Chapter 4: Documentation of Wire 1.0 (The Liberty BASIC Wire Frame Functions)

Tom Nally -
steelweaver52

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

Contents

- Function FF.LBWF.CreateBox(BoxName$, xdim, zdim, boxheight, BoxColor$)
- Function FF.LBWF.CreateCylinder(CylName$, radius, numSides, cylheight, CylColor$)
- Function FF.LBWF.CreatePyramid(pyrName$, xdim, zdim, pyrheight, pyrColor$)
- Function FF.LBWF.CreateCone(ConeName$, radius, numSides, coneheight, ConeColor$)
- Function FF.LBWF.CreatePolygon(PolyName$, radius, numSides, PolyColor$)
- Function FF.LBWF.CreateGridObject(GridName$, NumUnitsXdirection, NumUnitsZdirection, UnitSize, GridColor$)
- Function FF.LBWF.CreateDome(DomeName$, radius, numSides, DomeColor$)
- Function FF.LBWF.CreateLine(LineName$, x1, y1, z1, x2, y2, z2, LineColor$)
- Function FF.LBWF.CreateCyl2(Cyl2Name$, radiusBottum, radiusTop, numSides, cyl2height, Cyl2Color$)
- Function FF.LBWF.CreateComplexObject(ObjectName$, FileName$, ObjectColor$)

Functions Which Set the Properties of Objects

- Function FF.LBWF.ObjectSetColor(ObjectName$, NewColor$)
- Function FF.LBWF.ObjectSetLineThickness(ObjectName$, LineThickness)
- Function FF.LBWF.HideObject(ObjectName$)
- Function FF.LBWF.ShowObject(ObjectName$)
- Function FF.LBWF.ObjectAssignCustomType(ObjectName$, CustomType$)
- Function FF.LBWF.HideObjectsOfType(ObjectType$)
- Function FF.LBWF.ShowObjectsOfType(ObjectType$)
- Function FF.LBWF.HideAllObjects()
- Function FF.LBWF.ShowAllObjects()

Functions Which Translate or Rotate Objects

- Function FF.LBWF.TranslateObject(ObjectName$, transX, transY, transZ)

Functions Which Affect the View

- **Function FF.LBWF.Camera(CamX, CamY, CamZ, VCtrX, VCtrY, VCtrZ, ZoomFac)**

In order to create a wire model view, the programmer must establish the point in space where the camera resides, and then identify the point in space where the camera points.

- CamX, CamY, CamZ - The x-, y- and z-coordinates of the camera location.
- VCtrX, VCtrY, VCtrZ - The x-, y- and z-coordinates of the point in space where the camera is pointed.
- ZoomFac - The zoom factor. Start out by assigning ZoomFac = 1. If the image is too large or too small, adjust accordingly.

Calling this function doesn't change the picture in the `graphicbox` instantly. Rather, the picture will not change until a subsequent FF.LBWF.DrawObject(), FF.LBWF.DrawAllObjects(), or FF.LBWF.DrawObjectsOfType() function is called.

As of version 0.6 of the library, the programmer can also change the camera location, the viewing center, and the zoom factor individually.

Top of Page

- **Function FF.LBWF.CameraLocation(CamX, CamY, CamZ)**

Typically, the programmer has no need to change the camera location, the viewing center, and the zoom factor all at once. The **FF.LBWF.CameraLocation()** function was created so that the programmer can change the location of the camera without worrying about the viewing center or the zoom factor. This function can be seen as a subset of function FF.LBWF.Camera().

- CamX - The x-coordinate of the camera location.

>CamY - The y-coordinate of the camera location.

>CamZ - The z-coordinate of the camera location.

Calling this function doesn't instantly change the picture in the `graphicbox`. Rather, the picture will not change until a subsequent FF.LBWF.DrawObject(), FF.LBWF.DrawAllObjects(), or FF.LBWF.DrawObjectsOfType() function is called.

Top of Page

- **Function FF.LBWF.ViewingCenter(VCtrX, VCtrY, VCtrZ)**

The viewing center is that point in space where the camera is pointed. This function was created to allow the programmer to change the viewing center without calling the FF.LBWF.Camera() function, which requires seven arguments. **FF.LBWF.ViewingCenter()** can be considered a subset of FF.LBWF.Camera().

- VCtrX - The x-coordinate of the point in space where the camera is pointed.
- VCtrY - The y-coordinate of the point in space where the camera is pointed.
- VCtrZ - The z-coordinate of the point in space where the camera is pointed.

Calling this function doesn't instantly change the picture in the `graphicbox`. Rather, the picture will not change until a subsequent FF.LBWF.DrawObject(), FF.LBWF.DrawAllObjects(), or FF.LBWF.DrawObjectsOfType() function is called.

Top of Page

- **Function FF.LBWF.ZoomFactor(ZoomFac)**

The zoom factor effects how large the objects appear when they are drawn in the graphicbox. This function requires a single argument. Typically, the programmer needs to experiment with the size of the zoom factor in order to make her objects appear the right size. Start with a zoom factor of 1, and then change it as needed. (The programmer can also set the zoom factor by calling the function FF.LBWF.Camera().)

- ZoomFac - The zoom factor.

Calling this function doesn't instantly change the picture in the `graphicbox`. FF.LBWF.DrawObject(), FF.LBWF.DrawAllObjects(), or FF.LBWF.DrawObjectsOfType().

Top of Page

- **Function FF.LBWF.ScreenCenter(ScrCenterX, ScrCenterY)**

This function establishes the "screen center". Essentially, the screen center is the point in your Liberty BASIC `graphicbox` where the viewing center is mapped. The viewing center is discussed here.

- ScrCenterX - Usually, this is the x-coordinate of the center of the graphicbox
- ScrCenterY - Usually, this is the y-coordinate of the center of the graphicbox

Occasionally, the programmer does not want to plot the 3D image at the center of the `graphicbox`. In that case, move ScrCenterX and ScrCenterY accordingly.

Top of Page

- **Function FF.LBWF.PointCameraAtObject(ObjectName$)**

Recall that the viewing center is that point in space to which the camera points. The **FF.LBWF.PointCameraAtObject()** sets the viewing center to the geometric center of the object named as an argument in the function.

- ObjectName$ - The name of the object at which you want to point the camera.

In other words, if the programmer wanted the camera to point an an object called "RedCylinder", she would issue this function call

```
AAA = FF.LBWF.PointCameraAtObject("RedCylinder").
```

Calling this function doesn't instantly change the picture in the `graphicbox`. Rather, the picture will not change until a subsequent FF.LBWF.DrawObject(), FF.LBWF.DrawAllObjects(), or FF.LBWF.DrawObjectsOfType() function is called.

Top of Page

Functions Affecting the Graphicbox

- **Function FF.LBWF.BackGroundColor(BGColor$)**

Calling this function will identify the background color of the graphic box. After this function has been called, the graphicbox will be painted with the color every time the FF.LBWF.ClearGraphicScreen() function is called.

- BGColor$ - A string identifying the background color of the graphicbox, using the typical Liberty BASIC methods for indicating colors.

Top of Page

- **Function FF.LBWF.ClearGraphicScreen()**

This function clears the graphicbox and paints the background with the color identified in the FF.LBWF.BackGroundColor() function. The **FF.LBWF.ClearGraphicScreen()** function requires no arguments.

Top of Page

Functions Which Set the Properties of the Axes

- **Function FF.LBWF.SetAxesProperties(AxesLength, AxesThickness)**

In version 0.6 of the LBWF Library, the default length of the `axes` lines is 50 units, and the default width is 2 pixels. However, with the **FF.LBWF.SetAxesProperties()** function, the programmer can change the appearance of the `axes`.

- AxesLength - The length of the axes in whatever units the programmer is using.
- AxesThickness - The thickness of the axes, in pixels.

The `axes` will be drawn according to these new properties the next time a FF.LBWF.DrawAllObjects() function is called, provided that the `axes` are designated as shown instead of hidden.

Top of Page

- **Function FF.LBWF.ShowAxes()**

By default, the `axes` are not visible. This function ensures that the `axes` will be drawn when the FF.LBWF.DrawAllObjects() function is called. Function **FF.LBWF.ShowAxes()** requires no arguments. I like to call this function immediately after calling the Camera and ScreenCenter functions.

When the `axes` are drawn, the blue line represents the `x axis`, the red line represents the `y axis`, and the dark green line represents the `z axis`.

Top of Page

- **Function FF.LBWF.HideAxes()**

If the programmer does not want the `axes` to appear the next time the FF.LBWF.DrawAllObjects() function is called, she may use Function **FF.LBWF.HideAxes()**. This function requires no arguments.

Top of Page

Functions Which Create Objects

Version 0.6 of the LBWF Library provides functions for the creation of nine different objects. Below, I will provide the documentation for each one.

- **Function FF.LBWF.CreateBox(BoxName$, xdim, zdim, boxheight, BoxColor$)**

As the reader could guess, this function creates a *box*. Understand, however, that the data for the box is merely created in memory. The box will not be plotted in the `graphicbox` until a function is called which draws the object.

- BoxName$ - All LBWF objects must be assigned names. Names are handled as strings. The Library recommends that the programmer use object names without spaces.
- xdim - The dimension of the box in the x-direction
- zdim - The dimension of the box in the Z-direction
- boxheight - The height of the box.
- BoxColor$ - All objects must be given a color, passed to the function as a string. See the Liberty BASIC help files for proper ways to designate colors.

So, where are boxes, and other objects, plotted? The default location of a box is centered on the `y axis`, with the bottom of the box resting on the `X Z plane`. All objects are originally positioned similarly. Elsewhere, I will identify the function that allows the programmer to move any objects she creates.

Top of Page

- **Function FF.LBWF.CreateCylinder(CylName$, radius, numSides, cylheight, CylColor$)**

This function creates the data for a *cylinder*. The cylinder itself will not appear in the `graphicbox` until a function is called with draws the object.

- CylName$ - The name of the cylinder, sent to the function as a string.
- radius - The radius of the cylinder
- numSides - The number of sides that you want the cylinder to have. Not that in the example above, the cylinder has 20 sides.
- cylheight - The height of the cylinder.
- CylColor$ - The color of the cylinder, sent to the function as a string.

Top of Page

- **Function FF.LBWF.CreatePyramid(pyrName$, xdim, zdim, pyrheight, pyrColor$)**

This function creates the data for a *pyramid*. The base of the pyramid resides on the `X Z plane`, and the `axis` of the pyramid is on the global `Y axis`. The pyramid itself will not appear in the `graphicbox` until a function is called which draws the object.

- pyrName$ - The name of the pyramid input as a string. The Library recommends that the programmer use object names without spaces.
- xdim - The dimension of the pyramid in the x-direction
- zdim - The dimension of the pyramid in the Z-direction

- pyrheight - The height of the pyramid.
- pyrColor$ - The color of the pyramid, passed to the function as a string. See the Liberty BASIC help files for proper ways to designate colors.

Top of Page

- **Function FF.LBWF.CreateCone(ConeName$, radius, numSides, coneheight, ConeColor$)**

This function creates the data for a *cone*. The base of the cone will be located in the `X Z plane`, and the `axis` of the cone is on the global `Y axis`. The cone itself will not appear in the `graphicbox` until a function is called which draws the object.

- ConeName$ - The name of the cone, sent to the function as a string.
- radius - The radius of the cone.
- numSides - The number of sides that you want the cone to have.
- coneheight - The height of the cone.

>ConeColor$ - The color of the cone, sent to the function as a string.

Top of Page

- **Function FF.LBWF.CreatePolygon(PolyName$, radius, numSides, PolyColor$)**

This function creates the data for a *polygon*. When created, the polygon will reside in the `X Z plane`, and will be centered on the `Y axis`. The polygon itself will not appear in the `graphicbox` until a function is called which draws the object.

- PolyName$ - The name of the polygon, sent to the function as a string.

- radius - The radius of the polygon.
- numSides - The number of sides that you want the polygon to have. The numSides value should be an integer.
- PolyColor$ - The color of the cone, sent to the function as a string.

Top of Page

- **Function FF.LBWF.CreateGridObject(GridName$, NumUnitsXdirection, NumUnitsZdirection, UnitSize, GridColor$)**

This function creates a *grid object*. A grid object is a rectangular arrangment of intersecting lines. When created, a grid object resides in the `X Z plane`, and is centered on the `Y axis`. A grid object is very useful for showing "depth" in the graphicbox when objects are plotted.

The grid itself will not appear in the `graphicbox` until a function is called which draws the object.

- GridName$ - The name of the grid, sent to the function as a string.
- NumUnitsXdirection - The number of grid squares that the grid will have in the x-direction.
- NumUnitsZdirection - The number of grid squares that the grid will have in the x-direction.
- UnitSize - The size of one side of a grid square.
- GridColor$ - The color of the cone, sent to the function as a string.

Top of Page

- **Function FF.LBWF.CreateDome(DomeName$, radius, numSides, DomeColor$)**

This function creates a *dome*, which is the top half of a sphere. The base of the dome resides in the `X Z plane`, and the dome is centered on the `Y axis`. The dome itself will not appear in the `graphicbox` until a function is called which draws the object.

- DomeName$ - The name of the dome, sent to the function as a string.
- radius - The radius of the dome.
- numSides - The number of sides that you want the dome to have.
- DomeColor$ - The color of the dome, sent to the function as a string..

Top of Page

- **Function FF.LBWF.CreateLine(LineName$, x1, y1, z1, x2, y2, z2, LineColor$)**

This function creates a *line* in space. A series of lines can be created within a counted loop in order to make stars and other interesting geometrical shapes.

Newly created lines will not appear in the `graphicbox` until a function is called which draws the object.

- LineName$ - The name of the line, sent to the function as a string.
- x1, y1, z1 - The x-, y- and z-coordinates of the starting point of the line.
- x2, y2, z2 - The x-, y- and z-coordinates of the end point of the line.
- LineColor$ - The color of the line, sent to the function as a string..

Top of Page

- **Function FF.LBWF.CreateCyl2(Cyl2Name$, radiusBottum, radiusTop, numSides, cyl2height, Cyl2Color$)**

A *cyl2* is a cylinder like object whose bottom surface and top surface can have different radii. For example, a cyl2 with a bottom radius of 20 and a top radius of 10 would look like a cone with it's pointed top removed.

The cyl2 itself will not appear in the graphicbox until a function is called which draws the object.

- Cyl2Name$ - The name of the cyl2, sent to the function as a string.
- radiusBottum - The radius of the bottom surface of the cyl2.
- radiusTop - The radius of the top surface of the cyl2.
- numSides - The number of sides of the cyl2.
- cyl2height - The height of the cyl2.
- Cyl2Color$ - The color of the line, sent to the function as a string..

Top of Page

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

A *complex object* is a 3D wire frame object that must be created by assembling node and line data in an *external file*. The **CreateComplexObject** function allows the programmer to create objects beyond the standard 3D wire frame objects that come programmed into 1.0's functions.

- ObjectName$ - The name of the complex object, sent to the function as a string.
- FileName$ - The name of the complex object data file in which the node and line data for the complex object resides.
- ObjectColor$ - The color of the complex object, sent to the function as a string.

The format of the node and line data in the complex object datafile must abide by a few simple syntactical rules. This format is discussed in the article which introduces ***Wire 1.0***.

Top of Page

Functions Which Set the Properties of Objects

- **Function FF.LBWF.ObjectSetColor(ObjectName$, NewColor$)**

This function allows the programmer to change the *color* that was assigned to an object when that object was created with a create function.

- ObjectName$ - The name of the object whose color you want to change.
- NewColor$ - The new color of the object, sent to the function as a string..

Notice that the programmer can also *retrieve* the current color of an object by calling the FF.LBWF.RequestObjectColor$() function.

Top of Page

- **Function FF.LBWF.ObjectSetLineThickness(ObjectName$, LineThickness)**

When an object is created, the Library will give that object's lines a default thickness of 1 pixel. The **FF.LBWF.ObjectSetLineThickness**() function allows the programmer to change the line thickness of the object named.

- ObjectName$ - The name of the object whose line thickness you want to change.
- LineThickness - The new line thickness of the object.

The programmer can also *retrieve* an object's current line thickness by calling the FF.LBWF.RequestObjectLineThickness() function.

Top of Page

- **Function FF.LBWF.HideObject(ObjectName$)**

- ObjectName$ - The name of the object to be hidden. This must be the name used when the object was created.

When this function is called, the object is not *hidden* instantly. Rather, the function sets the visibility property of the object to "hidden". Then, the object will *not be drawn* the next time the FF.LBWF.DrawAllObjects() function is called.

Note also that the visibility property or *visible state* of any object can be retrieved at any time by calling the FF.LBWF.RequestObjectVisibleState() function.

Top of Page

- **Function FF.LBWF.ShowObject(ObjectName$)**

- ObjectName$ - The name of the object to be shown. This must be the name used when the object was created.

When this function is called, the object *is not shown* instantly. Rather, the function sets the visibility property of the object to "visible". Then, the object *will be drawn* the next time the FF.LBWF.DrawAllObjects() function is called.

Note also that the visibility property or *visible state* of any object can be retrieved at any time by calling the FF.LBWF.RequestObjectVisibleState() function.

Top of Page

- **Function FF.LBWF.ObjectAssignCustomType(ObjectName$, CustomType$)**

When objects are created, they are given a "type" during creation. The nine default types are box, cylinder, pyramid, cone, grid, polygon, dome, line and cyl2. These types become properties of the object, and the programmer can query the object type using the request function, FF.LBWF.RequestObjectType$().

But the programmer can also assign an object a *custom type*. A custom type is a name selected by the user to help organize objects into groups or collections. Types are useful because the ***LBWF Library*** allows the programmer to operate on objects based on their type. For instance, the programmer can hide, show, draw or translate objects based on their type.

- ObjectName$ - The name of the object to be assigned a custom type. This must be the name used when the object was created.
- CustomType$ - The custom type of the object, passed to the function as a string.

Here are a few things to note about custom types:

- An object's custom type will replace the object's default type. However, the programmer can reassign the object it's default type by using the **FF.LBWF.ObjectAssignCustomType()** function again.
- The ***LBWF Library*** will not prevent the programmer from assigning an object a custom type that is exactly the same as one of the default types. For instance, if the programmer has created a *cylinder* object, the library will not interfere if the programmer wants to give that object the type, *box*.
- Neither an object's default type or custom type negates the object's unique identity indicated by its name. In other words, the programmer can still operate on all objects individually inspite of whatever default types or custom types they have.

Top of Page

- **Function FF.LBWF.HideObjectsOfType(ObjectType$)**

This function will hide all objects of the type specified. Note that the objects will not be *instantly* hidden. Rather, the objects will merely have their visibility property set to "hidden". Then, the next time a FF.LBWF.DrawAllObjects() function is called, the objects will *not be drawn*.

- ObjectType$ - The type identifying the objects that the programmer wants to hide.

The object type passed as a parameter can either be a default type or a custom type.

Note also that the visibility property or *visible state* of any object can be retrieved at any time by calling the FF.LBWF.RequestObjectVisibleState() function.

Top of Page

- **Function FF.LBWF.ShowObjectsOfType(ObjectType$)**

This function will show all objects of the type specified. Note that the objects will not be *instantly* shown. Rather, the objects will merely have their visibility property set to "show". Then, the next time a FF.LBWF.DrawAllObjects() function is called, the objects will be drawn.

- ObjectType$ - The type identifying the objects that the programmer wants to show.

The object type passed as a parameter can either be a default type or a custom type.

Note also that the visibility property or *visible state* of any object can be retrieved at any time by calling the FF.LBWF.RequestObjectVisibleState() function.

Top of Page

- **Function FF.LBWF.HideAllObjects()**

This function sets the visibility property of all objects currently in inventory to "hidden". Then, the next time a FF.LBWF.DrawAllObjects() function is called, all objects in inventory will be hidden.

The **FF.LBWF.HideAllObjects()** function requires no arguments.

Note also that the visibility property or *visible state* of any object can be retrieved at any time by calling the FF.LBWF.RequestObjectVisibleState() function.

Top of Page

- **Function FF.LBWF.ShowAllObjects()**

This function sets the visibility property of all objects currently in inventory to "visible". Then, the next time a FF.LBWF.DrawAllObjects() function is called, all objects in inventory *will be drawn*.

The **FF.LBWF.ShowAllObjects()** function requires no arguments.

Note also that the visibility property or *visible state* of any object can be retrieved at any time by calling the FF.LBWF.RequestObjectVisibleState() function.

Top of Page

Functions Which Translate or Rotate Objects

- **Function FF.LBWF.TranslateObject(ObjectName$, transX, transY, transZ)**

To *translate* an object means to move the object *without rotating* it. LBWF provides a single translation function to move an object in any or all of the `x`, `y`, or `z axes`.

- ObjectName$ - The name of the object to be translated. This must be the name used when the object was created.
- transX, transY, transZ - The number of units to move the object in the x-, y- and z-directions.

Note that this function will always move the object relative to the object's current position. Note also that the object will not *appear* to move until a subsequent draw command is issued.

Top of Page

- **Function FF.LBWF.MoveObjectAbsolute(ObjectName$, pX, pY, pZ)**

This function translates the named object such that the object's geometric center now resides at the point (pX,pY,pZ). In other words, the function moves the object to an *absolute* point or destination, rather than moving the object relative to the object's current position.

- ObjectName$ - The name of the object to be moved. This must be the name used when the object was created.
- pX, pY, pZ - The x-, y- and z-coordinates of the destination point to where the object will be moved.

There might be occasions when the programmer wants to move the object to a specific X and Z location, but wants the y-coordinate of the object to remain the same. (That is, the programmer wants to move the object laterally, but doesn't want to "elevate" the object.) When that is the case, the programmer should request the object's geometric center using the FF.LBWF.RequestObjectGeometricCenter$() function. When the result is returned from the function, the programmer needs to parse the result for the y-coordinate of the object's geometric center. This value, then, is passed as pY in the **FF.LBWF.MoveObjectAbsolute()** function. That way, the object moves in the `x` and `z` directions, but its `y` coordinate does not change.

Top of Page

- **Function FF.LBWF.RotateObjectAboutY(ObjectName$, YRotationInDegrees)**

An object's `local y axis` runs through the geometric center of the object and is parallel to the `global y axis`. The **FF.LBWF.RotateObjectAboutY()** function will rotate the object about its `local y axis`.

- ObjectName$ - The name of the object to be moved. This must be the name used when the object was created.
- YRotationInDegrees - The angle, in degrees, through which the object should be rotated.

Objects also have `local x` and `z axes`. These `axes` always remain "attached" to the geometric center of the object, and will always be parallel to their corresponding global `axes` even after a rotation has occured. For instance, if the programmer rotates an object 45 degrees about the object's `local y axis`, the same object's `local x` and `z axes` do not rotate along with the object. All `local axes` stay parallel to the `global axes`.

The object specified will appear in its new rotated position the next time a FF.LBWF.DrawObject() or a FF.LBWF.DrawAllObjects() function is called.

Top of Page

- **Function FF.LBWF.RotateObjectAboutX(ObjectName$, XRotationInDegrees)**

This function rotates an object about the object's `local x axis`. For a discussion of `local` and `global axes` see the FF.LBWF.RotateObjectAboutY() function.

- ObjectName$ - The name of the object to be moved. This must be the name used when the object was created.
- XRotationInDegrees - The angle, in degrees, through which the object should be rotated.

The object specified will appear in its new rotated position the next time a FF.LBWF.DrawObject() or a FF.LBWF.DrawAllObjects() function is called.

Top of Page

- **Function FF.LBWF.RotateObjectAboutZ(ObjectName$, ZRotationInDegrees)**

This function rotates an object about the object's `local z axis`. For a discussion of `local` and `global axes` see the FF.LBWF.RotateObjectAboutY() function.

ObjectName$ - The name of the object to be moved. This must be the name used when the object was created.

ZRotationInDegrees - The angle, in degrees, through which the object should be rotated.

The object specified will appear in its new rotated position the next time a FF.LBWF.DrawObject() or a FF.LBWF.DrawAllObjects() function is called.

Top of Page

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

This function will translate all objects of the type specified. It will translate each object the distance specified (transX, transY, transZ) relative to the current position of each object.

- ObjectType$ - The type of the objects that the programmer wishes to translate. This can either be a default type or a custom type.
- transX - The distance that the collection will be translated in the x-direction.
- transY - The distance that the collection will be translated in the y-direction.
- transZ - The distance that the collection will be translated in the z-direction.

The objects will appear in their new position the next time a FF.LBWF.DrawObject(), FF.LBWF.DrawAllObjects(), or FF.LBWF.DrawObjectsOfType() function is called.

Top of Page

Functions Which Draw One or More Objects

- **Function FF.LBWF.DrawObject(ObjectName$)**

- ObjectName$ - The name of the object to be drawn. This must be the name used when the object was created.

The DrawObject function draws the object identified in the argument. When the function is called, the object will be drawn *regardless* of whether its visibility property was set to "hidden" by the FF.LBWF.HideObject(ObjectName$) function.

Top of Page

- **Function FF.LBWF.DrawAllObjects()**

This function requires no arguments. The function will draw all objects, except for those whose visibility property is set to "hidden" by the FF.LBWF.HideObject() function.

Top of Page

- **Function FF.LBWF.DrawObjectsOfType(ObjectType$)**

This function will draw all objects of the type specified. The objects will be drawn *regardless* of whether their visibility property has been set to "hidden".

- ObjectType$ - The type identifier of the object collection to be drawn. This can be either a default type or a custom type.

Top of Page

Functions Which Return Information About Objects or the LBWF System ("Request" Functions)

- **Function FF.LBWF.RequestLibraryResources$()**

In the ***LBWF Library***, node data, line data and object data are stored in arrays. Therefore, the amount of data that you can store is not unlimited. The **FF.LBWF.RequestLibraryResources$()** function provides information about the total capacity of the library before any resources are used.

Specifically, in version 0.6 of the ***Library*** the function returns the following string: "4000 4000 500". This means that the ***Library*** routines have the capacity to store data for 4000 nodes, 4000 lines and 500 objects before any objects are created. The programmer can extract the individual pieces of data from this return string by using Liberty BASIC's native **WORD$()** and **VAL()** functions.

The **FF.LBWF.RequestLibraryResources$()** function requires no arguments.

Top of Page

- **Function FF.LBWF.RequestRemainingResources$()**

This function is similar to FF.LBWF.RequestLibraryResources$() except that it returns a string containing the remaining resources while the programmer or user is in the process of creating objects. For instance, if the function returned the following string -- "3120 3350 420" -- that would mean that the programmer has enough storage space for 3120 nodes, 3350 lines, and 420 objects.

The **FF.LBWF.RequestRemainingResources$()** function requires no arguments.

Top of Page

- **Function FF.LBWF.RequestObjectGeometricCenter$(ObjectName$)**

This function will return the `x`, `y` and `z coordinates` of the geometric center of the object named. The result is returned as a string. In the string, the three numbers are separated by spaces, allowing the programmer to parse the return string to obtain individual coordinate numbers. For example, the function may return **"65.702 8.666715 106.815547"**. This would mean the center of the object named is at

```
x=65.702, y=8.666715, z=106.815547
```

- ObjectName$ - The name of the object of which geometric center coordinates are desired.

Note that this function was also mentioned in the description of the FF.LBWF.MoveObjectAbsolute() function.

Top of Page

- **Function FF.LBWF.RequestObjectColor$(ObjectName$)**

This function returns the *color* of the object identified as an argument.

- ObjectName$ - The name of the object of which the color is desired.

Remember that the color of an object is set when the object is created, and the object's color can be changed with the FF.LBWF.ObjectSetColor() function.

Top of Page

- **Function FF.LBWF.RequestObjectLineThickness(ObjectName$)**

This function returns the *line thickness* of the object identified as an argument.

- ObjectName$ - The name of the object of which the line thickness is desired.

Note also that the programmer can set the line thickness with the FF.LBWF.ObjectSetLineThickness() function.

Top of Page

- **Function FF.LBWF.RequestObjectType$(ObjectName$)**

This function returns the *type* of the object identified as an argument. The type can either be a *default* type or a *custom type*. Custom types are set with the FF.LBWF.ObjectAssignCustomType() function.

- ObjectName$ - The name of the object of which the type is desired.

Top of Page

- **Function FF.LBWF.RequestObjectExtents$(ObjectName$)**

This function essentially returns the `x`, `y`, and `z coordinates` of the "bounding box" of the object named in the function. These six values represent the minimum and maximum X, Y and Z values of the area that the object occupies in space. This six values are assembled together in a string which can be parsed by the programmer.

As an example, the function might return the value, **"25.2 35.7 86.5 125.9 -55.621 12.448".** The programmer would interpret these to mean that the object is bounded

```
in the x- dimension by xmin = 25.2 and xmax = 35.7
in the y- dimension by ymin = 86.5 and ymax = 125.9
and in the z- dimension by zmin = -55.621 and zmax=12.448
```

- ObjectName$ - The name of the object whose extents is desired.

Top of Page

- **Function FF.LBWF.RequestObjectVisibleState(ObjectName$)**

This function returns the *visibility property* of the object as set by any of the functions which allow hiding or showing of objects.

- ObjectName$ - The name of the object whose visibility state is desired.

The quantity returned by the function is an integer. If the visibility property of the object is set to "visible", then this function will return 1. Otherwise, if the visibility property of the object is set to "hidden", then this function will return 0.

Top of Page

- **Function FF.LBWF.RequestObjectNameFromXY$(ScreenX, ScreenY, pixelLimit)**

This function returns the name of that object which has a line which passes through (or near) the graphicbox pixel having coordinates `ScreenX` and `ScreenY`. This function requires an extensive explanation, to be provided after the identification of arguments below.

- ScreenX - The x-coordinate of the graphicbox pixel through which the object's line passes (or passes nearby).
- ScreenY - The y-coordinate of the graphicbox pixel through which the object's line passes (or passes nearby).
- pixelLimit - The allowable offset distance, in pixels, between the graphicbox point defined by (ScreenX, ScreenY) and the object's line, which will still register a "hit" on an object.

The purpose of this function is to give programmers and users a way to select a 3D object by clicking on one of the object's lines in the graphicbox in which the object appears. This function returns the name of the object which has a line passing through, or near, the `graphicbox` point at (`ScreenX`, `ScreenY`). Once the programmer is in possession of the object's name, then she can manipulate the object using any of ***Wire's*** other functions.

How does the function provide results?

If the programmer passes to the function a pair of coordinates, `ScreenX` and `ScreenY`, that is within the `pixelLimit` distance of any line of an object, then the function returns that object's name. Otherwise, if the graphicbox point sent to the function is more than `pixelLimit` distance from any line of any object, then the function returns the string, **"null00"**, meaning that no object has been selected. Because the string "null00" is used to communicate that no object is selected by the function, "null00" should be considered a **`reserved word`** by ***Wire*** programmers and users.

Programmers must understand that `ScreenX` and `ScreenY` are `graphicbox` coordinates, not space coordinates. Even though objects made with ***Wire*** are fundamentally defined as space objects, they are rendered into a Liberty BASIC `graphicbox`. Therefore, the vertices (or *nodes*) of a 3D object will have `graphicbox` coordinates in addition to spatial coordinates. For each and every 3D object, the ***Wire*** engine *knows fully* the graphicbox coordinates of that object's nodes.

How does the programmer select screen coordinates to pass to the function?

Programmers should remember that Liberty BASIC always knows the location of the pointer within a Liberty BASIC graphicbox. Liberty BASIC uses the variables `MouseX` and `MouseY` to hold the `graphicbox` coordinates of the pointer. The values stored in `MouseX` and `MouseY` change instantly as the pointer is moved around within the `graphicbox`. As the programmer or user moves the pointer over a line of a ***Wire*** object, the programmer can capture the coordinates of the pointer by clicking on the line, and reading the coordinates by trapping a mouse event, such as **"when leftButtonUp"**. (See the discussion of `graphicbox` commands in Liberty BASIC's help file for more information.) Once these pointer coordinates are captured, the programmer can send them to the

FF.LBWF.RequestObjectNameFromXY$() function.

What is the purpose of the pixelLimit argument?

Wire understands that it would be inconvenient and unproductive to require that a programmer or user always click exactly on top of a line if she wants to select an object. The `pixelLimit` argument is provided to give the programmer a range distance adjacent to a line, and still have the function indicate a "hit" on an object. That is, if the `pixelLimit` is set to three, then the user only has to designate a `graphicbox` point, or click on a `graphicbox` point, that is only three pixels from the line of an object in order to register a "hit" on that object.

What else should the programmer know about FF.LBWF.RequestObjectNameFromXY$()?

Because lines of different objects will often overlap each other, there can be many graphicbox pixels that are within pixelLimit of more than one object. The **FF.LBWF.RequestObjectNameFromXY$()** will always return the name of the first object that is within `pixelLimit` of the `graphicbox` coordinates sent to the function. *The order of objects is determined by the order in which they are created by the programmer or user*.

Because multiple objects can appear within `pixelLimit` of the same `graphicbox` point, the programmer and user are advised to click on areas of lines that tend *not to be too close* to the lines of other objects. This will help the user avoid selecting the wrong object.

Top of Page

Miscellaneous Functions

- **Function FF.LBWF.ZeroAllData()**

This function erases all node, line and object data, and essentially resets the entire ***LBWF*** system back to its state at startup. After calling this function, the number of nodes is zero, the number of lines is zero, and the number of objects is zero.

This function requires no arguments.

Top of Page

- **Function FF.LBWF.PauseMilliseconds(DelayMS)**

This function will cause a *delay* in program execution by continually querying the system clock until the designated number of milliseconds has passed. Pause functions are useful during animation to prevent the screen from being redrawn too rapidly.

- DelayMS - The time length of the pause, given in milliseconds.

Liberty BASIC user Welopez pointed out that this particular pause function seems to consume all system resources. For that reason, an alternate pause function is provided which uses Liberty BASIC's native **TIMER** command. For the time being, *both* pause functions will remain in the ***LBWF Library***.

Top of Page

- **Function FF.LBWF.PauseUsingTimer(DelayMS)**

This function creates a *pause* in system execution by use of Liberty BASIC's native **TIMER** command. It may be used as an alternative to the FF.LBWF.PauseMilliseconds(DelayMS) function discussed above.

- DelayMS - The time length of the pause, given in milliseconds.

Top of Page

- **Function FF.LBWF.ATAN2(x, y)**

This function returns the *arc* whose *tangent* is `(y/x)`. The result is returned in radians. The result will always be in the range of `0 to (2 x pi)`.

- x - The x-length of the right triangle whose angle is desired.

- y - The y-length of the right triangle whose angle is desired.

The programmer may never have a reason to use the **FF.LBWF.ATAN2()** function. However, ***LBWF Library*** itself requires **ATAN2** in order to successfully perform the object rotations called for by FF.LBWF.RotateObjectAboutY(), FF.LBWF.RotateObjectAboutX(), and FF.LBWF.RotateObjectAboutZ().

Top of Page

- **Function FF.LBWF.LBWFVersion$()**

This function will return the *current version* of the ***LBWF Library***. It requires no arguments. As of the publication of this document, the current version of the ***Library*** is 0.6.

Top of Page

- **Function FF.LBWF.About()**

This function displays a Liberty BASIC `NOTICE` box which contains the ***LBWF*** version number and an attribution of ***library authorship*** to ***Tom Nally***. The function requires no arguments.

Top of Page

Tom Nally
Steelweaver52@aol.com

Note: This linked article accompanies Chapter 4: Wire 1.0 Released (Making Complex Objects With Wire)*, which originally appeared in the Liberty BASIC Newsletter, Issue #137. It is reprinted here with the permission of the author. -*
JanetTerra