The Liberty BASIC Wire Frame Library

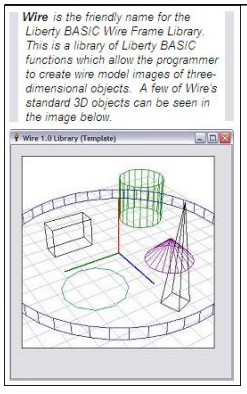
Tomas P Nally -
steelweaver52

Chapter 6: Using Wire (Strange Things, Reminders, and Tips)

This sixth and final chapter of the Liberty BASIC Wire Frame Library series originally appeared in the Liberty BASIC Newsletter, Issue #142. It is reprinted here with the permission of the author.

Table of Contents

To see the comprehensive documentation of Wire 1.0 functions, go here.



Wire, To Date

Wire was first rolled out as a beta product in July of 2005, in Liberty BASIC Newsletter|Liberty BASIC Newsletter, Issue 134 (*That article appears here as* Chapter 1: Introducing the Liberty BASIC Wire Frame Library. In the articles since that time, I've attempted to provide comprehensive discussions of ***Wire's*** features. These articles, in aggregate, can serve as additional documentation to the ***Wire*** library. The formal documentation of the functions was typically published with each article (as it is once again here).

With much of the detailed information about ***Wire*** already in the books, I thought this would be a good time to attempt to provide a wrap-up article. So here it is! After discussing a few strange things about ***Wire***, I will provide a few reminders and tips!

- Here is a handy index to all six articles:
- Chapter 1: Introducing the Liberty BASIC Wire Frame Library
- Chapter 2: The Liberty BASIC Wire Frame Library - Version 0.6
- Chapter 3: Version 1.0 of the Wire Frame Library is On the Horizon
- Chapter 4: Wire 1.0 Released (Making Complex Objects With Wire)
- Chapter 5: Using Wire (Focusing on the FF.LBWF.RequestObjectNameFromXY$() Function)
- Chapter 6: Using Wire (Strange Things, Reminders, and Tips)

A Few Strange Things About Wire

Wire is not a typical programming aid. There are a few unusual things about ***Wire***, and the programmer probably needs to work with ***Wire*** for a day or two before she can start to feel comfortable with the library.

Here are a few of those unusual things about ***Wire***:

- ***Wire 1.0*** is an *extensive* collection of functions, approximately 50! While some of ***Wire's*** functions are ten lines long or fewer, several of the functions are more than 100 lines long. The entire library is approximately 2,700 lines of code! So, before the programmer writes even the first line of her application, it might already be the longest program she has ever written. In all the templates and example programs I have written so far, I have placed ***Wire's*** functions at the end of the program in order to minimize the extent to which the library will intrude on the hard work that the programmer needs to accomplish.

- The *names* of ***Wire's*** functions are usually pretty long, and all function names begin with the characters, "LBWF.". In my view, the function names are too long to manually type into one's code each time a function is used. So, from the very beginning of ***Wire's*** development, I've always copied the function names (as well as the function's argument list) from elsewhere in the program. I discuss this more in the Tips section below.

- All 3D shapes drawn by ***Wire's*** functions will be drawn to a `graphicbox` called **`#main.wfscene`**. If your ***Wire*** program contains no other controls, it must at least have a `graphicbox` with this name. If the programmer really needs to, ***Wire*** *could draw* to a `graphicbox` with a name other than **`#main.wfscene`**. However, the programmer would need to go into the library's functions and change every instance of **`#main.wfscene`** to some other control name. As the programmer, you will have to determine for yourself whether it is worth the effort.

- Even though the ***Wire*** library is in the form of functions, most of these functions *do not* return a value. Most of the functions are used to create and organize the 3D data from which the wire model objects are made. For the few functions that *do* return values, I have tried to make that obvious by putting the word **"Request"** in the function name. (An example would be FF.LBWF.RequestObjectGeometricCenter$(objectname$), which returns the `x`, `y` and `z` `coordinates` of the geometric center of the object named.) Appropriately, functions of this type are referred to as **"Request Functions"**.

A Few More Things to Remember

Here are a few more things to remember which will help you use the ***Wire*** library efficiently:

- **What is meant by the "camera position"?** Imagine a collection of 3D objects on a table in your dining room. Imagine further that you are instructed to take a picture of those objects with your digital camera. The appearance of the objects in your picture depends on the placement of the camera. In the same way, ***Wire*** allows the programmer to designate the location of the camera in space by giving the camera an x-, y- and z-coordinate. The camera coordinates define the camera position.

- **What is meant by the "viewing center"?** In ***Wire***, the viewing center is that point in space to which the camera points. Going back to the objects on the dining table mentioned above, your digital picture of the objects will look differently depending on whether your camera is pointed to the center of the object group, to the left, or to the right.

- **What is meant by the "screen center"?** The screen center refers to that location in the **`#main.wfscene`** `graphicbox` to which the viewing center "maps". Think of it this way. Say that there is a very small cube located at the following `x, y, z coordinates`: (0,0,0). Since the cube is located at that point in space, I decide to make that same point (0,0,0) my viewing center. Say also that my camera is located in space at (200,200,200). Despite the fact that the camera has now been defined satisfactorily, and the object is defined satisfactorily, we still haven't

determined how the object will plot to the graphicbox. If we want, we could make the space point (0,0,0) plot to the left side of the `graphicbox`. Just as easily, we could make it plot to the top of the `graphicbox`, or to the right side. The point is this: the programmer must elect which point in the `graphicbox` will correspond to the point in space designated as the viewing center. Most of the time, the geometric center of the graphic box will serve quite nicely as the "screen center". So, if the **`#main.wfscene`** `graphicbox` is 480 pixels wide by 480 pixels high, then a good spot to place the screen center is at
(x=240, y=240) which is the center of the graphicbox.

A Few Tips for Writing Wire Applications

Table of Contents

- To start each ***Wire*** program, use the template that comes in the zip archive,

  LBWF10_template.zip

  - Details
  - Download
  - **13 KB**

  . The template contains the entire ***Wire*** library. It sets up the `graphicbox`, and draws seven standard objects. You can erase the code for these objects, one at a time, and then begin your own programming. (A screen shot of the template program is created at the top of this article.)

- To help you get a feel for the space in which ***Wire's*** objects are drawn, turn on the system's `axes`, and create a *grid object*. (***Wire's*** documentation describes the functions which allow you to do this.)

This will help you see the "depth" of drawing space, as well as the orientation of the `x`, `y`, and `z` `axes`.

- Experiment with ***Wire*** by making one object at a time. After you've made an object, change the camera location, and run the program again. This helps you get a feel for what your wire model objects look like from different vantage points.

- Remember, objects *will not show* up in the graphicbox until you issue one of the draw commands, such as FF.LBWF.DrawAllObjects().

- There are six different functions to move objects around ***Wire*** space. Test these functions one at a time. The one that the author uses most often is FF.LBWF.TranslateObject(ObjectName$, transX, transY, transZ). This function simply moves an object relative to its current position. Compare this to the FF.LBWF.MoveObjectAbsolute(ObjectName$, pX, pY, pZ) function, which moves an object to a specific point in space *regardless* of the original location of the object.

- The ***Wire*** library provides two functions which allow the programmer to *pause* the action for a number of miliseconds. During program development, use these functions liberally between object creation and drawing functions. This will slow down the object drawing process, and allow the user to see if the program is doing what she intended.

Have fun!

Tom Nally
Steelweaver52@aol.com

Note: There were some minor editing changes made to this article from the original. These changes were made with the author's permission and do not alter the informational content of the article in any way. Specifically, references to issues of the Liberty BASIC Newsletters were changed to also include chapters. This change was made for the singular purpose of providing clarity and ease in navigating throughout these ***Liberty BASIC Programmer's Encyclopedia*** *pages. -*
JanetTerra