The Liberty BASIC Wire Frame Library

Tomas P Nally -
steelweaver52

Chapter 2: The Liberty BASIC Wire Frame Library - Version 0.6

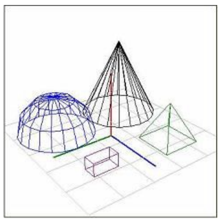
Newest Features in the LB Toolkit For the Creation of Wire Models

Table of Contents

This second article of the Liberty BASIC Wire Frame Library series originally appeared in the Liberty BASIC Newsletter, Issue #135. It is reprinted here with the permission of the author.

Introduction

The ***Liberty BASIC Wire Frame Library*** (LBWF Library) is a collection of functions which a programmer can use to create wire model objects in the Liberty BASIC programming language. A *wire model object* is a 3-D object whose edges are shown, but whose surfaces are transparent. A number of wire model images are shown in the figure below. The ***LBWF Library*** was previously discussed in Chapter 1: Introducing the Liberty BASIC Wire Frame Library.


Wire Frame Objects created using
the Liberty BASIC Wire Frame Library

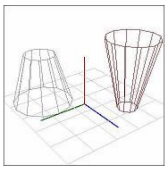
Version 0.6 of the ***LBWF Library*** contains 51 functions, all written in native Liberty BASIC. This article discusses the newest features of the ***Library***. Additionally, thorough documentation of all 51 functions can be found in this companion article.

Version 0.6 - New Wire Frame Objects

The previous version of the ***LBWF Library*** supported the creation of six standard 3-D objects: *boxes, cylinders, cones, pyramids, polygons and grid* objects. Version 0.6 of the Library brings the number of standard objects up to nine with the inclusion of three new standard objects: *domes, lines and cyl2s.*

In the ***LBWF Library***, a *dome* is essentially the top half of a sphere. One can be seen in the figure provided above and right. A *line*, of course, is simply a *line in space*. Despite the fact that a *line* is the simplest of all ***LBWF*** standard objects, it requires more arguments -- 8 -- to define it than any other object.

A *cyl2* in the ***LBWF Library*** is similar to a cylinder, except that the top surface of a *cyl2* can have a different radius than the bottom surface. Therefore, a *cyl2* can resemble a cone with its pointed top cut off. Or, if one makes the radius of the top surface of a *cyl2* larger than the radius of the bottom surface, the resulting object looks something like a cup. For an example of two *cyl2s*, see the figure below.


cyl2 - a new object in version 0.6

Version 0.6 - All Objects Have a "Type" Property

With version 0.6 of the ***Library***, all objects now have a *type* property. During the object-creation process, one of the default types is automatically assigned to the newly-created object. The nine default types are the same as the nine standard objects: box, cylinder, cone, pyramid, etc. After an object is created, however, the programmer can give it a *custom type*. A new function -- **FF.LBWF.ObjectAssignCustomType()** -- was created for this purpose.

The objective of aggregating objects by type is to give the programmer a way of operating on multiple objects with a single function. The four functions identified below allow the programmer to operate on multiple objects of the same type:

- **Function FF.LBWF.HideObjectsOfType(ObjectType$)**
- **Function FF.LBWF.ShowObjectsOfType(ObjectType$)**
- **Function FF.LBWF.TranslateObjectsOfType(ObjectType$, transX, transY, transZ)**
- **Function FF.LBWF.DrawObjectsOfType(ObjectType$)**

When ***ObjectType$*** is an argument of a function, the function doesn't care whether the type specified is a *custom type* or one of the *default types*. To read the detailed documentation of these functions, go to the companion article.

Version 0.6 - Objects Can Be Rotated

In the previous version of the ***Library***, a function was provided to *translate* objects in ***LBWF*** space. (To *translate* an object means to move it without rotating it.) In the current version of the ***Library***, three functions are provided to *rotate an object* about the object's own center:

- **Function FF.LBWF.RotateObjectAboutY(ObjectName$, YRotationInDegrees)**
- **Function FF.LBWF.RotateObjectAboutX(ObjectName$, XRotationInDegrees)**
- **Function FF.LBWF.RotateObjectAboutZ(ObjectName$, ZRotationInDegrees)**

The reader might be curious as to why a single function is provided to translate objects, while three functions are provided to rotate objects. The ***Library***'s author thought that it wouldn't be *unusual* to want to translate an object in more than one axis direction simultaneously, but it would be unusual to want to rotate an object simultaneously around two or three axes. So, one function each was provided to rotate objects about each of the three axes.

Version 0.6 - *Request Functions* Return Information About Objects

After the programmer creates and moves objects, she also might wish to know where the objects reside, how big they are, what type they are, what their color is, etc. The programmer can keep data records of this information as the objects are created and moved, but she doesn't have to. This is because version 0.6 of the Library provides eight "request" functions.

Six of the request functions return information about objects. For instance, the function **FF.LBWF.RequestObjectVisibleState(ObjectName$)** returns the *visibility property* of the object named as the argument. (If an object is *hidden*, its visibility has been set to zero; otherwise, its visibility is one.) In order to query a property of an object, the programmer must know the object's name. So, even if the programmer is not keeping track of object visibility states or object colors, the programmer should at least be keeping track of object names.

The other two request functions return information about the ***LBWF*** data resources available to the programmer. These are explained in greater detail in the documentation.

Version 0.6 - Simplifying the Control of the Camera

In Version 0.5 of the ***Library***, a single function was provided for camera control. The name of this function (with arguments shown) is FF.LBWF.ScreenCenter(ScrCenterX, ScrCenterY). The **FF.LBWF.Camera()** function allows the programmer to control the camera location in space (the first three arguments); the point in space at which the camera is pointed (the *viewing center*, 4th through the 6th arguments); and the *zoom factor* (the last argument).

In reality, however, it's unlikely that the programmer would want to change both the camera location and the viewing center and the zoom factor at the same time. It's much more likely that the programmer will want to change the camera location by itself (during a "fly by"), or the viewing center (during panning), or the zoom factor (during zooming).

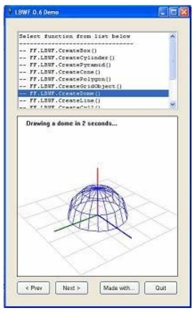
For this reason, Version 0.6 of the LBWF Library offers three new functions to control each of these three aspects separately:

- Function FF.LBWF.CameraLocation(CamX, CamY, CamZ) - for controlling the camera location.
- Function FF.LBWF.ViewingCenter(VCtrX, VCtrY, VCtrZ) - for controlling the viewing center.
- Function FF.LBWF.ZoomFactor(ZoomFac) - for controlling the zoom factor.

This arrangement strikes the ***Library***'s author as a much more reasonable way to control these aspects of camera operation. Note, however, that the original **FF.LBWF.Camera()** function remains in the library, and will remain in the library for the foreseeable future.

Version 0.6 - New Miscellaneous Functions

Thanks to the good observations of savvy LB programmer - Welopez, the ***LBWF Library*** has an alternate pause routine. - Welopez noticed that the original routine, **FF.LBWF.PauseMilliseconds(DelayMS)**, uses 100% of system resources. - Welopez recommended that a pause routine be developed that uses Liberty BASIC's native **TIMER** command. The new pause routine does just that, and is named **FF.LBWF.PauseUsingTimer(DelayMS)**. The demo program that accompanies this chapter will use the new pause routine exclusively. The original pause routine, however, will remain in the ***LBWF Library***.



```
Select function from list below
--------------------------------
-- FF.LBWF.CreateBox()
-- FF.LBWF.CreateCylinder()
-- FF.LBWF.CreatePyramid()
-- FF.LBWF.CreateCone()
-- FF.LBWF.CreatePolygon()
-- FF.LBWF.CreateGridObject()
-- FF.LBWF.CreateDome()
-- FF.LBWF.CreateLine()
```

Drawing a dome in 2 seconds...

Demo program accompanying
LBWF Library Version 0.6

Another new miscellaneous function is called **FF.LBWF.ZeroAllData()**. Though the Library has no provision for deleting individual objects in Version 0.6, the *ZeroAllData* function will delete *all* objects, and essentially reset the system back to its original state at startup.

This function is used extensively in the demo program accompanying Version 0.6. (In fact, it was written *for* the demo program.) The demo program provides numerous routines which illustrate the capabilities of many of the ***LBWF*** functions. Since the demo program allows the user to try any function over and over again, the **FF.LBWF.ZeroAllData()** function is used to erase all pre-existing 3-D objects before each new demo is executed. That way, the user doesn't create and accumulate numerous copies of the same object if she tries the same function numerous times. I only mention this to inform the programmer that the **FF.LBWF.ZeroAllData()** function is not one that has to be used routinely. I personally do not anticipate using it very often in my own programs.

The Demo Program

A demo program called WF_Demo06.bas accompanies the release of Version 0.6 of the ***LBWF Library***.

This program displays 17 Library functions in a listbox. Clicking on any of the functions will provide a short demonstration of the capabilities of the function. Typically, the program creates a delay between the selection of the function and the execution, so the user is advised to wait a few seconds for the function to execute before going to the next one.

I am also including a new template LBWF06_template.bas that can be used as your starting point for any wire frame projects. This file contains version 0.6 of the library; it sets up a `graphicbox` for the display of shapes; and it creates a single graphic object.

Subsequent Versions of the LBWF Library

Table of Contents

The next version of the ***LBWF Library***, likely to be Version 1.0, will provide a way for the programmer to create *complex* objects. A complex object will be one in which the user has complete control over the numbers and locations of ***nodes*** and ***lines***.

Also, look forward to additional examples of how to use the 51 functions that currently make up the ***LBWF Library***.

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 future issues of the Liberty BASIC Newsletters were changed to refer to further 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