The Liberty BASIC Wire Frame Library

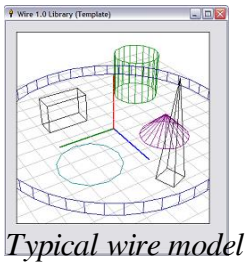
Tomas P Nally -
steelweaver52

Chapter 5: Using Wire (Focusing on the FF.LBWF.RequestObjectNameFromXY$() Function)

This fifth chapter of the Liberty BASIC Wire Frame Library series originally appeared in the Liberty BASIC Newsletter, Issue #139. 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.



Typical wire model objects

What is Wire?

Wire is the friendly name for the ***Liberty BASIC Wire Frame Library***. This is a library of Liberty BASIC functions which allow the programmer to create three dimensional images of wire model objects. A "wire model object" is a 3-D shape whose edges are visible but whose surfaces are transparent. For example, a wire model image of a cube would look like a cube made from the wire of a coat hanger. A few typical wire model objects can be seen in the figure above.

This is the fifth article about the ***Wire*** library. If you are interested in the previous four articles, you may read them at the following links:

- 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)

The complete documentation of the ***Wire*** function library can be found in this companion document. If you are already sufficiently familiar with ***Wire*** and would like a starter program from which to build an application,

LBWF10_template.zip

- Details
- Download
- 13 KB

for the Liberty BASIC source file LBWF10_template.bas.

Before we talk about the **FF.LBWF.RequestObjectNameFromXY$()** function, it's important that we discuss how objects are addressed in ***Wire***.

Addressing Wire Model Objects by Name

Three dimensional objects created using ***Wire's*** functions have a number of properties. Among an object's properties are *name*, *type*, *color*, and *line thickness*. While an object's *type*, *color* and *line thickness* can be changed by invoking ***Wire's*** functions, the object's *name* will remain unchanged from the instant the object is created. (That's the way ***Wire*** operates at this time. No predictions are being made for the future operability of Wire.)

The constancy of an object's name is important because the programmer will manipulate the object by referring to it by name. For example, say that I have an object called "cone1". If I want to rotate this object by 62 degrees about its Y-axis, I do so by making the following call to a Wire function:

```
AAA = FF.LBWF.RotateObjectAboutY("cone1", 62)
```

Note the object's name, "cone1" in the arguments of the function. Addressing objects by name is logical and seems to work very well.

However, if you are programming a ***Wire*** application, you would not necessarily want to give your users the burden of remembering the name of every wire model object created during a session. After all, ***Version 1.0*** of ***Wire*** allows the creation of 500 objects! It would be much more helpful if your users could select an object, say, by clicking on it with the mouse rather than by typing the object's name.

The purpose of **FF.LBWF.RequestObjectNameFromXY$()** is to make object *selection by mouse-click* possible. The function doesn't remove the need for object names. Rather, it gives the programmer access to an object's name via a mouse-click.

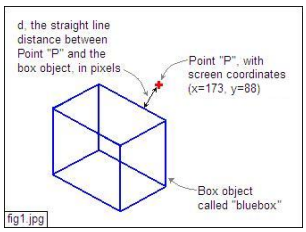
Using the FF.LBWF.RequestObjectNameFromXY$( ) Function

The function identified above, let's call it ***"RON"*** for ***"Request Object Name"***, takes three arguments. ***RON's*** first two arguments, identified as `ScreenX` and `ScreenY`, consist of the `x` and `y` `coordinates` of a single pixel within the Liberty BASIC `graphicbox` where the 3-D objects are being drawn. The third argument, called `pixelLimit`, is an "allowable offset distance" in pixels from the screen point identified by the coordinates (`ScreenX`, `ScreenY`). When used in your source code, the function will look like this:

```
AAA$ = FF.LBWF.RequestObjectNameFromXY$(ScreenX, ScreenY, pixelLimit)
```

This function will return the name of that object which has a line which passes near (or through) the

graphicbox pixel having coordinates `ScreenX` and `ScreenY`. To understand this better, see the figure below.



This figure represents a Liberty BASIC `graphicbox` control. It contains a 3D wire model object named "bluebox". In addition, a single pixel is identified in the `graphicbox` with a red cross. The screen (or graphicbox) coordinates of that pixel are

```
x = 173 and y = 88
```

We are going to use this information in the ***RON*** function, but first we must designate a value for `pixelLimit`. We'll do that as we set the values for all three arguments prior to calling the function:

Using the RON Function With Liberty BASIC Mouse Events

The most efficient way to put ***RON*** into service is to use it in conjunction with Liberty BASIC's handling of `mouse events`. For example, the programmer can set up a routine to process all `left mouse button clicks`.

Recall that the location of the mouse pointer is held in the two variables, `MouseX` and `MouseY`. The contents of these two variables are updated continuously as the mouse moves within the `graphicbox`. When the user clicks the `left mouse button`, the programmer can capture the location of the pointer at the time of the `left-click` by reading the variables `MouseX` and `MouseY`. Then, the values of `MouseX` and `MouseY` can be stored in variables which are subsequently used as the `ScreenX` and `ScreenY` arguments in the function call to ***RON***. Here is a short code snippet which shows how that can be done:

```
'Establish the code block [MouseLeftButtonUp] as the routine which
'processes the leftButtonUp event...

print #main.wfscene, "setfocus; when leftButtonUp [MouseLeftButtonUp]"

Wait

[MouseLeftButtonUp]
```

```
ScreenX = MouseX  'Capture the x-coordinate of the pointer
                  'at the time of the mouse event.

ScreenY = MouseY  'Capture the y-coordinate of the pointer
                  'at the time of the mouse event.

pixelLimit = 3    'Set the pixelLimit to 3...

'...then call the RON function...
ObjectName$ = FF.LBWF.RequestObjectNameFromXY$(ScreenX, ScreenY, pixel
Limit)

Wait
```

When this code section executes, the ***RON*** function will find an object located *within 3 pixels* of the pixel that was clicked, and place that object's name in the string variable, **ObjectName$**. In the event that no object was within 3 pixels, the ***RON*** function will return **"null00"**.

Will RON Always Identify the "Correct" Object?

Say that the programmer issues a call to the ***RON*** function, but there happens to be two objects that are within `pixelLimit` of `ScreenX` and `ScreenY`. Which object name does ***RON*** return?

Before that is answered, understand that the ***Wire*** system records object information in *the order in which the objects are created*. If two or more objects happen to be within `pixelLimit` of `ScreenX` and `ScreenY`, ***RON*** will return the name of the *first* object, based on the order in which the objects were created, which has a line within `pixelLimit` of `ScreenX` and `ScreenY`.

Because of ***RON's*** potential to return the name of the "wrong" object when two objects are near to `ScreenX` and `ScreenY`, the programmer should consider providing the user with feedback on the user's selection. For example, if the user clicks on an object, the programmer can change the color of that object to light gray as a way to tell the user, "this is the object that ***RON*** has identified". If it happens to be the "wrong" object, then the programmer can allow the user to cancel out of the current operation, and select a different part of the object which will be less confusing to ***RON***.

Demo Program: Point2Object3.bas

Table of Contents

Download

Point2Object3.zip

- Details
- Download
- 14 KB

for the demo program, Point2Object3.bas, which puts the ***RON*** function to the test. The program draws six pyramids on top of a grid. The user is invited to click on any of the pyramids, and watch them translate, rotate, change color, or change line thickness. As an object is selected, its name appears in a textbox at the bottom of the application's window.

The user can also click a button to randomly change the camera location, at which time she can click on the pyramids again. This demonstrates that ***Wire*** always knows the screen location of every object, and is able to use that knowledge in the ***RON*** function.

Tom Nally
Steelweaver52@aol.com

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