.txt`, and save it to this location. In Windows you can do this by right clicking in the explorer window and selecting **New >> Text Document**

i) Note that this file now appears in the project window



Part 3: Additional Project Enhancements

1) You can now search the contents of the project by pressing CTRL+F or selecting **Edit** >> **Find Project Items**



2) You can now select **File** >> **Save As** and re-create the contents of an entire project and its dependencies elsewhere on disk. This is only possible if no conflicts exist
3) Replace With
   a) Right click on a VI inside the Project Window
   b) Select 'Replace With' to swap a VI for another VI.
   c) LabVIEW automatically adjusts linking

End of Exercise 1

Exercise 2 VI Merge

Goal

Familiarize yourself with how to use VI Merge to automatically recombine changes made by multiple developers to the same piece of graphical code and how to handle conflicting modifications.

Scenario

Two different developers edit a shared VI at the same time. To continue making progress the work from both developers needs to be combined into a single VI.

Description

VI Merge requires three copies of the VI in order to properly compare and merge the code.

Base VI – The original, un-modified code

Their VI – A copy of Base VI that has been modified

Your VI – A second copy of Base VI that has also been separately modified

VI Merge can be run interactively from within the development environment, or it can be called externally as an executable. Either way, it must be supplied these three parameters to operate.

Concepts Covered

Will the original code run without modification?

What is the difference between the two saved copies of the original code?

How does VI Merge handle conflicting modifications?

What process would be necessary to accomplish the same goal without VI Merge?

How can this process be used to improve development cycle efficiency?

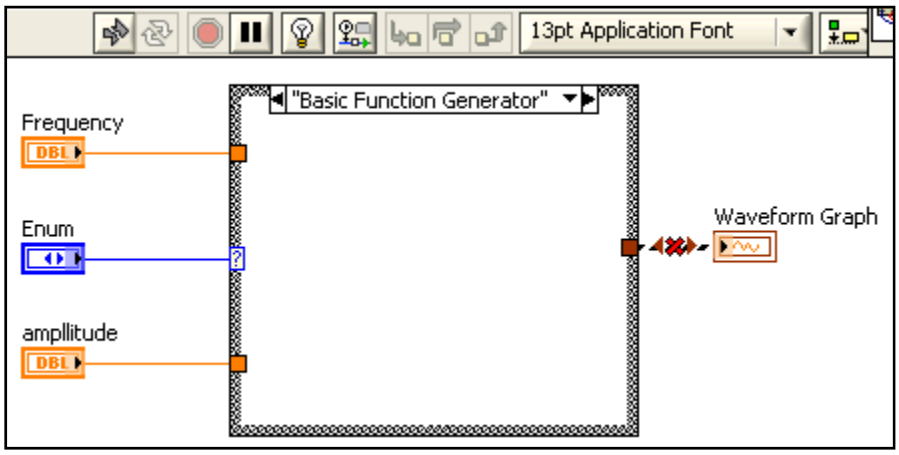
Part 1: Recombine Code

1) View `Base VI.vi`

   a) Open `Hands-On Exercises\VI Merge\Base.vi`

   b) View the block diagram

   c) Note the broken run arrow and the broken wire



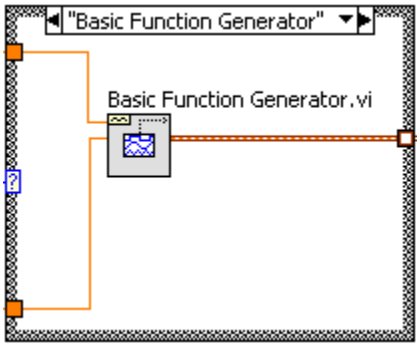
   d) Browse the two cases to observe that they are both currently empty

2) View the two modified versions, `developer1.vi`, and `developer2.vi`. These VIs are effectively the result of distributing `base.vi` to two separate developers and asking them to populate each case separately.

   a) Open `Hands-On Exercises\VI Merge\developer1.vi`

   b) Note that the amplitude control has been changed to a knob

   c) View the block diagram



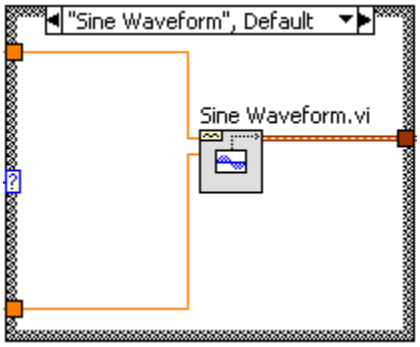
d) Note that the **Sine Waveform** case has been populated with code, but that the **Basic Function Generator** case has not

e) Close the current VI

f) Open `Hands-On Exercises\VI Merge\developer2.vi`

g) Note that the amplitude control has been changed to a slider

h) View the block diagram

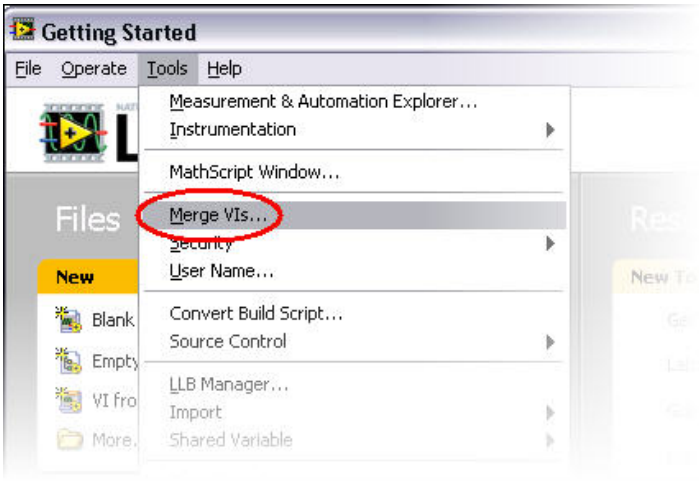


i) Note that the **Basic Function Generator** case has been populated with code to generate a sine wave, but that the **Sine Waveform** case has not

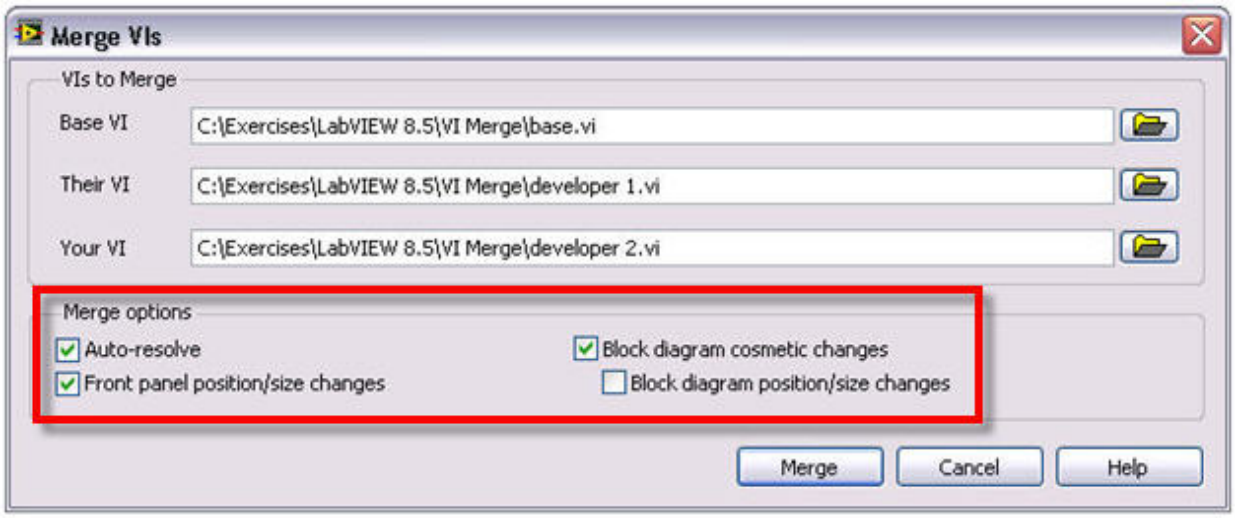
j) Close the current VI

3) Merge Modifications Together

a) On the main menu select **Tools » VI Merge**



b) Enter the location of the appropriate pieces of code into the dialog as shown



c) Make sure that the various Merge options are set as shown above

d) Click **Merge**

4) Merged Result

a) After running VI Merge, several windows will appear

b) **Base VI**, **Their VI**, and **Your VI** will be displayed on the left

c) The automatically merged VI will be shown on the top right of the screen

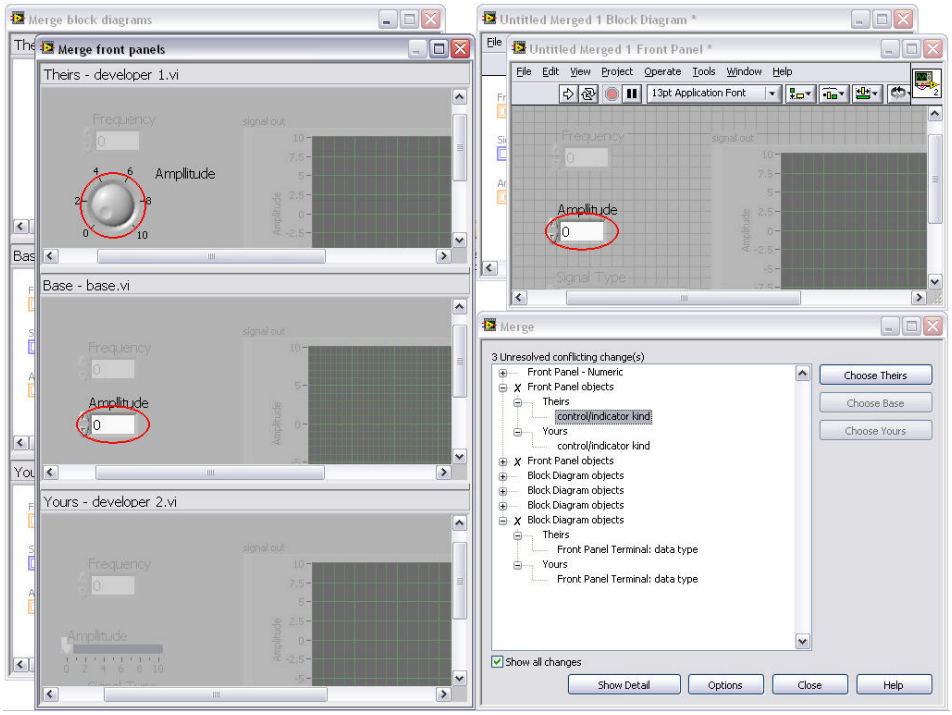
d) Note that the merged VI now has a working run arrow, indicating that there are no errors and that code was successfully merged

e) The bottom right dialog displays all the mergers that were automatically performed

Note: Clicking on the listed mergers in the dialog will highlight the modification that were made on the diagram of the VI.

5) Resolving Conflicts

a) The bottom right dialog will display any conflicts that LabVIEW could not automatically perform. When navigating through these conflicts you have the option of selecting either the modification from **Your VI** or from **Their VI**. If no option is selected, the original, unchanged object from the `base VI.vi` remains.



b) In this example, a conflict exists between the modification to the amplitude control. Developer 1 made it a know and developer 2 made it a slider; therefore, LabVIEW does not know which modification to select.

c) Manually resolve this conflict by selecting the Knob control from Developer1.VI

d) Click Close and save the merged VI to Open `Hands-On Exercises\VI Merge\Merged.VI`

End of Exercise 2

Exercise 3 In Place Element Structure

Optional Exercise

Goal

Familiarize yourself with how to use the In Place Element Structure to explicitly tell the LabVIEW compiler not to allocate memory or copy data.

Scenario

Applications often modify an existing element in an array. An example of this might be an array containing information about a list of people. Each person's name, birthday, and children's names are included. The children's names need to be converted into all capital letters and the new string must replace the current string.

Description

The In-Place Element Structure will be used to minimize the number of copies that are made in memory of the data array.

Concepts Covered

When is it pertinent to use the In Place Element Structure?

What palette is the In Place Element Structure located on?

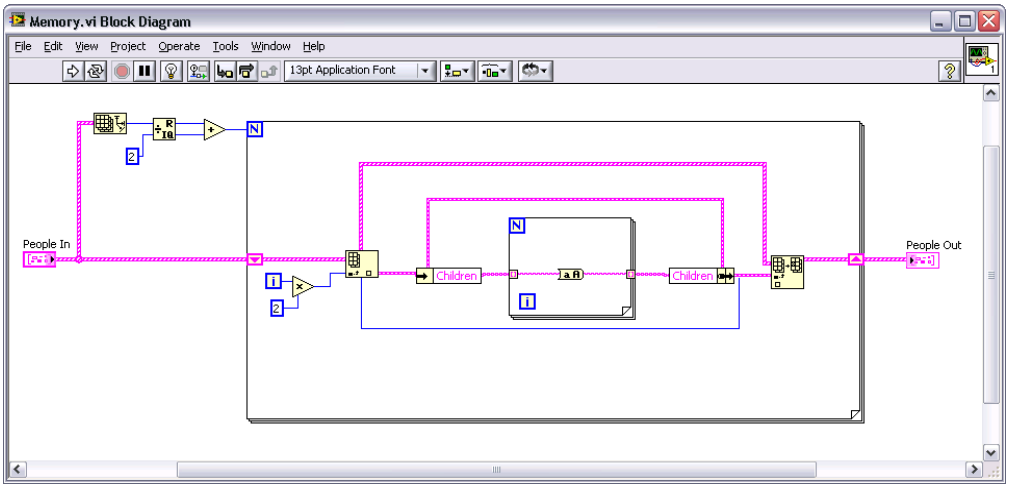
Will the unbundled allow the user to select Birthday and Birthday.Day?

Why is a shift register included in this VI if the In Place Element Structure already tells the compiler to replace the data?

Part 1: LabVIEW Data Memory Allocation

1) Examine memory usage without the In Place Element Structure

   a) Open `Hands-On Exercises\Memory Management\Memory.vi`

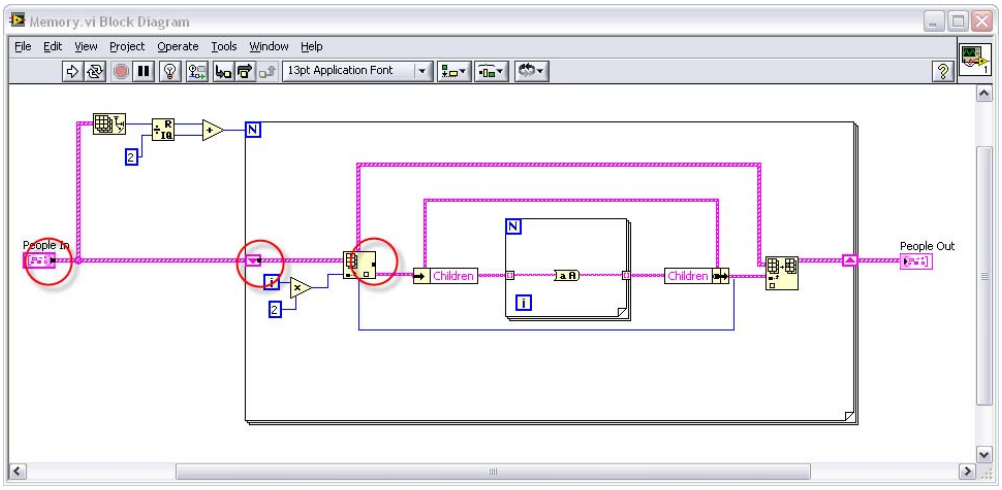


   b) View the block diagram

   c) Select **Tools >> Profile >> Show Buffer Allocations**.

   d) Select **Arrays** and **Clusters**

   e) Click **Refresh** to show black dots on the block diagram where the compiler will be making a copy of the data in memory. Large, memory intensive applications often require certain coding practices in order to minimize the number of copies made, such as the use of the In Place Element structure.

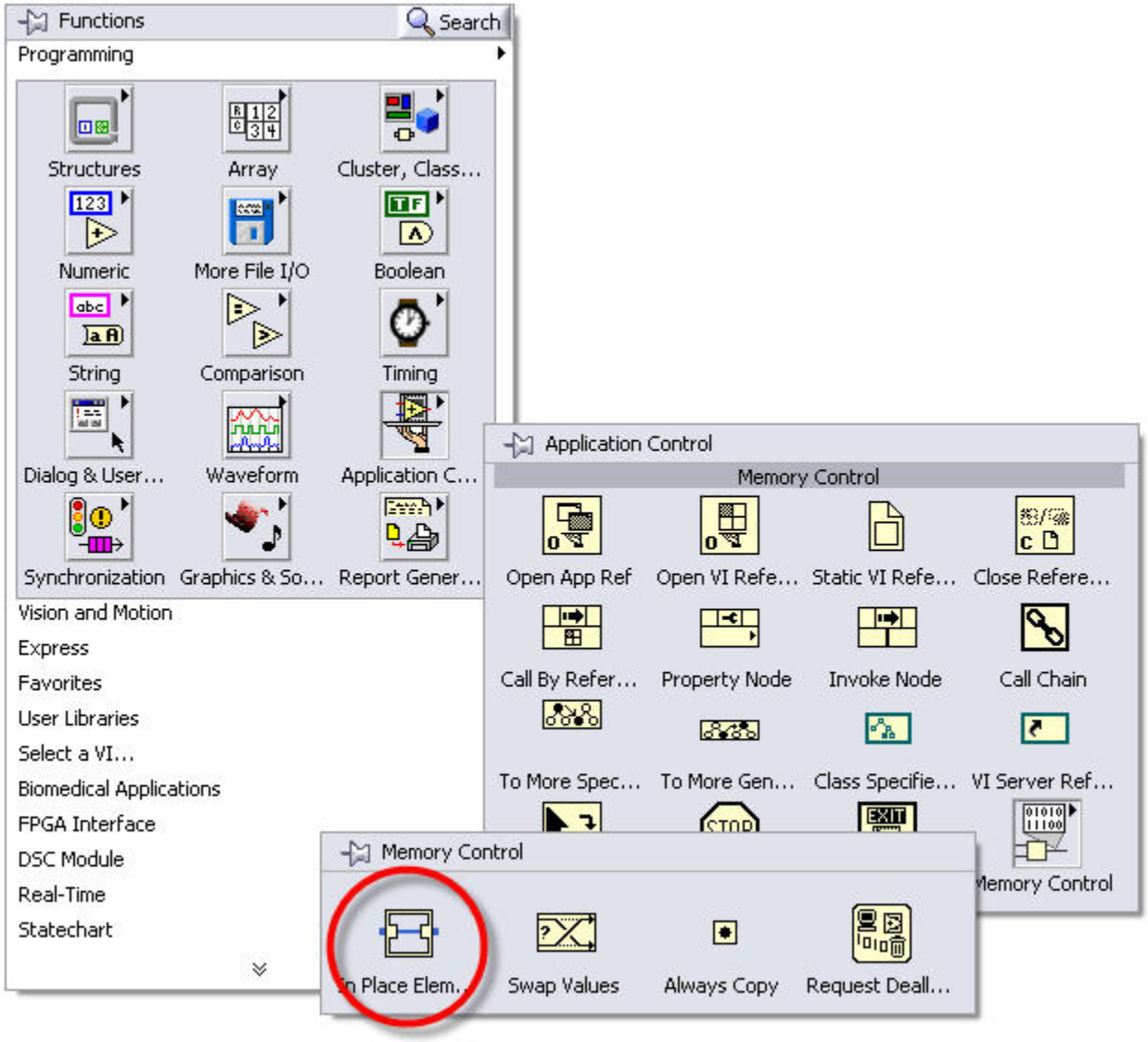


2) Use the In Place Element Structure to reduce the number of copies made

   a) Delete the Index Array and the Replace Array Subset from the VI block diagram

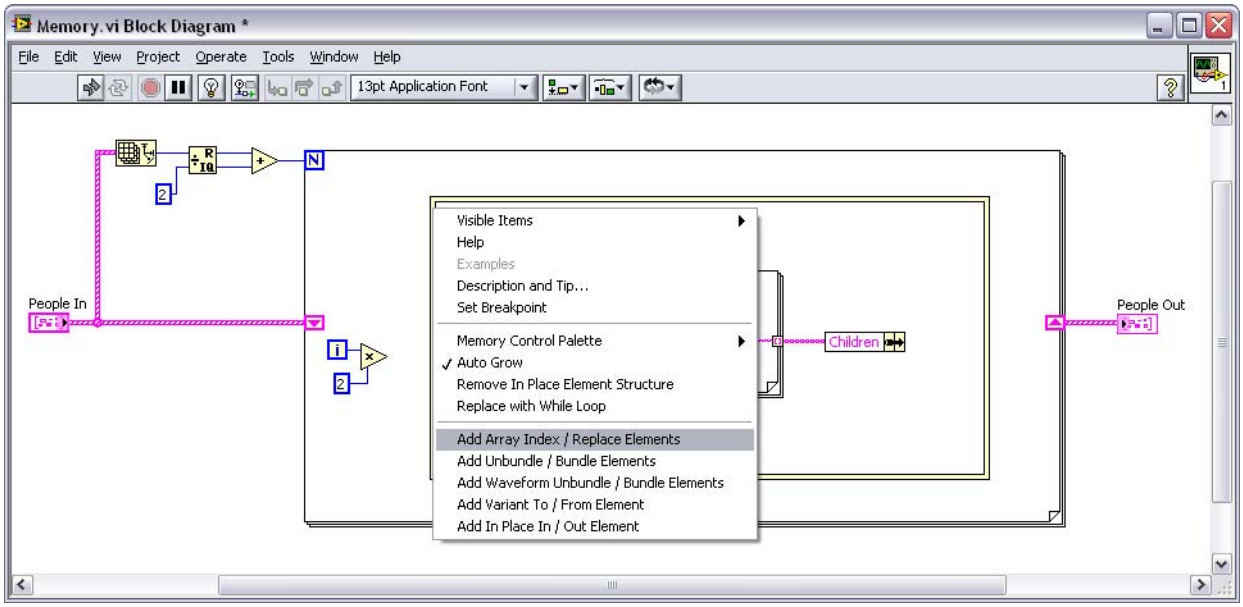
   b) Right-click on the block diagram to see the Functions Palette

   c) Go to **Programming >>Application Control >> Memory Control** and select the **In Place Element Structure**

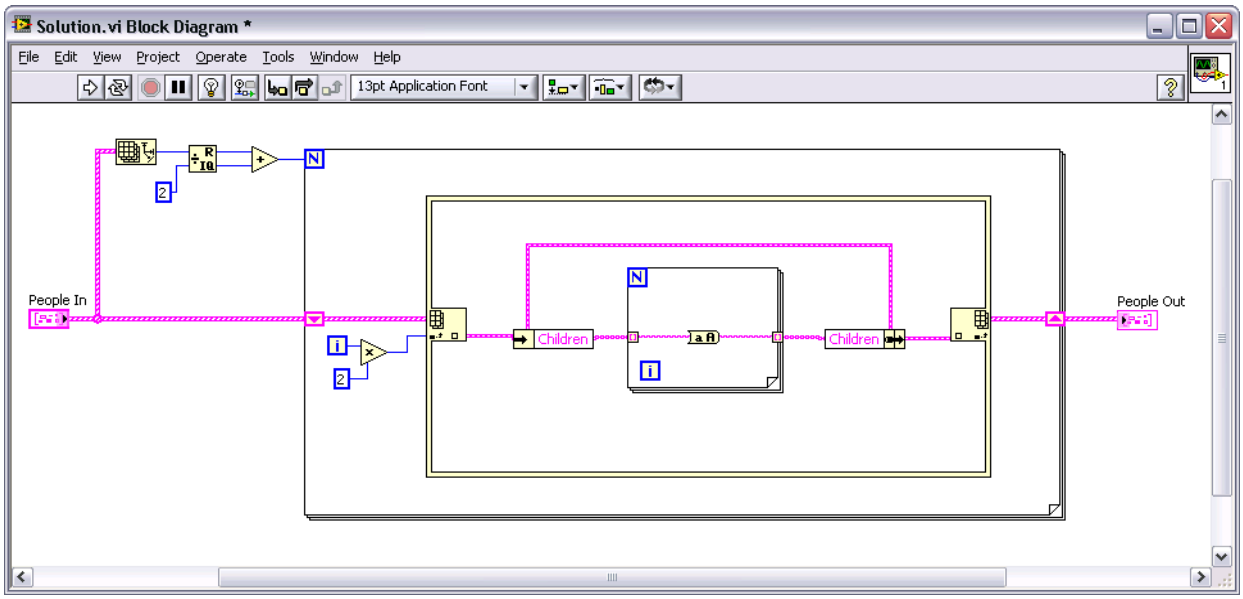


d) Place the structure on the block diagram around the unbundle and bundle

e) Right-click on the frame and select **Add Array Index / Replace Elements**



f) Wire as shown in the diagram

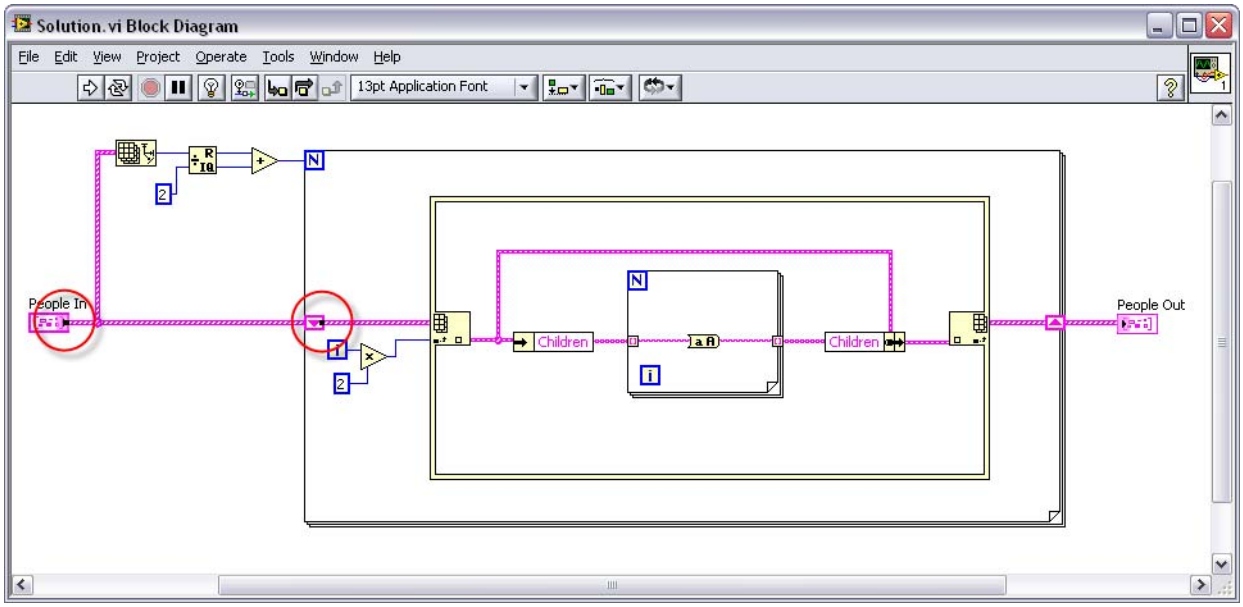


3) Use Show Buffer Allocations to view the number of dots with the In Place Element Structure

   a) Go to **Tools >> Profile >>Show Buffer Allocations**

   b) Select Arrays and Clusters

   c) Click Refresh to show the dots



   d) Note that the number of dots has been reduced by one and that the compiler is therefore making one less copy in memory per iteration

End of Exercise 3

Exercise 4 LabVIEW MathScript

Goal

Familiarize yourself with the new functionality of the MathScript Node in LabVIEW 8.5.

Scenario

A developer is utilizing the hybrid approach to programming in LabVIEW. He is writing a program to generate two test signals, add some noise to these signals, and apply a moving-average filter to remove the noise. With the MathScript Node, the developer is attempting to call a User-Defined Function (UDF) but is running into errors in his LabVIEW diagram. In previous versions of LabVIEW MathScript, error-checking was not performed until run-time. With LabVIEW 8.5, the developer can now take advantage of edit-time error-checking within the MathScript Node, as well as the LabVIEW probe to simplify the development process of incorporating textual math into the graphical programming environment of LabVIEW.

Description

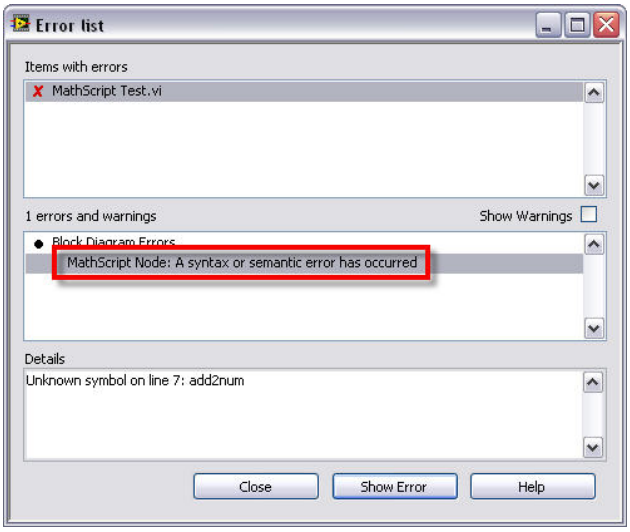
LabVIEW 8.5 adds a number of features for working with MathScript such as:

- edit-time error-checking
- in-node line numbers
- probing
- *.m file incorporation with the LabVIEW Project

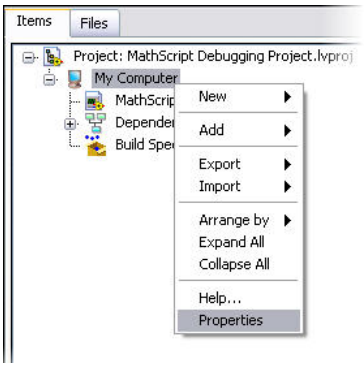
These new features ease the process of incorporating textual math into LabVIEW. MathScript is generally compatible with .m file script syntax, which is the syntax widely used by alternative technical computing software. Such compatibility means that you can work with many previously developed m-file scripts with those available in engineering textbooks or on websites that distribute open-source m-file scripts.

Part 1: Improved MathScript Error Checking

1) Set MathScript search path(s) in LabVIEW to include a User Defined Function (UDF)
   a) Open `Hands-On Exercises\MathScript\MathScript Debugging\MathScript Debugging Project.lvproj`
   b) Double click `MathScript Test.vi`
   c) Note the broken run arrow. Click on the **List Errors** button to display the error:



   d) LabVIEW recognizes that this is an unknown symbol because it does not match any previously-defined variables, native MathScript functions, or loaded UDFs. The unknown symbol error when calling a UDF tells us that the function is not located in a MathScript Search Path. To fix the error, we need to add the directory of the .m file to the MathScript Search Path(s)
   e) In the project window, right click on **My Computer** and select **Properties**

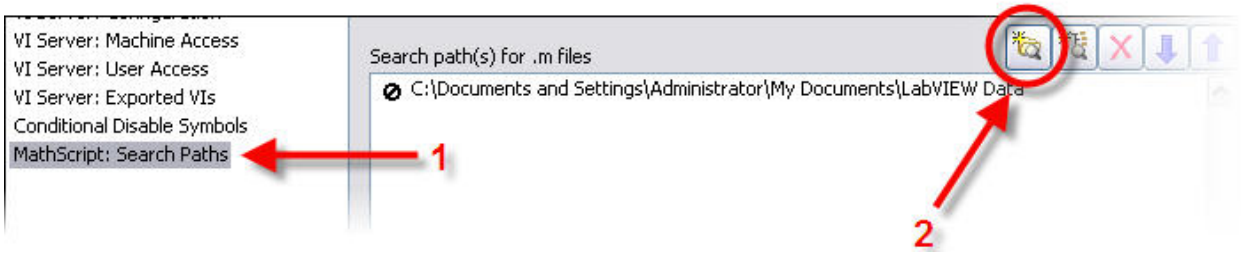


f) Add the `Hands-On Exercises\MathScript\My Scripts` to the Search Path(s)

i) Highlight **MathScript: Search Paths** in the Category list on the left

ii) Click on the folder icon to add a folder

iii) Navigate to `Hands-On Exercises\MathScript\My Scripts` and select **Current Folder**



iv) Click **OK** to close the properties window

v) Right click on **Dependencies** and select **Refresh**

vi) LabVIEW will now list `add2num.m` under dependencies

2) Eliminating syntactical errors from `MathScript Test.vi`

a) Now that the .m file is added to our Search path, `add2num.m` should be a recognized. However, the run arrow is still broken because LabVIEW has detected another error, and LabVIEW will list all errors within the VI if you attempt to run it.

b) Click on the run arrow to display the error. Although there are no problems within the script node, the error tells us that our UDF, `add2num.m,` contains a syntax error.

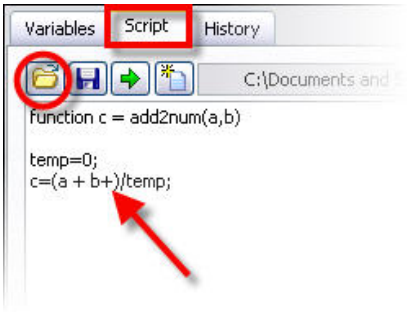
Note: Prior to LabVIEW 8.5, we would not be able to debug UDFs in this manner.

3) Open `add2num.m` in the MathScript window

   a) Open the MathScript window by clicking **Tools >> MathScript Window…**

   b) Click on the **Script** tab

   c) Click on the folder icon and select the `add2num.m` file in order to open and edit it



   d) Note the syntactical error on the fourth line. Delete the unnecessary '+' character.

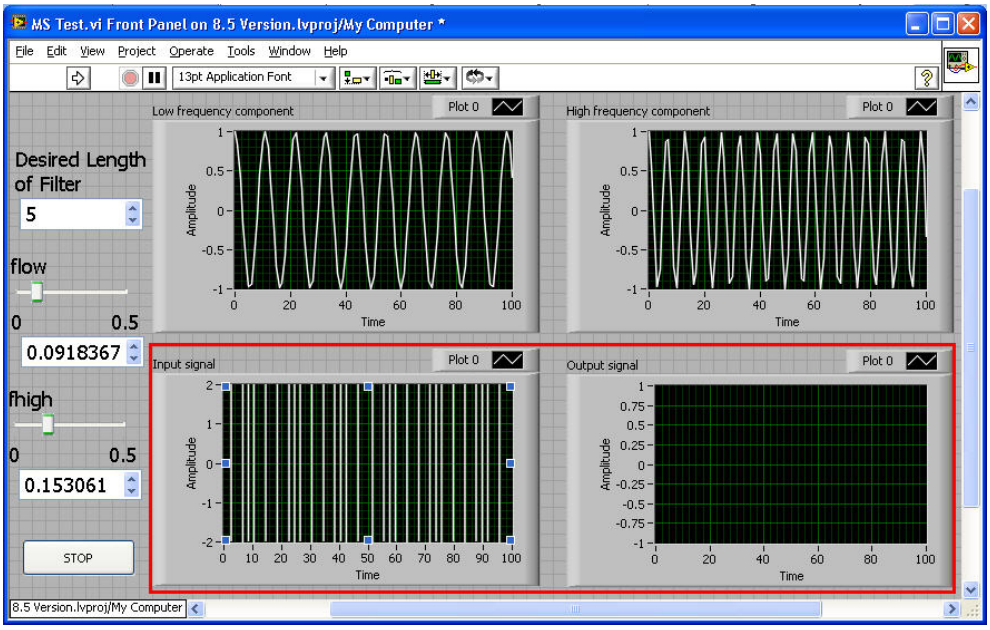
   e) Save the file by clicking on the disk icon in the script window.

   f) Overwrite the existing `add2num.m` file when prompted

   g) Once the save is complete, return the front panel of `MathScript Test.vi` and note that you can now run the VI

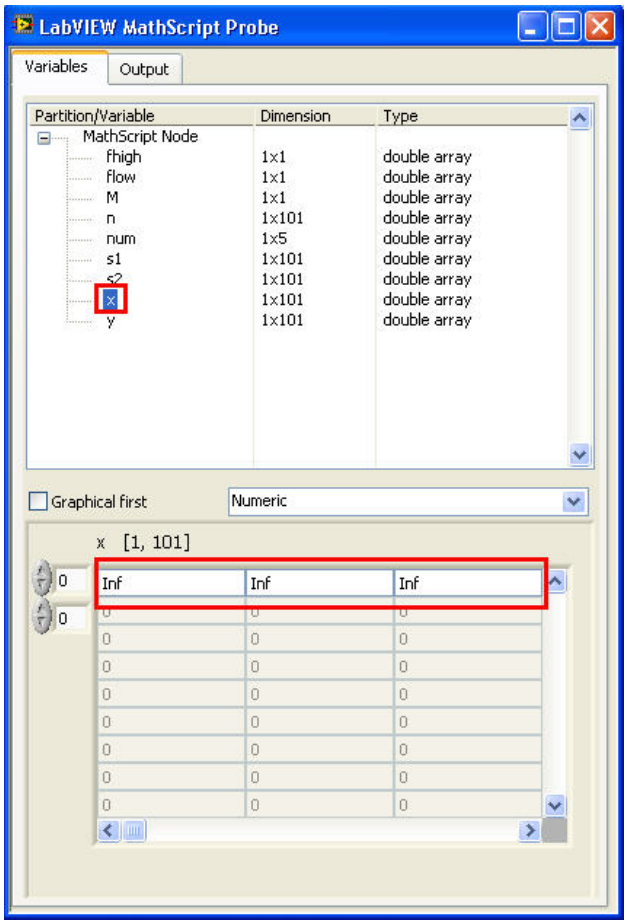
   h) Click the run arrow to run `MathScript Test.vi`

   i) Two of the graphs are displaying incorrect data. In part two we will use new debugging tools to find the source of our erroneous computation.



Part 2: New Tools for Debugging MathScript Code

1) In Part 1, we successfully debugged syntactical errors within `MathScript Test.vi` and we are now able to run the VI. However, the data appears to have a problem caused by incorrect code, which we will need to isolate and resolve. With LabVIEW 8.5, we can probe the MathScript Node and see intermittent values of the variables defined within the node.
   a) Right-click the border of the MathScript Node on the block diagram of `MathScript Test.vi`, and select **Probe**.
   b) The LabVIEW MathScript Probe window will appear
   c) Return to the front panel of the VI and click the run arrow.
   d) Observe that the MathScript Probe window is being populated with variables and their values.
   e) Click the **STOP** button on the front panel of the VI.



f) Double click on x and observe that all values are infinite

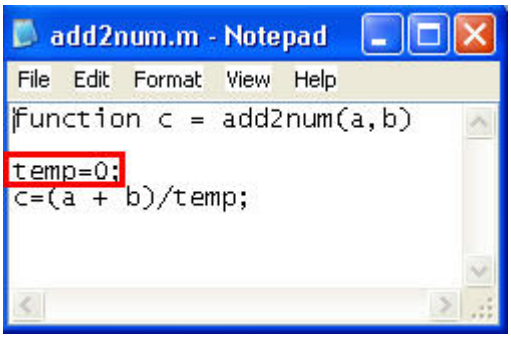
g) Double click on y and observe that all values are not a number (NaN)

2) Identifying the reason for the incorrect values

a) The x variable is set by the returned value of the `add2num.m` function, which indicates that there is a mistake within this UDF.

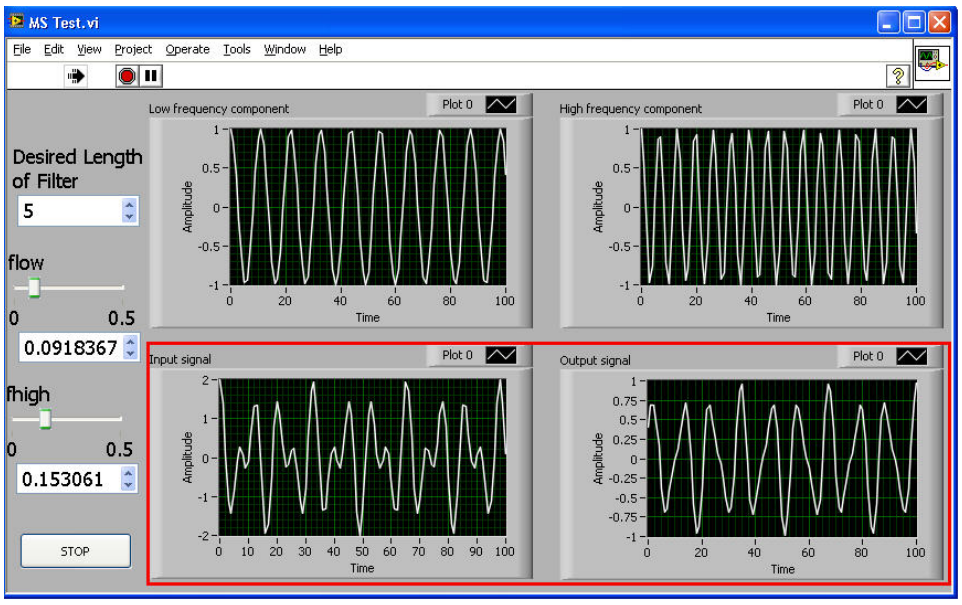
b) Return to the LabVIEW MathScript window and examine how the returned value is set

c) c is infinite since it is divided by the value of temp, which is set to 0.



d) Resolve this issue by changing the value of temp to 1

e) Run the VI and observe that the data is correct and that the program is working



End of Exercise 4

Additional Information

Visit **ni.com/labview/upgrade** to

- Learn about new features in LabVIEW 8.5
- View the upgrade notes
- Review Bug Fixes in LabVIEW 8.5
- Determine NI Software Compatible with LabVIEW 8.5