

LABVIEW 2013 NEW FEATURE DEMO SCRIPT (WHAT'S NEW)

SUMMARY

This demo explores many of 2013's new features, including:

- [LabVIEW Bookmark Manager](#)
- [Attachable Comments](#)
- [New Excel Integration \(Optional\)](#)
- [Mouse Wheel Support for Controls](#)
- [Event-Based Programming Improvements](#)
 - Static Event for Mouse Wheel Interaction
 - Event Inspector Window
 - High Priority User Events
- [Simplified Web Service Experience](#)
 - Project Item
 - Debugging
 - Deploy with Executable

DEMO REQUIREMENTS

- LabVIEW 2013
- Application Builder

SETUP INFORMATION (PRIOR TO PRESENTATION)

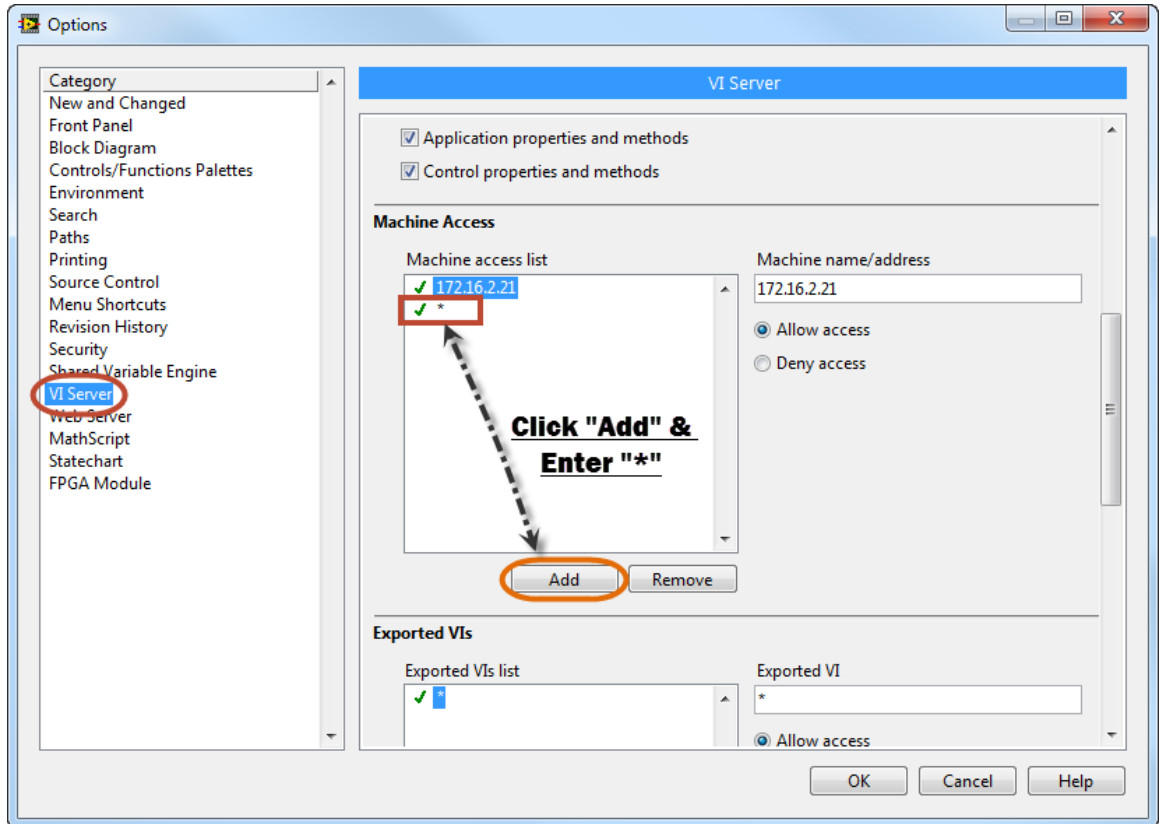
1. Download the What's New in LabVIEW 2013 VI Package file, and double-click on the VI package file.



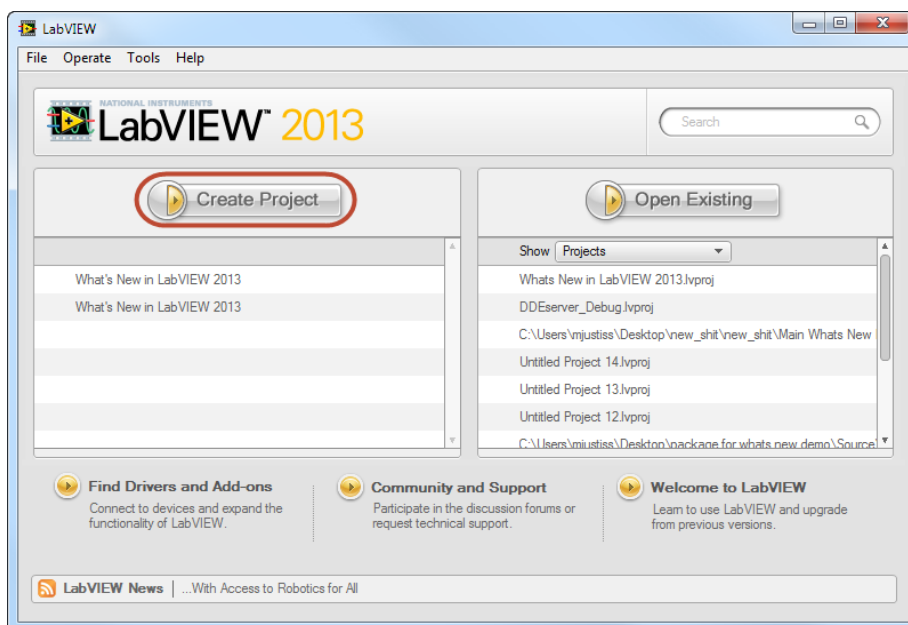
Whats New
in 2013.vip

2. Choose to "Install" the VI package file.

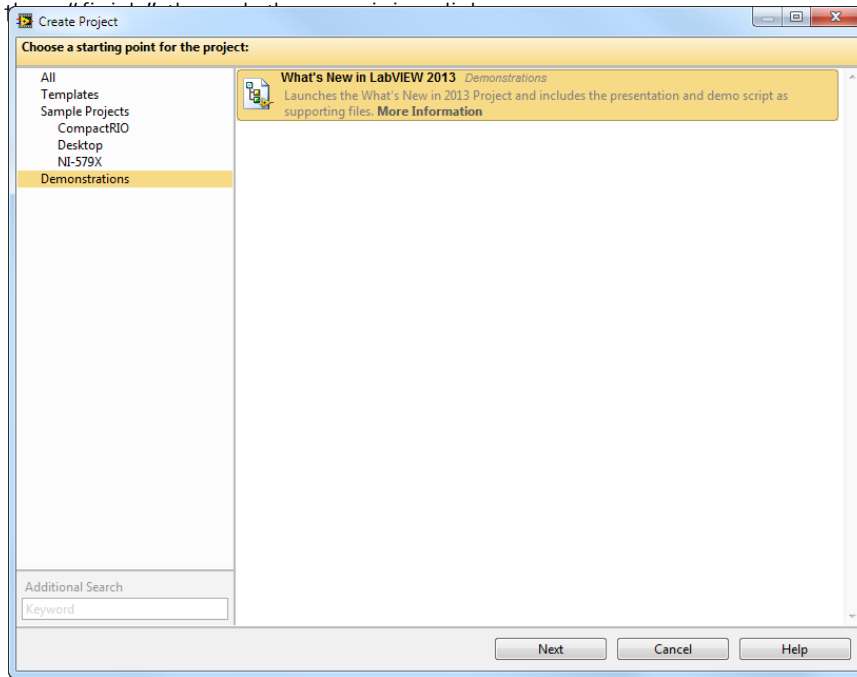
NOTE: If Package Manager claims it cannot access LabVIEW over VI Server, open LabVIEW and Tools >> Options drop-down menu. Look at the "VI Server" category, and ensure both "Machine Access" and "Exported VIs" subcategories have "*" items. This will be absolutely sure that Package Manager may access LabVIEW.



5. From the LabVIEW Getting Started Screen, choose "Create Project".



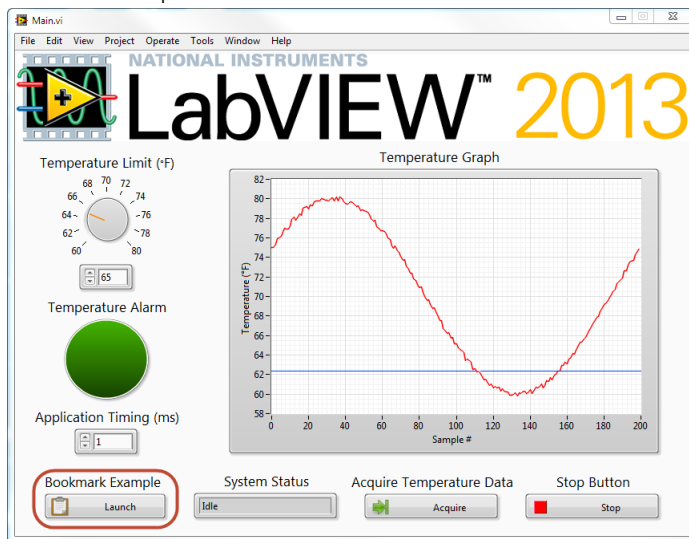
6. Under "Demonstrations", choose "What's New in LabVIEW 2013" and click "next"



NOTE: Once the VI Package file has been installed. You only need to begin this presentation and demo from the "create project" dialog to start with the same files each time. All necessary files will be accessible from the project created through this dialog.

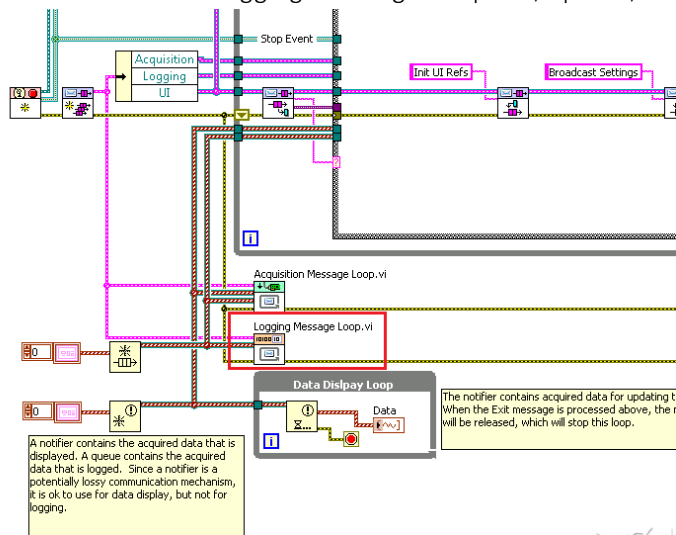
STEP-BY-STEP INSTRUCTIONS

1. **DO:** PRESENT THE WHAT'S NEW SLIDE DECK THROUGH THE DATA DASHBOARD 2.2 SLIDE.
2. **Explain:** Each year, NI strives to incorporate user feedback when determining the features we'll add to LabVIEW. Now that we've discussed some of the new hardware for 2013, I'd like to jump over to LabVIEW and show you some of the new features we've implemented to help you be more productive.
3. **DO:** Ensure the "Whats New in 2013" project is open.
4. **DO:** Open Main.VI and Run It
5. **Explain:** Anyone who has ever inherited and had to interpret existing LabVIEW code knows the value of good documentation.
6. **DO:** Open the Bookmark example VI by clicking the "Bookmark Example" button on the Main.vi front panel.



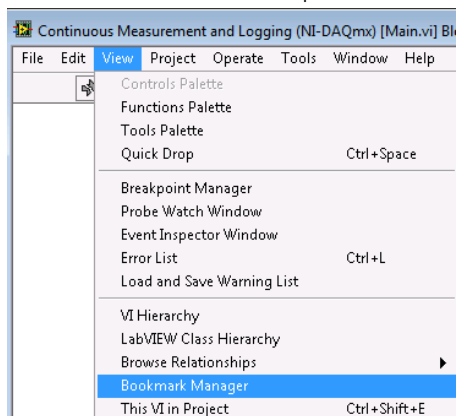
7. **Explain:** Let's imagine we inherited this particular application, based on a LabVIEW sample project, and were tasked with adding additional data logging functionality. Where to start?

8. **DO:** Point out the “Logging Message Loop” VI, open it, and switch to its block diagram.



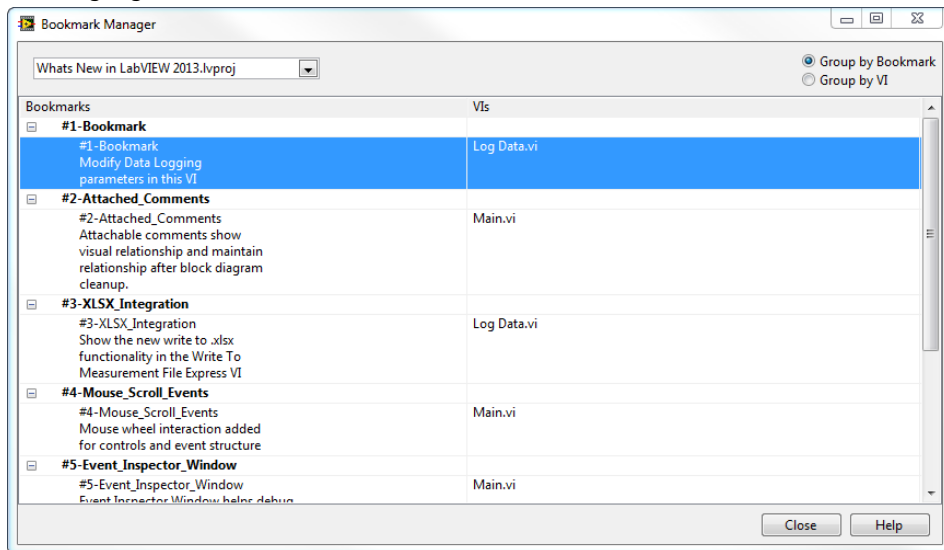
9. **Explain:** For many applications, our code and VI hierarchies quickly become so complex we cannot identify, at a glance, where specific functionality resides. In this case, where I want additional data logging functionality, I could continue to parse through this code, ultimately determining where modifications were necessary, but what if there was a better way? In LabVIEW 2013, there is, and it’s called bookmarks.

10. **DO:** Click the “View” dropdown and select “Bookmark manager”.



11. **Explain:** In 2013, I can turn any comment into a bookmark by simply beginning the comment with a hash tag (#). This prompts LabVIEW to recognize that comment as a bookmark, and aggregate it into this bookmark manager window.

12. **DO:** Highlight the bookmark titled “#1-Bookmark”.



13. **Explain:** Notice I’ve left a to-do bookmark for, perhaps, another developer, indicating I want this parameter change made.

14. **DO:** Double-click on the “#1-Bookmark” bookmark.

15. **Explain:** By simply double-clicking this bookmark, I am able to navigate them directly to the location in my application I wished for them to view and modify. In LabVIEW 2013, all sample projects will make use of the bookmarks and this feature will help single and multi developer efforts alike by simplifying the process of code navigation and note aggregation. NI considers this to be a best practice for code documentation moving forward.

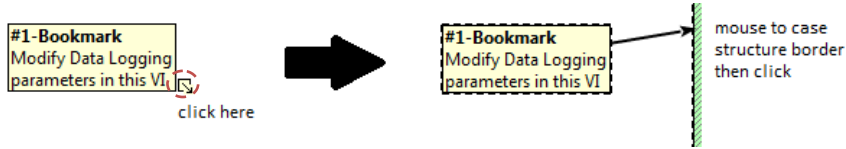
NOTE: You may not navigate to Bookmarks contained in running LabVIEW code. In this demo, the Main VI is running, but the sample project code, used to show Bookmarks, is not running to facilitate this demo.

16. **Explain:** Another feature we’ve implemented, to complement the addition of bookmarks, is the ability to attach comments on the block diagram. In past, LabVIEW developers used free labels to describe algorithms, leave placeholder notes, and generally put their thoughts onto block diagrams. But, aside from decorations, there has never been a way to create a direct and visual relationship between a comment and the code it describes. LabVIEW 2013 changes this by allowing us to attach our comments to any structure, function, or constant on the block diagram.

17. **Explain:** To attach this comment to any function, structure, or constant on my block diagram, I simply move my mouse to the bottom right, click the arrow icon that appears and click again on the entity on my diagram I care about.

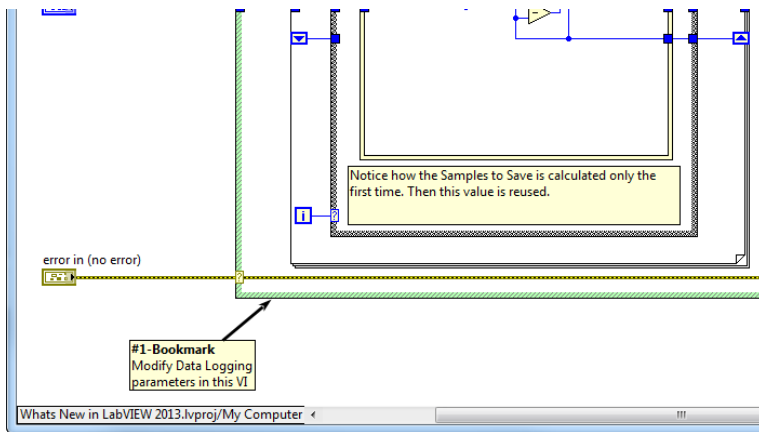
18. **DO:** Move mouse to bottom-right of “#1-Bookmark” comment, look for yellow boxed arrow to appear, click and move mouse to show the carried arrow. Move mouse to the border of

the case structure and click after the structure is highlighted.



19. **Explain:** Now that I've attached this comment, I may move the comment wherever I like and still maintain a visual relationship with this control I've chosen.

20. **DO:** Click and move the comment around to show this relationship.



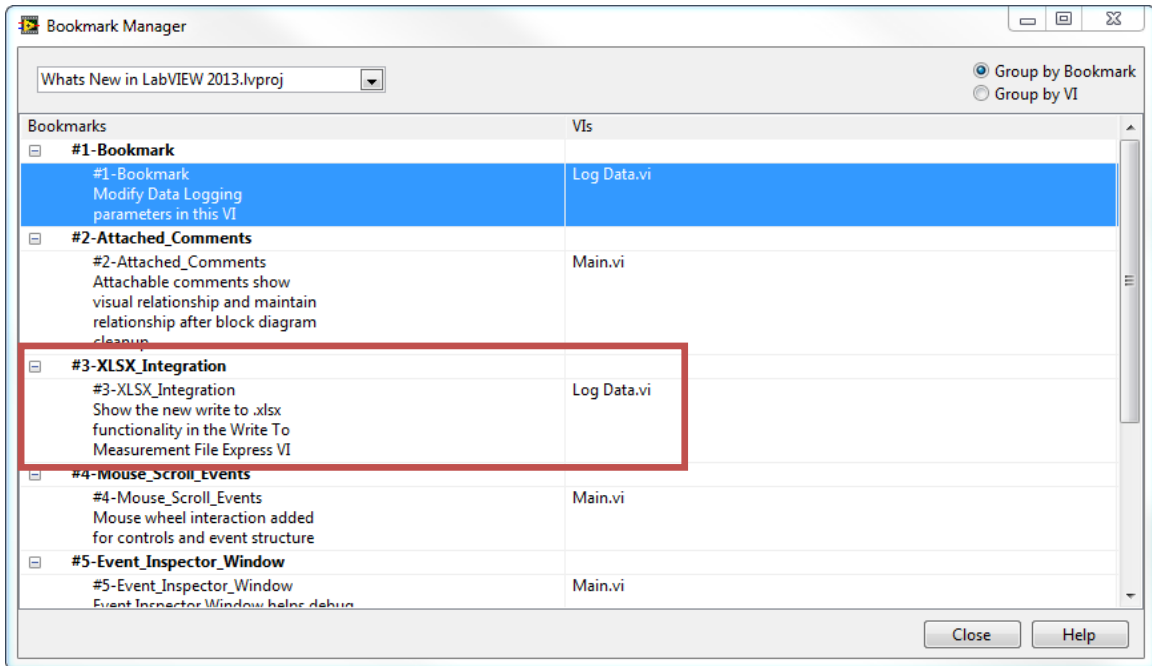
21. **Explain:** Another great attribute of attached comments is their persistence through block diagram cleanup. After attaching, LabVIEW's block diagram cleanup utility will now take this attachment into account, and attempt to keep the comment close to the item it's attached to. No longer will documentation be sent adrift when using this convenient utility.

22. **DO:** Close the "Log Data" VI, Bookmark Manager Window, and "Bookmark Example" VI. Save no changes. – You should now be back to the front panel of Main.VI

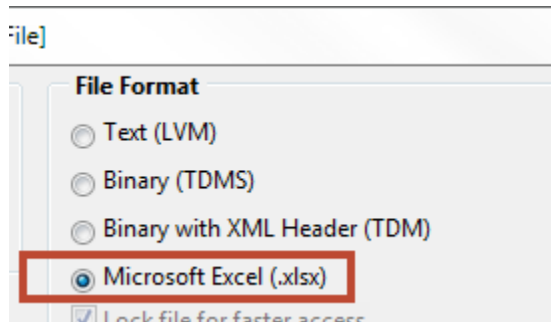
23. **Explain:** We'll use this project to show you some additional new productivity features of LabVIEW 2013. This project is designed to simulate a temperature alarming application, and is based on the common producer/consumer, event-based architecture.

OPTIONAL FEATURE DEMO

24. **Do:** Double-click Bookmark #3-XLSX_Integration to open the Log Data VI.



25. **Do:** Double-click the “Write to Measurement File” Express VI

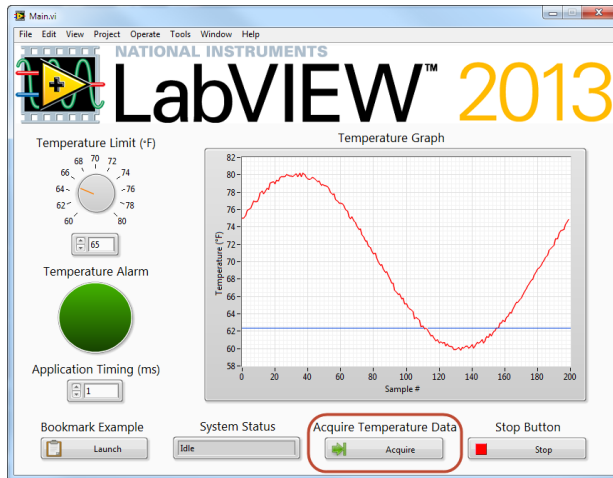


26. **Explain:** In LabVIEW 2013, you may now write data directly to an Excel file type from the Write to Measurement File Express VI. So, if all I need to do is take my data and export it, I have a quick way to do this using LabVIEW 2013, without needing to use the .lvm text file format.

NOTE: This functionality is WRITE-ONLY. We cannot read this file back using the Read From Measurement File Express VI. Also, this functionality makes use of the Open Office file format. So, once the resultant .xlsx file has been opened and saved in Microsoft Excel, the NI Report Generation Toolkit for Microsoft Office would then be necessary to use this file in LabVIEW. Moreover, no extensive formatting options are available, so the Report Generation Toolkit is still recommended for any report creation in LabVIEW.

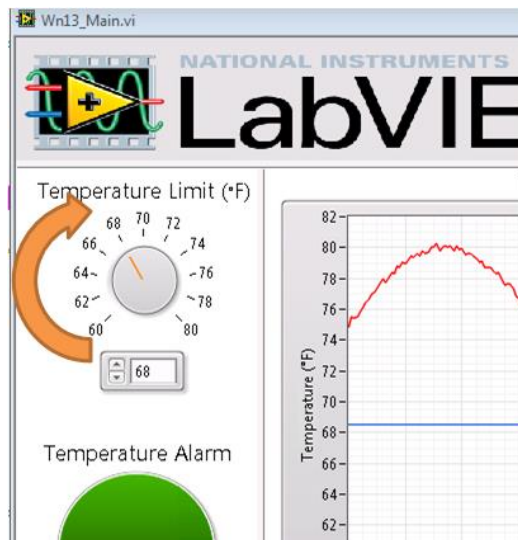
27. **Explain:** Each year, we try to incorporate our user’s feedback, and a highly requested feature has been native mouse wheel support for front panels. We’ve added this support in 2013, allowing simplified programming of mouse wheel interaction with LabVIEW UIs.

28. DO: Click the “Acquire” button to populate the temperature graph.



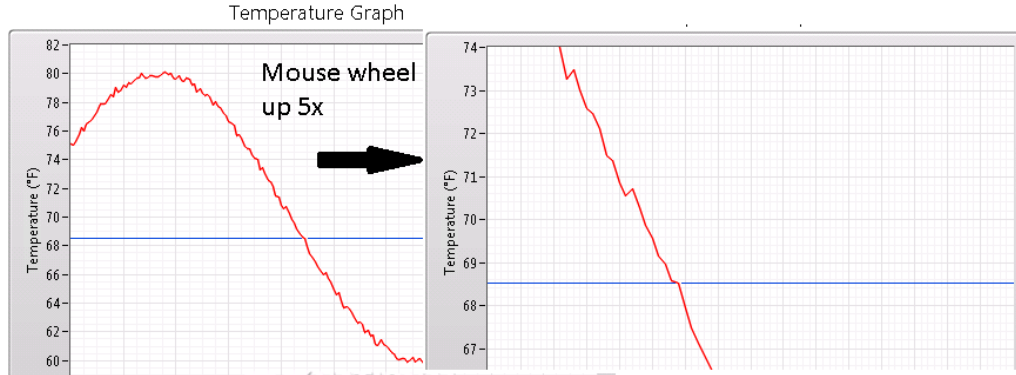
29. Explain: In this example application, I'm performing some logic on my simulated temperature readings to determine if I should alarm my user, based on a limit they've set. [The default limit value is 65, which should alarm when you click “acquire”] If I want to quickly change my limit value, I can now accomplish that with my mouse wheel.

30. DO: Hover the mouse over the “Temperature Limit” control and mouse wheel up and down to change the control value. This should also update the cursor value on the graph. [You may also use your touchpad's scroll if there is no mouse]

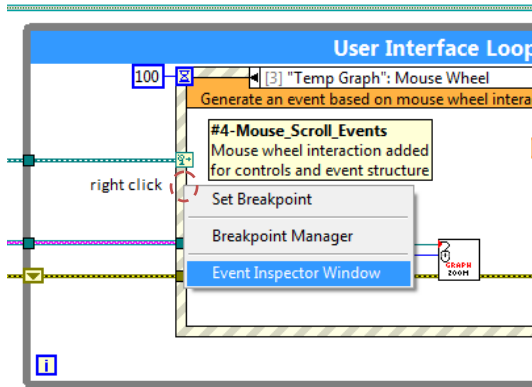


31. Explain: As I mentioned, this program captures UI interaction with events. In 2012, we introduced new templates and sample projects as starting points and references for best practice architectures. Many of these center around using Event Structures to capture user interaction. And, in adding mouse wheel support, we've added an event to capture any mouse wheel interaction with items on our front panel. This allows me to do things like better visualize the data I've taken by zooming in and out on my graph.

32. **DO:** Mouse wheel up and down on the “Temperature Graph”. This should automatically adjust the X and Y scale to simulate a zoom-in and zoom-out effect.



33. **Explain:** In this case, I’ve written a VI to programmatically adjust the scales of my graph. [Just to point out this is not built-in functionality] But, this mouse interaction could be captured to prompt any response your application requires.
34. **Explain:** I mentioned templates and sample projects, and their use of event structures. We consider it best practice for any application, which must respond to user input, to leverage an event structure. In 2013, we’re making it simpler to debug event-based programs. We have always been able to use standard LabVIEW debugging tools for events, but this wasn’t always sufficient for applications processing a high volume of events.
35. **DO:** Move the front panel to the left side of the screen.
36. **Explain:** To better facilitate troubleshooting event-based code, we’ve introduced the event inspector window. This window allows us to view what events have been processed by our event structure, and also see any that are waiting to be processed. If you are new to event programming, you may be interested to know there is a queue working behind the scenes of an event structure to ensure we do not lose any user interaction we’d like to process.
37. **DO:** Move the block diagram to the right side of the screen. Snap to the right side if using Windows 7. Right-click on the event structure border in the “User Interface Loop” and click “Event Inspector Window”. Move the Event Inspector Window to the right so front panel and inspector window are side-by-side.



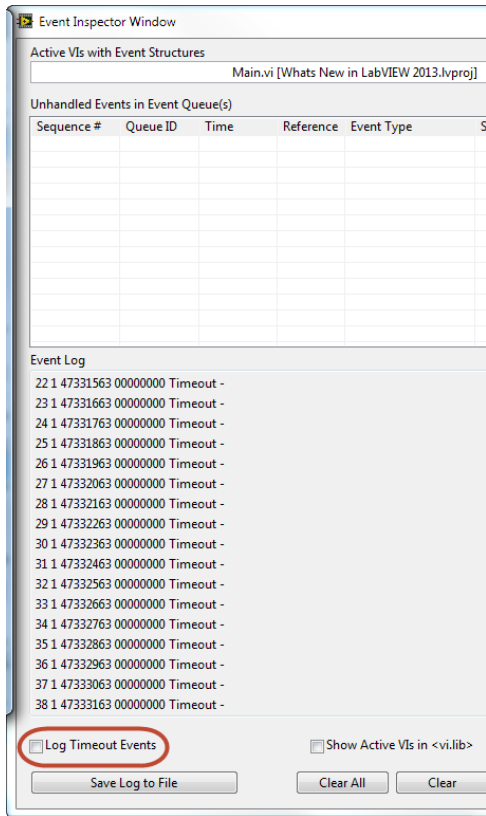
The screenshot displays the LabVIEW 2013 application interface. The main window shows a 'Temperature Graph' with a red line representing temperature over 200 samples. The graph has a y-axis from 58 to 82 and an x-axis from 0 to 200. A blue horizontal line is drawn at approximately 62.5. To the left of the graph are controls for 'Temperature Limit (°F)' (a dial set to 65), 'Temperature Alarm' (a green circle), and 'Application Timing (ms)' (a dial set to 1). At the bottom, there are buttons for 'Bookmark Example', 'System Status' (set to 'Idle'), 'Acquire Temperature Data' (with 'Acquire' and 'Stop' buttons), and 'Stop Button'.

On the right, the 'Event Inspector Window' is open, showing 'Active VIs with Event Structures' (Main.vi [Whats New in LabVIEW 2013.lvproj]) and 'Unhandled Events in Event Queue(s)'. Below this is an 'Event Log' table with columns for Sequence #, Queue ID, Time, Reference, and Event Type. The log contains 16 entries, all of which are 'Timeout' events.

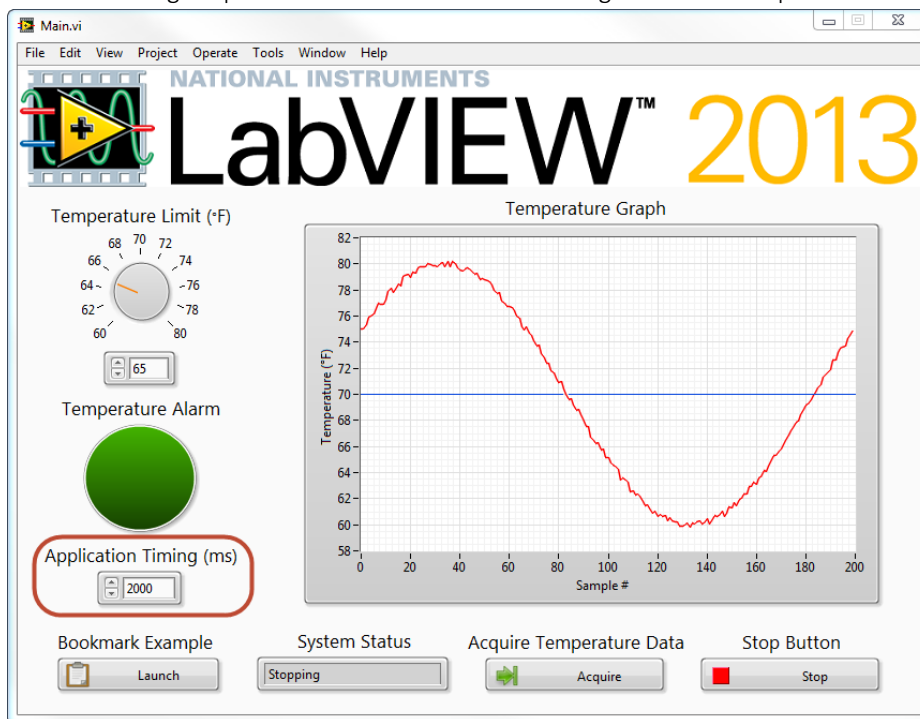
Sequence #	Queue ID	Time	Reference	Event Type
22	1	473331563	00000000	Timeout -
23	1	473331663	00000000	Timeout -
24	1	473331763	00000000	Timeout -
25	1	473331863	00000000	Timeout -
26	1	473331963	00000000	Timeout -
27	1	473332063	00000000	Timeout -
28	1	473332163	00000000	Timeout -
29	1	473332263	00000000	Timeout -
30	1	473332363	00000000	Timeout -
31	1	473332463	00000000	Timeout -
32	1	473332563	00000000	Timeout -
33	1	473332663	00000000	Timeout -
34	1	473332763	00000000	Timeout -
35	1	473332863	00000000	Timeout -
36	1	473332963	00000000	Timeout -
37	1	473333063	00000000	Timeout -

38. **Explain:** Notice that as I interact with my application, I see those events being captured in real-time. This can help me better understand how my application is behaving. And, the Event Inspector Window allows me to save my event logs to a text file, for documentation purposes.

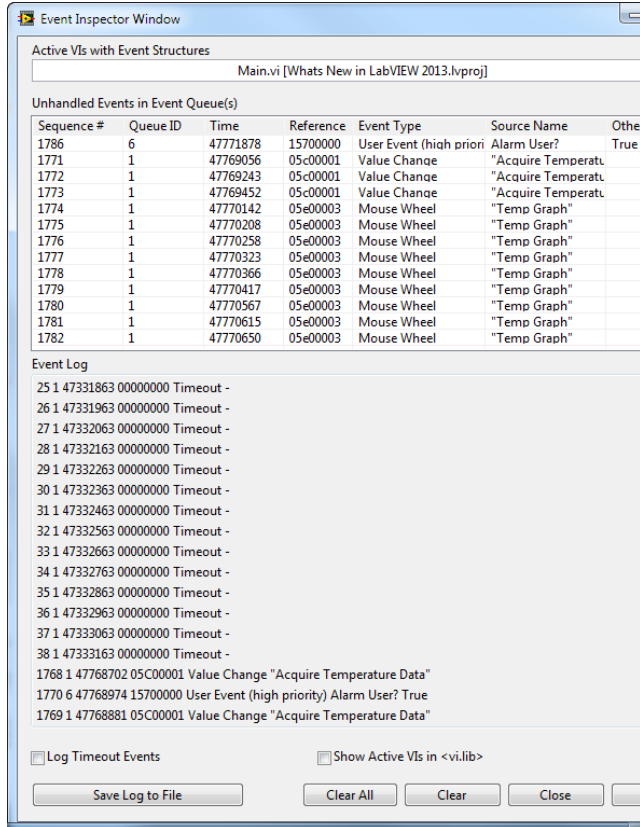
39. **DO:** Disable "Log Timeout Events".



40. DO: Increase the "Application Timing (ms)" to 2000ms or greater. This will slow down the event handling loop to better show events stacking in the event queue.



41. **DO:** Click “Acquire” or scroll the mouse wheel on the graph several times to build up a set of events in the event queue, shown in the Event Inspector Window.



42. **Explain:** If anyone has done event-based programming in past, they may be familiar with user events. These are simply events which are generated by logic in our program, rather than by a UI interaction. If anyone has taken the new Core 3 training, which uses the Queued Message Handler architecture, this leverages user events.

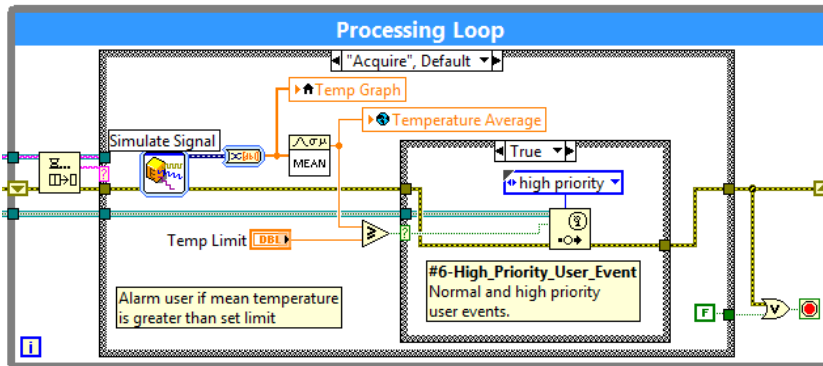
43. **Explain:** To build on templates and sample projects, and to encourage best practice architecture choices using event-based programming, we’ve implemented a highly requested feature, which is the ability to specify the priority of a user event. I mentioned there is a queue behind the scenes of an event structure, ensuring we do not lost any UI interaction from one loop iteration to the next. In past, user events always received the same priority as other events. This mean that system-critical tasks, such as a shutdown to avoid overheating, might require an extra queue or notifier in a large application, so they could be given the correct response priority.

44. **Explain:** In LabVIEW 2013, we’re helping users leverage the event-based architectures they are implementing, and handle these system-critical tasks with less development effort.

45. **DO:** Show the block diagram, and make sure the “Acquire” case is shown in the “Processing Loop”

46. **Explain:** In this example application, I'm going to alarm my user if the result of my temperature data acquisition and subsequent analysis is above the limit they've specified. However, I'm going to consider this a system-critical response, something that must be processed before all other actions, in the event an operator shutdown was required in response to this excessive temperature. I'm facilitating this with a high priority user event.

47. **DO:** Point out the temperature comparison and the "Generate User Event" function, which generates a "high priority" user event when called.

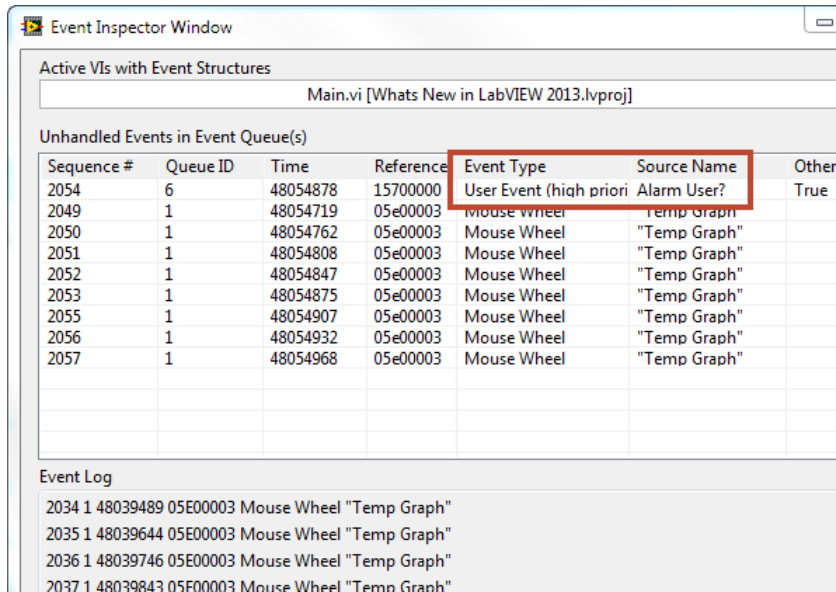


48. **Explain:** To show the way LabVIEW responds to high priority events, I've slowed down this application so we can observe what happens in the event queue when this high-priority event occurs.

49. **DO:** Bring the Event Inspector Window to the front so the UI and Event Inspector are side-by-side. Clear all events and ensure that timeouts are not being logged. Click the "Acquire" button, and then initiate several mouse wheel events on the "Temperature Graph" to fill up the event structure queue.

50. **Explain:** Notice that when a high priority event is fired, it automatically goes to the top of the event structure's list of items to process. Thus, we can make efficient use of this already-in-

place structure to handle system-critical responses.



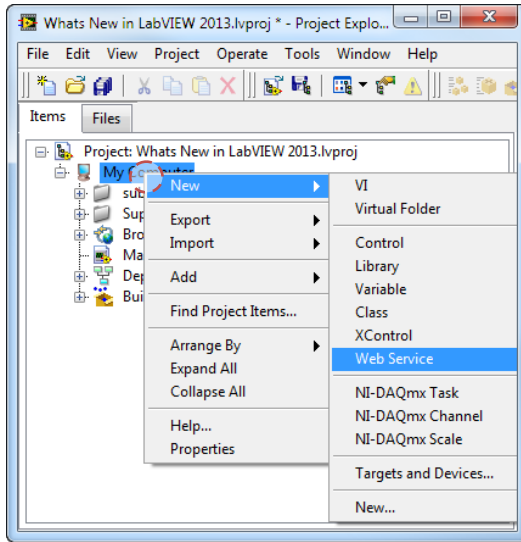
51. **DO:** Close the block diagram

52. **Explain:** We talked about some new development environment enhancements, available in LabVIEW 2013. I'd like to shift over toward the subject of deployment.

53. **Explain:** For anyone needing secure remote access to LabVIEW applications, whether running on a desktop or embedded target, web services are a great option for this communication. They allow SSL security, are industry standard and may be accessed by thin clients over a network, and are IT friendly. Web Services have been available in LabVIEW since 8.6, but we've gone to great lengths to simplify the process of creating, deploying, and debugging web services.

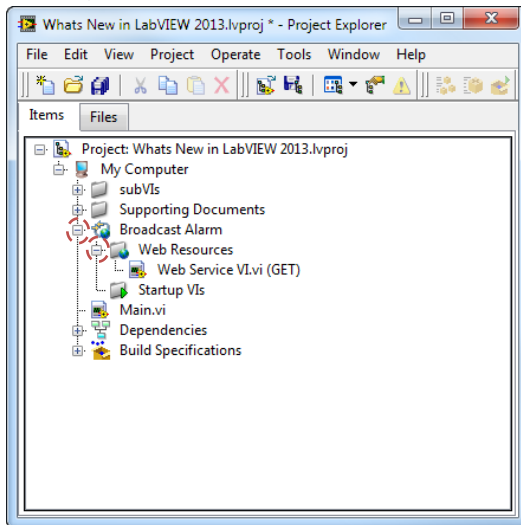
54. **Explain:** Web Services are now created as a project item. To do this, we simply right-click on "My Computer", then choose "New" and "Web Service" which creates a new item for us.

55. **DO:** Steps listed above

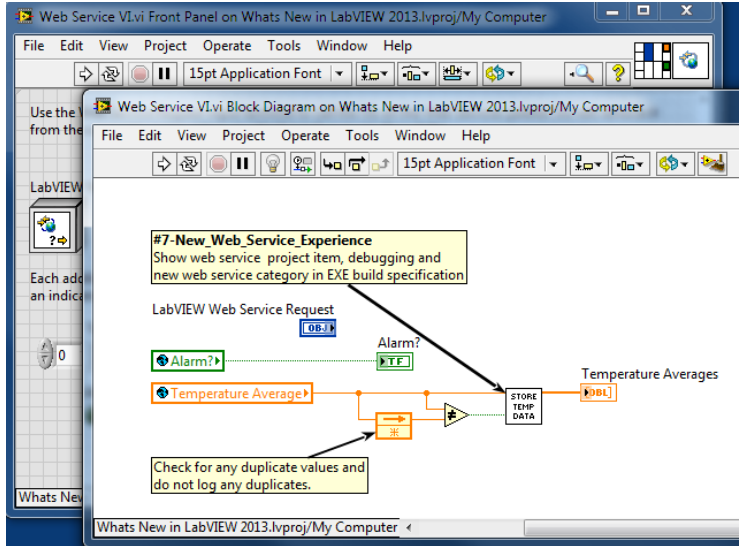


56. Explain: I've already added a web service, to show the new experience.

57. DO: Expand "Broadcast Alarm" web service project item and "Web Resources" folder.



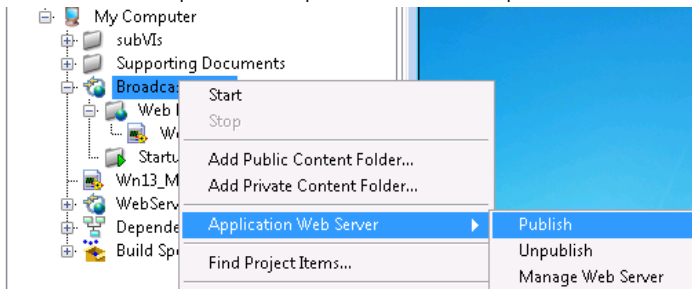
58. **DO:** Open the “Web Service VI.vi”, move both front panel and block diagram to the right side of the screen and overlay as shown.



59. **Explain:** In LabVIEW 2013, not only may I create and deploy web services to remotely access my LabVIEW applications, but I am able to debug the web method VIs, comprising that web service, directly from the LabVIEW project. This is an all new feature of LabVIEW 2013 web services. Moreover, you’ll notice this web service VI uses global variables. LabVIEW 2013 web services are created in the same namespace as regular LabVIEW applications, and therefore may use global variables and functional global variables to communicate, where before we’d have need to use TCP/UDP etc. to facilitate this cross-environment communication.

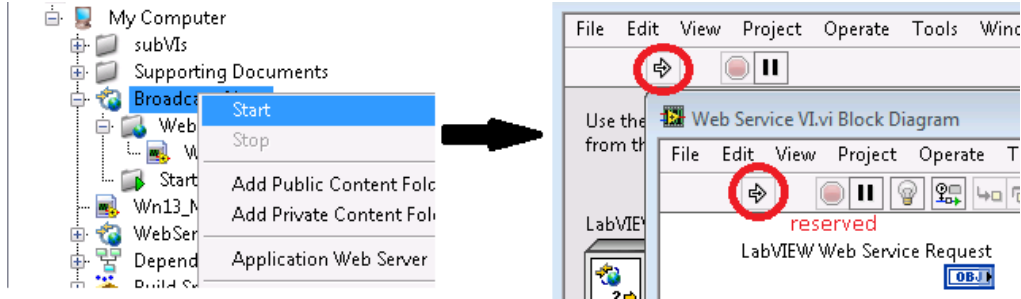
60. **Explain:** I can “publish” my web service, like you might be used to in LabVIEW.

61. **DO:** Right-click the “Broadcast Alarm” web service name, choose “Application Web Server” and show the options to “publish” and “unpublish”.



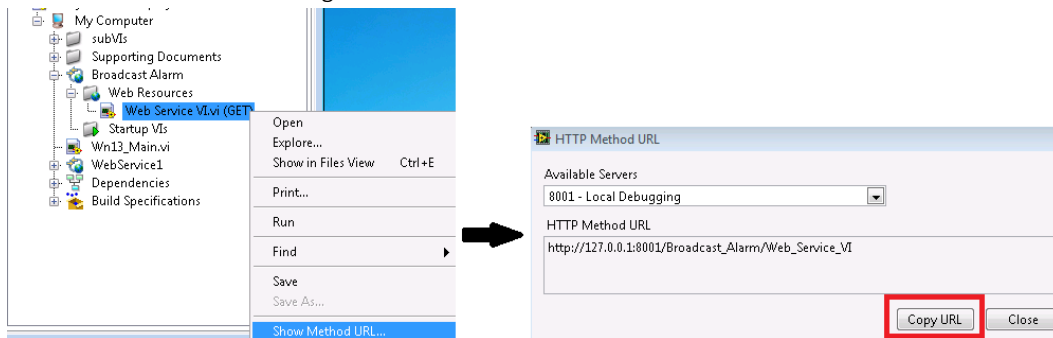
62. **Explain:** But, in LabVIEW 2013, I can begin a debugging session on my web method VI. I do this by right-clicking my “Broadcast Alarm” web service project items and choosing “start”. Notice that my Web method VI is now reserved and waiting to run.

63. **DO:** Right-click “Broadcast Alarm” and choose “start”. Then, point out the VI being reserved.



64. **Explain:** I may now use standard LabVIEW debugging tools to understand the behavior of my web service. To run the web service, I may obtain the URL of my web method VI.

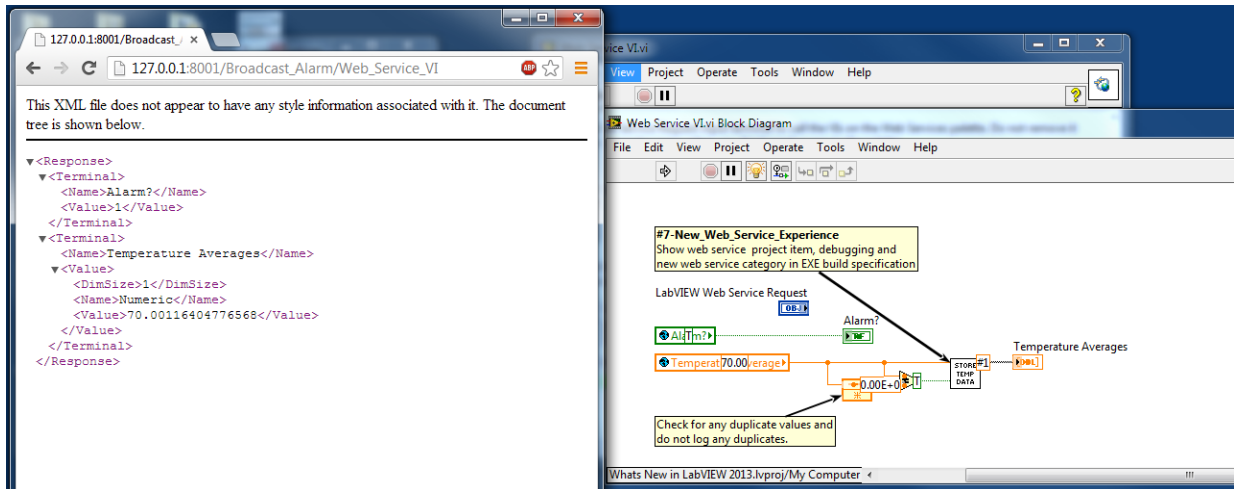
65. **DO:** Turn on highlight execution for the Web Service VI. Right-click on Web Service VI.vi (GET) and choose “Show Method URL”. Mention that LabVIEW automatically generates the URL for you. [This used to be a pain point for customers] Click on “Copy URL” and close the “HTTP Method URL” dialog.



66. **DO:** Open a browser and snap it to the left-side of the screen so both the browser window and Web Service VI block diagram are shown at a time. Paste the URL into the browser’s URL bar and hit enter.

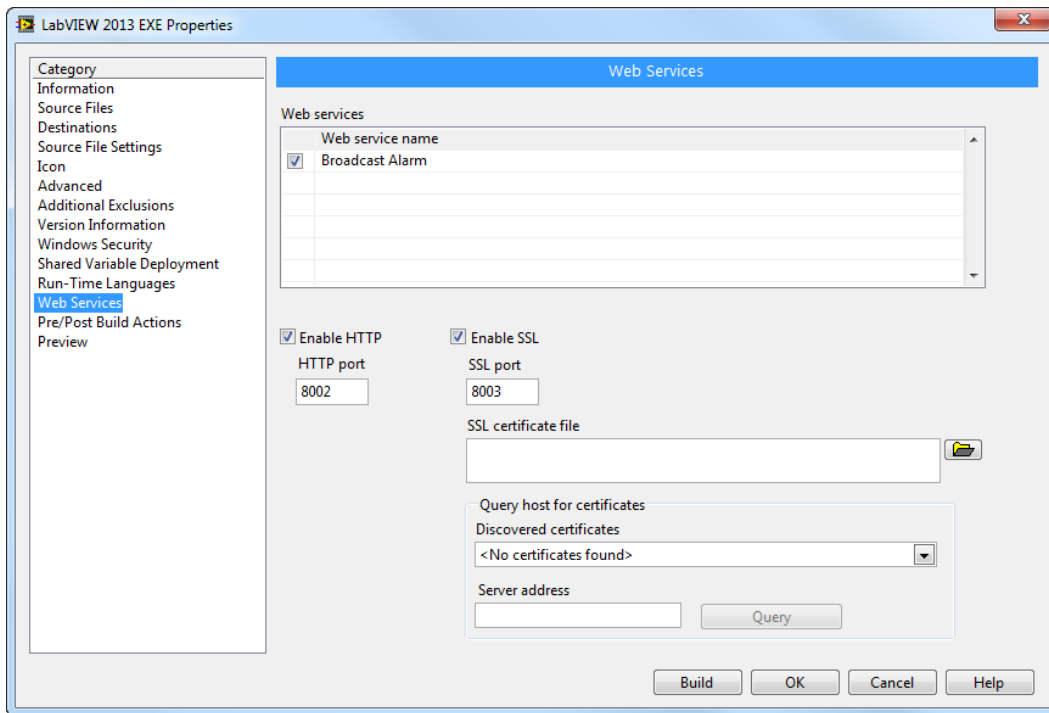
67. **Explain:** This starts my web method, allows me to observe the web service VI’s behavior with standard LabVIEW debugging tools [in this case highlight execution], and ultimately I

get my results in the browser window.



68. **DO:** Close the browser, close the web service VI, right-click on "Broadcast Alarm" web service item and choose "stop" to end debug session.
69. **DO:** Expand the "Whats New in LabVIEW 2013" project's build specifications, right-click on "LabVIEW 2013 EXE" and choose "Properties".
70. **Explain:** We did not forget about deployment when it comes to web services. In past there were many preparations necessary to deploy an application (EXE) which leveraged a web service. In 2013, we've streamlined this process by including a web service category in EXE build specs. Now, you can choose any web service included in your project, select SSL options, and build this web service into your EXE. It will then deploy automatically when the EXE runs, making remote access to both desktop and embedded LabVIEW applications a

breeze.



71. **DO:** Close the EXE build.

72. **Explain:** Now I'm going to switch back to slides and show you a few more deployment related features we've added in LabVIEW 2013.