dited by: ZiffDuck

For any of you who have read "Mac Programming for Dummies" by Dan Parks Sydow. You will recognize a lot of this text. Why did I compile information from this book? Well, it is one of the best books for learning how to program on the Mac, and when I was learning (and I still am =) I followed every step, reread things and well pretty much read all the fine print.

In case you are wondering why this article is first and not last or the middle is a good question to ask. Let me explain that topic to you. You see the toolbox is not an application, or lines of code. The toolbox is one of the very special things about the Macintosh computer; Apple has built in thousands and thousands of Mini programs. You can access these mini programs by just typing one line of code. Now these mini programs make your life easier thanks to those brilliant people at Apple. They make mini tasks easy because there already made for you. Now imagine your Mac is the toolbox, all the mini programs are the tools, and together make your Mac toolbox.

Calls to the toolbox make it easier to write an application when you are working with windows, fonts, and menus you name it! Apple has made these all for you so you don’t have to write an extra 50 lines of code. Were you aware that there are more then five thousand Toolbox calls? Were you also aware of the Toolbox when you bought your Macintosh? Probably not, but some people are aware, that is why a lot of people like to program on the Macintosh. By now you may or may not have figured out that there are no such things called "Mini programs" they are in fact called "Functions"

GetNewWindow, MoveTo, DrawString. And any other toolbox call is a function.

Function Parameters

Parameters (pronounced Pah-Ram-ah-ter) or Arguments (take your pick) provide the toolbox with information that it needs to supply to the toolbox so it can run properly. Some toolbox functions need more than one parameter, and some functions don't need any. All parameters are contained in parenthesis

Initialization Functions

There are eight Initialization functions you must include when writing a Macintosh program. They go after you declare you variables in your source code. Here I will go over these eight topics.

The Functions

```
InitGraf( &qd.thePort );
InitFonts();
InitWindows();
InitMenus();
TEInit();
InitDialogs( OL );
FlushEvents( EveryEvent, 0 );
InitCursor();
```

Displaying a Window

Just about everything in a Mac program is contained in a window, or a menu. You can guess why those calls are one of the most important Toolbox calls in Macintosh programming. The code below is what it will usually look like when you make a call to this function

```
WindowPtr

theWindow;
theWindow = GetNewWindow( 128, OL, (WindowPtr)-1L );
```

Activating a Window

Once you have made this beautiful, but plain window you want to do something with it. You don't want a plain window just sitting in your program, right? Well with this next section it will explain how to put text and pictures into a window.

```
SetPort( theWindow );
```

`SetPort` helps the Mac realize what window you are going to make it draw to. To help `SetPort` enable its function you pass it along to `WindowPtr` of the window to make the window current. You can use the `WindowPtr` variable that was returned by the call to `GetNewWindow`.

Displaying a MenuBar

In this call there are going to be one variable and three toolbox calls for you to display a menubar

```
Handle menuBarHandle;

menuBarHandle = GetNewMBar( 128 );
SetMenuBar( menuBarHandle );
DrawMenuBar();
```

This part of the code needs a little explaining... `GetNewMBar` is the call that handles the MBAR resources ID. Also `GetNewMBar` gives the program a `handle`. Now, SetMenuBar needs this `handle` in order to group all the information about the menubar and all the menus contained in it. `DrawMenuBar` is what makes the menu bar visible on your screen.

Capturing Events

You know how old men just kind of sit on their front porches widdlin' their sticks? Well `WaitNextEvent` is just like those old men. It just kind of sits there. It looks for an event to happen. When it notices this event is stores the information about the variable in `EventRecord`

```
EventRecord

theEvent;
WaitNextEvent( everyEvent, &theEvent, OL, OL );
```

an event is a variety of things that happens when the user does something, clicks the menu, clicks a window or even when the user clicks the mouse button.

Locating a Mouse Click

Usually, a user can click anywhere on the screen. But, it's up to your program that you code to figure out where the cursor is when the user makes a click. For your program to do this you must use FindWindow

```
EventRecord

TheEvent;
WindowPtr


 whichWindow;
short
```

```
thePart

thePart  =  FindWindow( theEvent.where, &whichWindow );
```

The variable of EventRecord is how `FindWindow` operates, so you must call what happens after `WaitNextEvent`. In overall, `FindWindow` gives your program the location of the screen, window or where ever the mouse was clicked. If the mouse happens to be clicked in a window, `FindWindow` uses the variable `whichWindow` to provide your program with a `WindowPtr` to the window.

Working With Windows

When/if FindWindow tells your homemade computer application that a user clicked somewhere in the drag bar of a window, you should respond by using the DragWindow. This function watches the mouse movements as the user moves the little mousie.

```
WindowPtr




whichWindow;
EventRecord
theEvent;

DragWindow( whichWindow, theEvent.where,









&qd.screenBits.bounds );
```

The first function of `DragWindow` tells the toolbox what window to drag. The second parameter is where the mouse was first clicked. The last parameter lets the toolbox know that the window can be dragged anywhere on the screen.

If FindWindow tells your application that the user clicked in the close box, you should call DisposeWindow to well, dispose the window.

```
WindowPtr


whichWindow;

DisposeWindow( whichWindow );
```

Managing Menus

Whenever a user clicks in the menu bar, a call to `MenuSelect` will take care of making menus drop when the mouse is dragged over them. When the mouse is clicked, a call to `MenuSelect` just happens to know what item and menu is selected. It stores all its valuable information in `menuAndItem`.

```
long

menuAndItem;
MenuAndItem = MenuSelect( theEvent.where );
```

To extract the number of the menu and menu item call `HiWord` and `LoWord` from `menuAndItem`

```
short

theMenu;
short

theMenuItem;
theMenu  = HiWord( menuAndItem );
theMenuItem = LoWord( menuAndItem );
```

When you call to MenuSelect it selects a menu in the menubar. After all this, call to HiliteMenu to return the menu name to its original place.

HiLiteMenu (0);

Drawing Text

On the Macintosh text is drawn not written. The toolbox function `DrawString` draws a word or line of text to a window. Enclose the written text in double quotes and precede the first word with the backslash key "\" and the letter p.

```
DrawString( "\pThis is where you put your text :-)" );
```

to select where in the window the text is drawn call the function MoveTo, then draw string

```
MoveTo( 20, 50 );
DrawString( "\pThis is where you put your text :-)" );
```

Drawing Shapes

Did you know you can also use the toolbox to make a rectangle, or fill it with a color or pattern? Well now you do! To do this you must call `FrameRect` to obviously draw a rectangle. Or you can call `FillRect` to fill the rectangle with a pattern. But you must tell the toolbox where the rectangle will go and how big or small it will be.

```
Rect
```

```
theRect;

Set( &theRect, 0, 0, 400, 280 );
```

You must be sure not mix the parameters of `SetRect`. You can follow them in this order

```
SetRect( &theRect, left, top, right, bottom );
```

after the size and location have been set, call `FrameRect` to draw a black line around it.

```
Framerect( &theRect );
```

If you want to fill the rectangle. Call `FillRect`. The second parameter tells the Toolbox what pattern to be filled with. here is how to fill it with a gray pattern

```
FillRect( &theRect, &qd.Gray );
```

other colors included are black, white, gray, ltgray, dkgray (always put &qd before the colors!)

Well people, that concludes my article in the Macintosh toolbox, if you have any comments, questions or ideas please email me or any one else on the team. Once again thanks! -Logik