The Liberty BASIC Wire Frame Library

Tomas P Nally -
steelweaver52

Chapter 4: Wire 1.0 Released (Making Complex Objects With Wire)

This fourth article of the Liberty BASIC Wire Frame Library series originally appeared in the Liberty BASIC Newsletter, Issue #137. It is reprinted here with the permission of the author.

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

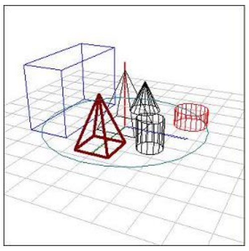
Wire 1.0 Released

Table of Contents

What is Wire?

Wire is the friendly name for the ***Liberty BASIC Wire Frame Library*** (LBWF). The library is a collection of 53 functions written in native Liberty BASIC code. These functions enable the programmer to easily create and manipulate 3-D "wire model" objects, such as boxes, cylinders, cones, lines, grids and pyramids.

A *wire model* object is a 3D object whose edges are visible but whose surfaces are transparent. For instance, a wire model cube would appear to be made from the wire of a coat hanger. See the figure below for a picture of wire model objects.



Wire model objects

For additional background on ***Wire***, see these three previous articles:

- Chapter 1: Introducing the Liberty BASIC Wire Frame Library
- Chapter 2: LB Wire Frame Library, Version 0.6
- Chapter 3: LBWF 1.0 ("Wire 1.0) Is On the Horizon

Wire 1.0 Is Ready For LB Programmers

In

LBWF10_template.zip

- Details
- Download
- 13 KB

, programmers can find the newest release of the ***Wire 1.0*** library. The library is contained within the source code program called LBWF10_template.bas. As the filename implies, this program is a rudimentary template for creating your first wire model images using ***Wire 1.0's*** fifty-three functions. When you run the program, it will create a `graphicbox` containing a grid, a set of axes, and several 3-D objects. This file is merely a starting point that the programmer can use to experiment with the functions. If you would like to examine the functions, they comprise the bottom 4/5ths of the LBWF10_template.bas source code file.

All of the fifty-three Wire 1.0 functions are comprehensively documented here in a companion article.

Wire 1.0 differs from the previous release, (0.6), in that it contains *two* new functions. One function, **FF.LBWF.RequestObjectNameFromXY$()**, was written to allow the programmer or user of ***Wire*** to select a 3D object merely by clicking on it with the mouse. An extensive discussion of **FF.LBWF.RequestObjectNameFromXY$()** can be found in the updated documentation, and additional information will be provided in the next article of this series.

This chapter of the series will discuss the second of the two new functions: **FF.LBWF.CreateComplexObject()**.

Making Complex Objects With Wire 1.0

- **Complex Objects 1: The Building Blocks of All Objects**

Before we talk about making complex objects with ***Wire***, let's talk about ***Wire's*** 3D objects in general. Three D objects are comprised of `vertices`, I like to call them "nodes", and `lines`. The lines of a 3D object are the visible edges of that object. `Nodes` are the points in space that form the endpoints of the lines. In fact, a line is defined by identifying the two end-nodes, which I like to call the `inode` and the `jnode`. A `node` is merely a point in space. It is defined by it's three spacial coordinates: `x`, `y` and `z`.

As an an example, let's create the data for a 3D object that consists of a single line in space. First, let's assign coordinates to `node 1`. `Node 1` will reside at the following coordinates:

```
x = 150, y = -125, z = 235
```

Next, let's define `node 2`. `Node 2` will reside at these coordinates:

```
x = -10, y = -350, z = 777
```

Last, let's define our 3D object, which consists of a single line in space. We will say that the `inode` of the object will be node 1, and the `jnode` of the object will be node 2. That's all there is to it. We've completely defined a simple, 3D object.

Of, course, we could make a slightly more complex object if we wanted to, a *pyramid* for example. A pyramid might conisist of 4 nodes and 4 lines to define the base of the pyramid. Then, we might define another node to serve as the peak of the pyramid. Last, we will need to define four additional lines, each one going from one of the base nodes up to the peak node. All of that data would be sufficient to define a 3D pyramind in space: 5 nodes and 8 lines.

- **Complex Objects 2: Is There a Difference Between Standard Objects and Complex Objects?**

The only difference between ***Wire's*** *standard objects* and *complex objects* is how the data is made. With standard objects, the coordinates of the nodes are assigned automatically by Wire's engine. For instance, if the programmer creates a box with the FF.LBWF.ObjectAssignCustomType() function.)//

Complex Objects 3: Creating Data With Complex Object Script

In the sections above, I discussed the fact that object data can be developed "by hand" nearly as easily as it can be developed by the ***Wire*** engine. Yet, we still need additional programming technology to allow this "external" object data to be accurately read by the engine.

Wire provides two technologies which help turn object data into actual complex wire model objects. The first technology is the set of syntactical rules used to specify node and line data. The rules are so few and simple that they really don't merit a name. Nonetheless, I will give them a name: *Complex Object Script*, or *COS*.

COS data can be written into an ascii text file using any text editor. When ***Wire*** reads this text file, it doesn't care what extension the text file has. However, I would like to standardize the process by giving these text files a **.cplx* extension. "Cplx" stands for "complex".

There are only two keywords in *COS*: *node* and *line*. Text lines which begin with the node keyword contain the ID of the node, followed by the `x`, `y` and `z coordinates` of the *node*. Text lines which begin with the line keyword contain the ID of the line, followed by the `inode` and the `jnode` of that line. These values should all be **space separated**, not comma separated.

Text lines which begin with any other word are simply ignored. For my personal use, I've decided to begin comment lines with the pound sign, ("#"). However, it really doesn't matter what the programmer does with any other lines. The only lines that register are those that begin with the keywords *node* or *line*.

A sample *.cplx text file is given below:

```
# Liberty BASIC Wire Frame Library
# Data for a ship

node 1   00.00  0.00    0.00
node 2   37.50  0.00  -25.00
node 3   43.75  0.00  -12.50
node 4   50.00  0.00  -12.50
node 5   50.00  0.00   12.50
node 6   43.75  0.00   12.50
node 7   37.50  0.00   25.00
node 8   12.50  0.00    0.00
node 9   31.25  0.00  -12.50
node 10  37.50  0.00    0.00
node 11  31.25  0.00   12.50

line 1      1   2
line 2      2   3
line 3      3   4
line 4      4   5
line 5      5   6
line 6      3   6
line 7      6   7
line 8      7   1
line 9      8   9
line 10     9   10
line 11     10  11
line 12     11  8
line 13     8   10
```

In the sample above, I've defined all the nodes first, and in order from node 1 to node 11. I've followed this by defining the lines in order for line 1 to line 13. But the ***Wire*** parser does not require that nodes be defined first; nor does the parser require that nodes be defined in numerical order. The programmer is perfectly free to intermix line and node definitions *in any order*. I wouldn't personally recommend that, but the ***Wire*** parser doesn't have a problem with it.

However, the parser does require that if your object has `n` number of nodes, then your nodes must be numbered `1` to `n`. If you skip a number in the sequence, then the ***Wire*** engine will have data mangement problems. As long as node numbers `1` to `n` inclusive are used, the order that they appear in the text file is irrelevant.

Likewise, if your complex object has `m` number of lines, then your lines must be numbered `1` to `m`. Again, as long as you use `1` to `m` inclusive, the order in which your line data appears doesn't matter.

Complex Objects 4: Reading *.cplx Files With The FF.LBWF.CreateComplexObject() Function

The second technology provided to turn *complex object data* into *actual wire model objects* is the **complex object function**. This function has three arguments as you will see from the line below:

Function FF.LBWF.CreateComplexObject(ObjectName$, FileName$, ObjectColor$)

The first argument, **ObjectName$**, is the name that the programmer or user wants to give to the complex object. The reader is reminded that all 3D objects are given names in ***Wire***. An object's name is used by the programmer to change or query an object's properties.

The second argument, **FileName$**, is the name of the *.cplx file that contains the complex object data. The format of the *.cplx file uses ***COS***, and is discussed in detail in the section above.

The third argument, **ObjectColor$**, is the color that the programmer or user chooses to give to the object. Colors, of course, are specified as strings just as they are elsewhere in the world of Liberty BASIC. (See the Liberty BASIC help file for tips on specifying colors.)

When one uses the complex object function in a ***Wire*** program, the code will probably look like this:

```
AAA = FF.LBWF.CreateComplexObject("Spacecraft1",
"Spacecraft01.cplx", "darkblue")
```

At this time, the reader should be reminded that, except for the "Request" functions, most of ***Wire's*** functions do not return values. Rather, most of them are used for the purpose of creating and organizing data for 3D objects. So, in the statement above, the variable **AAA** will not contain a meaningful value after the statement executes.

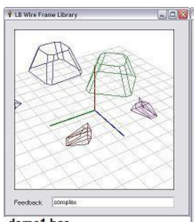
Once the statement above executes, a new object called "Spacecraft1" will be assembled in memory. It can be shown, moved, hidden, rotated and have its color changed just like any other wire model object.

Complex Object Demos 1 and 2

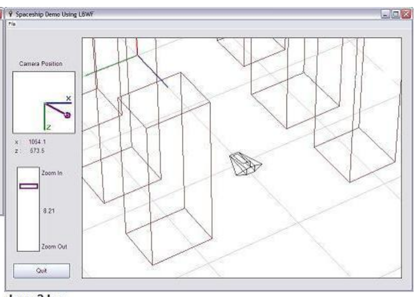
ComplexObjDemos.zip

- Details
- Download
- 28 KB

contains two demo programs featuring complex objects. Screen captures from both programs are shown below.



demo1.bas



demo2.bas

Table of Contents

Demo1 is a demo that requires no action from the user. Just patiently watch while the complex objects have their properties changed and undergo rotations. On slower computers, it might take a minute or two to finish.

Demo2 creates a spaceship in the midst of a number of rectangular buildings. Use the dial control on the left-hand side of the window to instantly change the position of the camera. Clicking and dragging the small circular button will change the camera postion. Click and drag the slider control to zoom in and zoom out.

Tom Nally
Steelweaver52@aol.com

Chapter 1: 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)

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