Developing Algorithms Using LabVIEW MathScript RT Module: Part 1 – The LabVIEW MathScript Node

Publish Date: Nov 09, 2012 | 25 Ratings | **3.96** out of 5

Overview

This tutorial provides detailed information on working with the MathScript Node as part of the LabVIEW MathScript RT Module.

Table of Contents

1. What Is the LabVIEW MathScript RT Module?

LabVIEW MathScript RT is an add-on module for the LabVIEW Full and Professional Development Systems. It is designed to natively add text-based signal processing, analysis, and math into the graphical development environment of LabVIEW. With more than 800 built-in functions, LabVIEW MathScript gives you the ability to either run your existing custom .m files or create them from scratch. Using this native solution for text-based math, you can combine graphical and textual programming within LabVIEW because the text-based engine is part of the LabVIEW environment. With LabVIEW MathScript RT, you can choose whether graphical or textual programming is most appropriate for each aspect of your application.

To learn more about the LabVIEW MathScript RT Module, read this detailed introduction.

2. How Do I Use the MathScript RT Module?

There are two methodologies for interacting with the MathScript Engine: the MathScript Interactive Window and the MathScript Node.

Using the MathScript Interactive Window

The MathScript Interactive Window, accessed from the Tools menu, provides an intuitive interface to MathScript. With a command-line interface and a window to build batch files, the MathScript Interactive Window is designed to help you develop your scripts.

For more detailed information on using the MathScript Interactive Window, read “Part 2 – The MathScript Interactive Window.”

Using the MathScript Node

The MathScript Node is a structure on the LabVIEW block diagram that gives you the ability to put text-based MathScript code inline with G. You can define inputs and outputs on the node borders to pass data back and forth between the two paradigms. The node even supports debugging, with single steps, breakpoints, syntax highlighting, and a probe for intermittent values.

With the MathScript Node you can run .m file scripts from your LabVIEW graphical programs (known as virtual instruments or VIs). You can use the MathScript Node to insert textual algorithms into a VI and then use the LabVIEW graphical programming environment to instrument the scripts by adding knobs, slides, buttons, graphics, and other user controls and indicators.

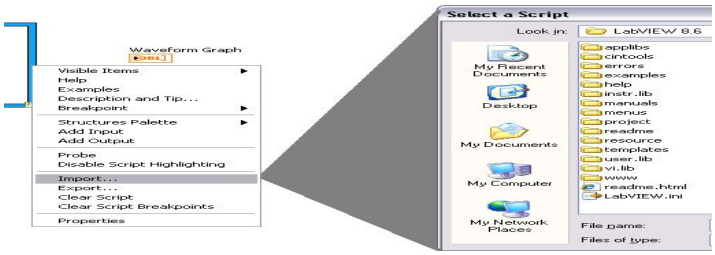

Figure 1. The blue rectangular region represents the MathScript Node.

3. Step-by-Step Example

This example assumes that you have basic experience working with LabVIEW. (Read the product information page for more details on working with the LabVIEW graphical development environment.) This example creates a sinusoid, executes a fast Fourier transform (FFT) on that sinusoid, and then outputs both graphs to a LabVIEW front panel.

1. Create a new VI.
2. Create a **Vertical Pointer Slide** (Controls **»** Modern) named *Amplitude* on the front panel.
3. Create two **Waveform Graphs** (Controls **»** Graph) named *Sinusoid* and *FFT Analysis* on the front panel.

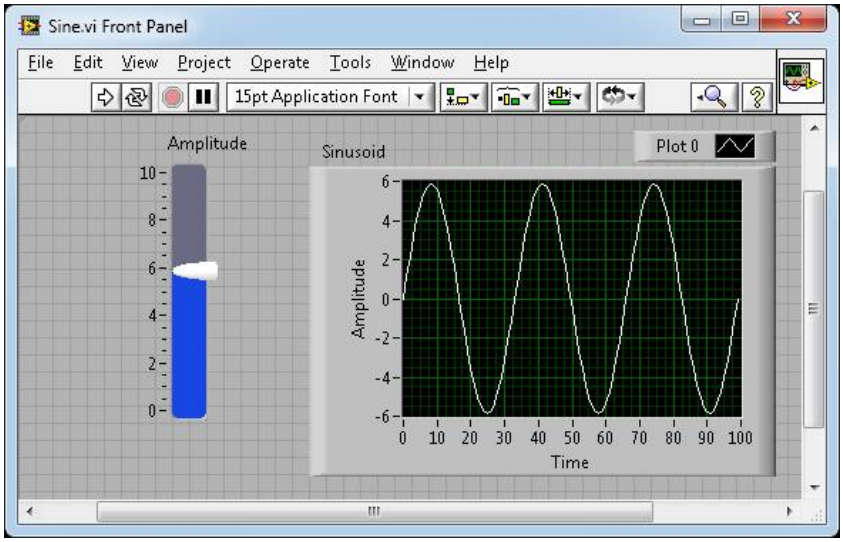

Figure 2. This shows the completed front panel for the example VI.

4. Select **Window » Show Block Diagram** to display the block diagram of the VI.
5. On the block diagram, select **View » Functions Palette** to display the functions palette.
6. Move the cursor over the icons on the programming palette to locate the structures palette.
7. Click the structures icon to display the structures palette.
8. Move the cursor over the icons on the structures palette to locate the MathScript Node.
9. Click the MathScript Node icon.
10. On the block diagram, click and drag the mouse in a rectangular shape to place the MathScript Node.

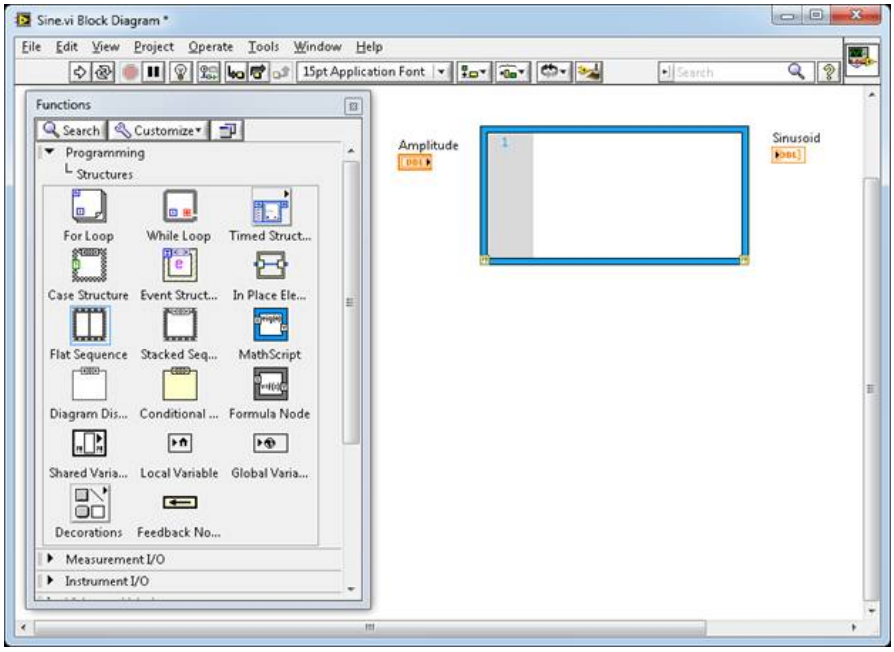

Figure 3. Place a MathScript Node on the block diagram by clicking and dragging the mouse in a rectangular shape.

11. Click inside the MathScript Node and type the following commands.
    x = linspace(0, 6*pi, 100);
    b = amp*sin(x);

    Notice that the “x = linspace(0, 6*pi, 100);” command creates a new variable x and populates that variable with 100 samples evenly distributed between 0 and 6*pi.

    (Optional) You also can right-click the MathScript Node and select Import from the shortcut menu to import an .m file that you created.

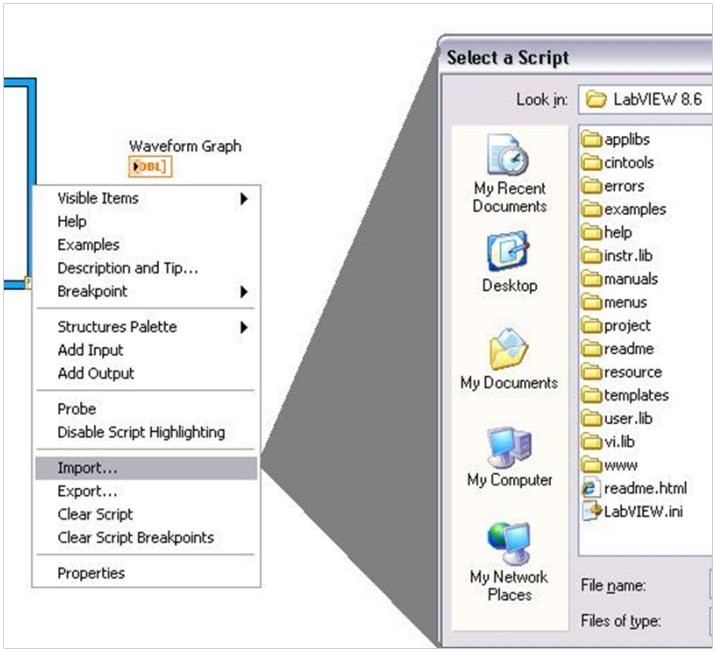

Figure 4. Easily import your custom .m files into the MathScript Node

12. Right-click the left side of MathScript Node frame and select **Add Input** from the shortcut menu.

    Type **amp** in the input terminal to add an input for the **amp** variable in the script. This creates an input to the MathScript Node that the **Amplitude** control terminal can be wired into.

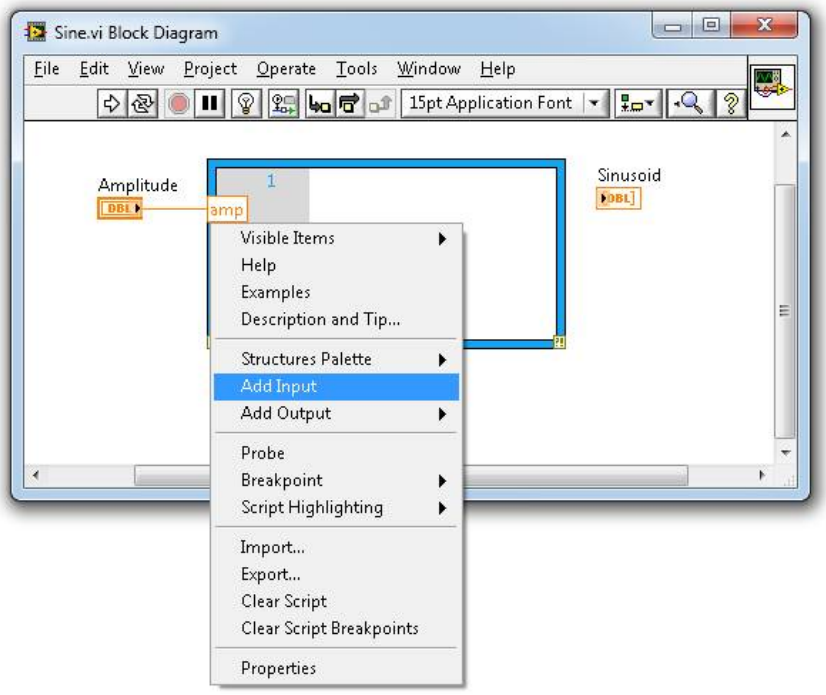

Figure 5. Create inputs on the MathScript Node to pass in data using graphical wires.

13. Right-click the right side of the MathScript Node frame and select **Add Output » sinusoid** from the menu. You'll notice that the menu provides all of the declared variables within the MathScript Node as an option to output. This creates an output from the MathScript Node that can be wired to the **Sinusoid** indicator terminal.

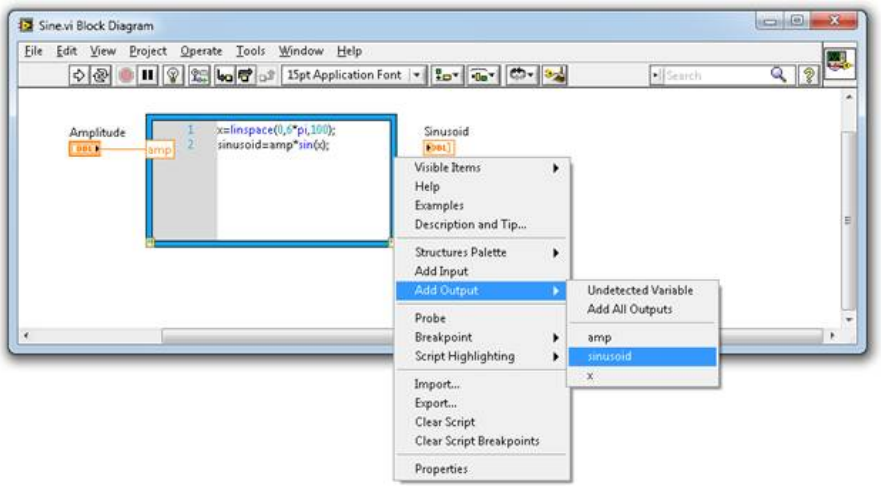

Figure 6. Create outputs on the MathScript Node to output MathScript variables.

14. The final block diagram should like Figure 7:

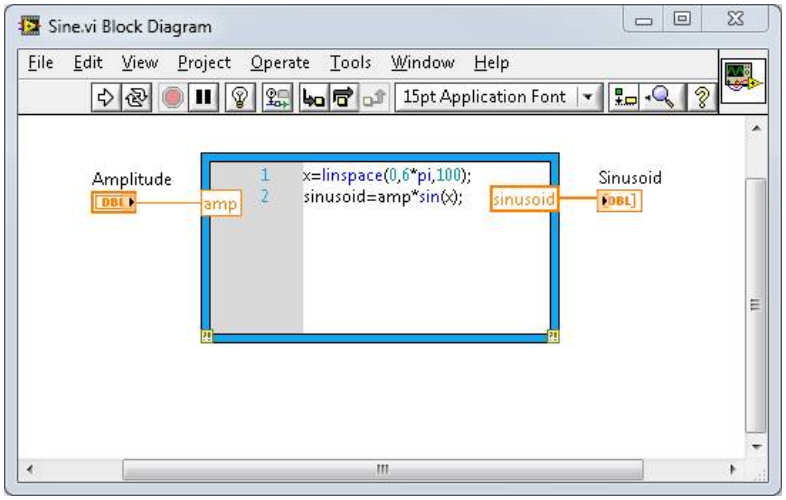

Figure 7. This is the final block diagram for sine.vi.

15. Select **Window » Show Front Panel** to display the front panel of the VI.
16. Drag the slider of the Amplitude control to approximately 3.
17. Click the Run button at the top of the front panel. Notice that the waveform graph updates with a sine wave.

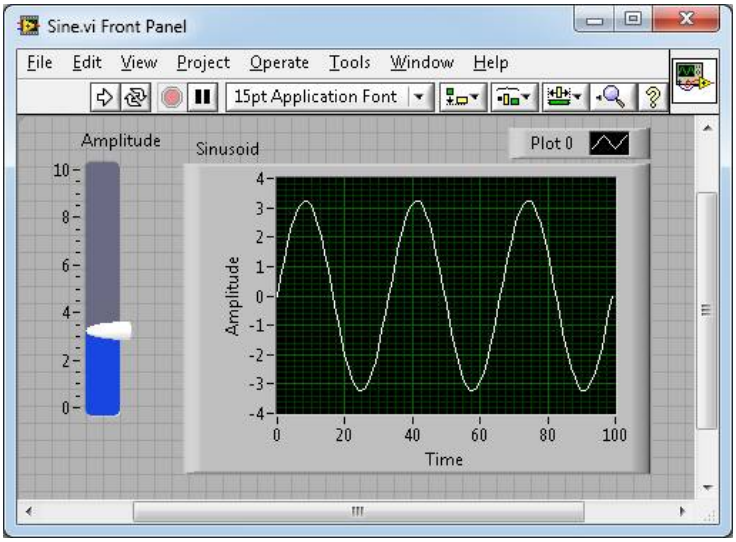

Figure 8. This is the Sine.vi front panel after running the application.

18. Drag the slider of the Amplitude control to a different value and run the VI again. Notice that the sine wave updates accordingly.
19. (Optional) Click the Run Continuously button, or the circular arrows, at the top of the front panel. Notice that the waveform graph updates each time you change the value of the Amplitude control.

4. Script Highlighting

Script highlighting uses colors to distinguish between different parts of a script in a MathScript Node. These colors improve the readability of the script and help you debug a script that contains errors or returns unexpected data. For example, you can use script highlighting to see when a user-defined function or a variable overrides a built-in MathScript function. Script highlighting is enabled by default, except for MathScript Nodes that were last saved in a version of LabVIEW prior to LabVIEW 8.5, and uses custom colors.

To disable script highlighting, right-click inside a MathScript Node and select **Script Highlighting » None.** To enable script highlighting, there are two options: syntax highlighting and data type highlighting.

Syntax Highlighting

Syntax highlighting displays script elements such as operators and comments in different colors. For example, if a variable or user-defined function overrides a built-in MathScript function, syntax highlighting colors that script element as a variable or user-defined function. You can customize the script highlighting colors for nodes in which it is enabled by using the *Syntax* section of the **MathScript** page of the **Options** dialog box:

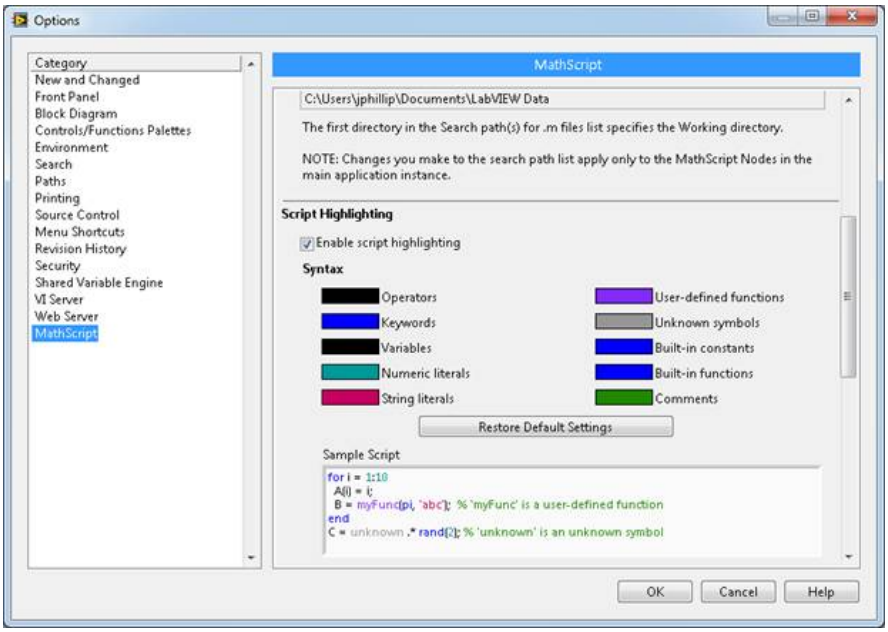

Figure 9. Customize syntax highlighting colors by using the MathScript category in the **Options** menu.

To enable syntax highlighting, right-click inside a node and choose **Script Highlighting » Syntax.**

Data Type Highlighting

Data type highlighting displays the data types of variables in different colors and font attributes. Data type highlighting gives you the ability to see when the script redefines a variable from one data type to another.

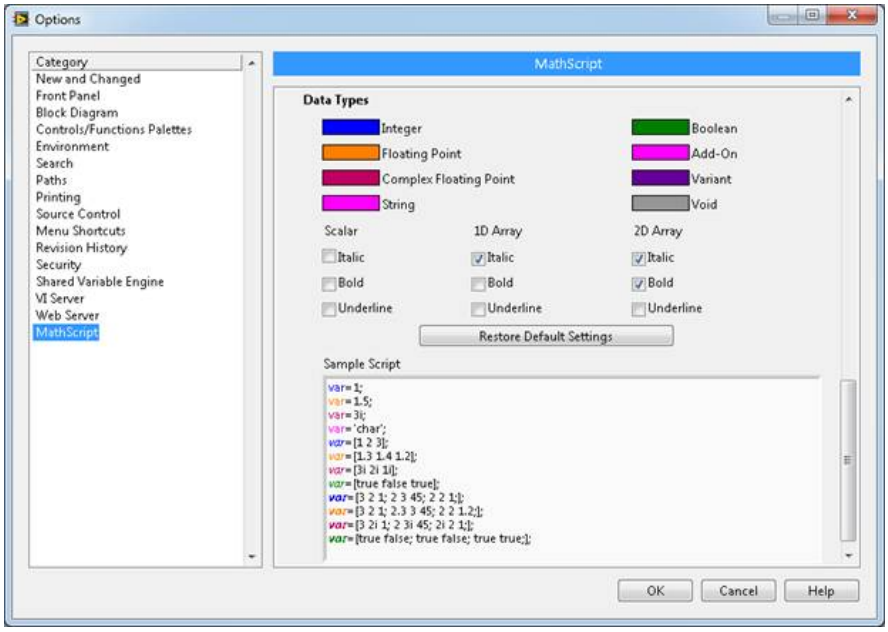

Figure 10. Customize data types highlighting colors by using the MathScript category in the **Options** menu.

To enable data type highlighting, right-click inside a node and choose **Script Highlighting » Data Types.**

5. Debugging Your Code Within the MathScript Node

You can perform in-node script debugging and use the MathScript probe to view intermediate variable values that are defined from within the node.

Using the MathScript Probe

Probes are invaluable tools for debugging LabVIEW applications because you can use them to see intermittent values. With the MathScript probe, you can debug your MathScript code by investigating unexpected results in the MathScript Node. You can use the probe to view the data in a MathScript Node as the VI runs. The probe displays a list of all variables you define in the script and previews variables you select. It also displays the output that MathScript generates from the script.

To view the MathScript probe, right-click inside a node and choose **Probe** (or just left-click over the node if the VI is running already).

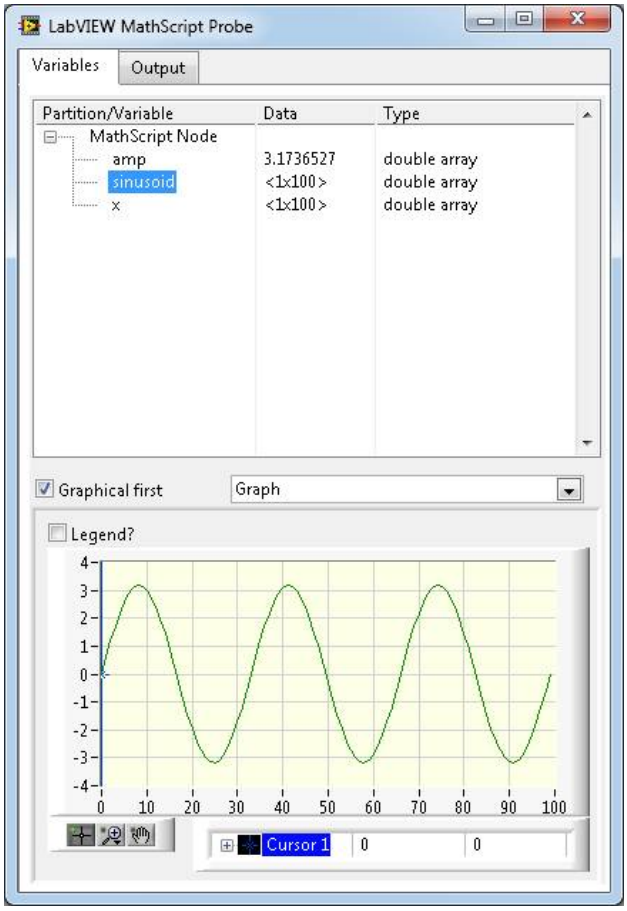

Figure 11. Use the MathScript probe to view intermittent script values when debugging your .m file code.

Using Breakpoints, Single Stepping, or Execution Highlighting in the MathScript Node

The MathScript Node also includes full debugging capabilities, including the following:

- Breakpoints
- Single stepping
- Execution highlighting

The gray region on the left side of the MathScript Node, used to interact with these tools, displays the following:

- Red error glyphs next to lines of the script that contain an error
- Warning glyphs
- Breakpoints

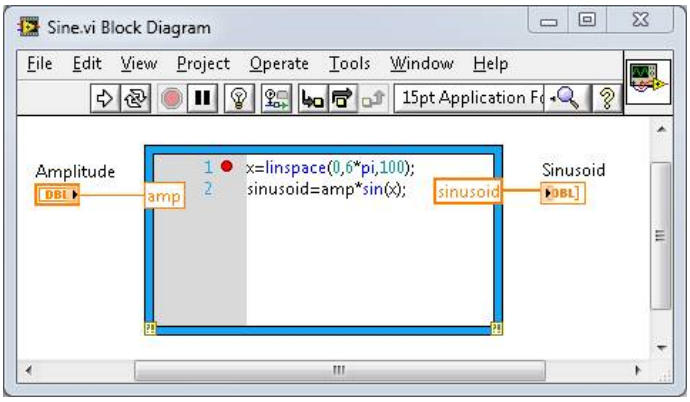

Figure 12. The gray region on the left side of the MathScript Node is used for debugging techniques such as setting breakpoints.

Just like breakpoints in graphical code, the block diagram brings the specific line of code where the breakpoint exists into key focus, and the VI pauses. You can then execute single stepping and execution highlighting using the appropriate buttons on the LabVIEW toolbar. The **error list** window also lists the lines of code where an error exists. When you select the error within this window and click the **Show Error** button, LabVIEW highlights the line of script that contains the error.

6. What Does This Yellow “Exclamation Point” Glyph Mean?

The LabVIEW MathScript RT Module is engineered for optimal performance in a real-time OS. Unlike most .m file environments, the MathScript Engine takes the loosely typed .m file language and strictly types it within the context of the LabVIEW graphical environment for efficient data type propagation throughout the underlying code. This strict typing ensures that LabVIEW can efficiently compile the text-based MathScript code for edit-time semantic and syntax error handling as well as in-node context help. The MathScript Engine does a significant amount of work at “edit time” within the LabVIEW development environment. There is a subset of functions that, when included in a script, prevent the compiler from statically compiling the script. These include functions such as **load, path, cd,** and **addpath,** which introduce syntax not included in the node.

If you call these functions from a MathScript Node or a user-defined function, a warning glyph appears in the MathScript Node next to the line from which you call the function.

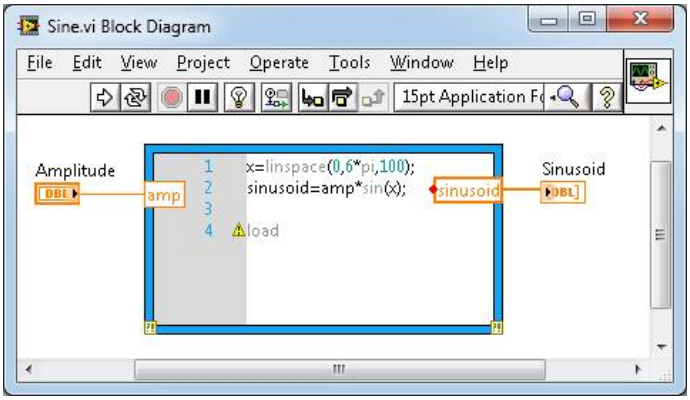

Figure 13. A warning glyph appears next to lines containing function calls that cause the MathScript Engine to execute with slower run-time performance.

The warning glyph indicates that LabVIEW operates with reduced error checking and slower run-time performance for the MathScript Node. To improve the error checking and optimize the performance of the MathScript Node, remove this function from scripts and user-defined functions. Also, do not change the MathScript search path list at run time. Instead, use the MathScript Options page to configure the default search path list for MathScript Nodes in the main application.