WinFormsX

Windows Form library for FreeBASIC

Quick Start

The design philosophy of WinFormsX is such that the programmer should be able to quickly design and code an application with as little amount of direct exposure to the underlying Windows API as possible. This Quick Start document explains the basics that you should know in order to achieve that goal.

You can design WinFormsX applications using just one single huge code file if you so desired, but such setups quickly become confusing and the logical structure of your program becomes lost. The following is the suggested way to setup your application code for the vast majority of WinFormsX programs. Small, simple, utility or example code programs could probably get away with a simplified version of this approach but for the vast majority of programs this should be the approach you take.

For this Quick Start, we will work with files, forms and controls, related to a hypothetical password storage program called *CryptoX*. We also assume that you are using the WinFBE Editor.

Main Code File

Every application will have a main code file. This is the file that you compile to an EXE and subsequently execute. You would normally name this file to the same name as your application. For our application this file will be named *CryptoX.bas*. Notice that this file has a *.bas* file extension.

CryptoX.bas

```
''
''  CRYPTOX
''  Password Storage application.
''

' Use the following Include line if the library has already been copied
' to the compiler's \inc folder.
#Include once "WinFormsX\WinFormsX.bi"

' When compiling, be sure to include the file "resource.rc". In WinFBE, this can be done
' via the Compiler Setup / Additional Compiler Settings, or using the #RESOURCE code
' directive as shown below.
'#RESOURCE "resource.rc"

Include once "frmMain.bi"
' As you can imagine, the application would have many more forms. Each additional form would
' place their *.bi here.

Include once "frmMain.inc"

' Each additional form would place their *.inc here.
```

```
''
''  Main code for the application.
''

' You can place Dim Shared variables here or whatever else you need to do prior
' to the application starting. An even better design choice would be to place
' those types of variables into a separate code file and #Include it here.

' You start the application by passing the variable that holds the main startup form to the
' Run method of the Application class.
'
Application.Run(frmMain)
```

Separate Definition and Code files for each Form

Every Form in the application is separated into two parts: (1) The form's definition, and (2) The form's code. This may appear at first glance to be overkill and unnecessary but I assure you that as your application grows and there becomes an interdependence amongst various forms then you will be very happy that the separation exists.

For our CryptoX application we have a main form that we will name *frmMain*. This form is split into two physical disk files called, *frmMain.bi* and *frmMain.inc*. For the sake of simplicity these files are displayed below and contain a minimum of controls in order not to overly complicate matters. In this case, frmMain merely displays three simple buttons that the user can press; *OK*, *Cancel*, and *Setup* buttons.

Look through each of these two files to get a feel for the layout and then we'll revisit the major points of these files for further explanations.

frmMain.bi

```
''
''  CRYPTOX - MAIN FORM DECLARATIONS FILE
''

declare function frmMain_Click( byref sender as wfxForm, byref e as EventArgs ) as LRESULT
declare function frmMain_cmdOK_Click( byref sender as wfxButton, byref e as EventArgs ) as
LRESULT
declare function frmMain_cmdCancel_Click( byref sender as wfxButton, byref e as EventArgs ) as
LRESULT
declare function frmMain_cmdSetup_Click( byref sender as wfxButton, byref e as EventArgs ) as
LRESULT

''
''  Define the form and the child buttons
''
TYPE TFORMMAIN extends wfxForm
   private:


   public:
      ' Controls
      cmdOK as wfxButton
      cmdCancel as wfxButton
      cmdSetup as wfxButton

      declare constructor
END TYPE

''
''  Use the Constructor to define the properties and assign methods
''  to the form and child controls.
''
constructor TFORMMAIN
   ' Set the properties of the form
   with this
      .Size          = 600, 400
      .StartPosition = FormStartPosition.CenterScreen
      .Text          = "CryptoX - Password Manager"
      .Name          = "frmMain"
      .Background    = Colors.SystemButtonFace
      .OnClick       = @frmMain_Click
   end with
   ' Add the form to the global application collection
   Application.Forms.Add(ControlType.Form, @this)

   with this.cmdOK
      .Parent = @this
      .Name = "cmdOK"
      .Text = "OK"
      .SetBounds(10, 25, 70, 30)
      .Selected      = true   ' set initial focus to this control
      .TextAlign     = ButtonAlignment.MiddleCenter
      .OnClick       = @frmMain_cmdOK_Click
   end with
   this.Controls.add(controltype.Button, @this.cmdOK)

   with this.cmdCancel
      .Parent = @this
      .Name = "cmdCancel"
      .Text = "Cancel"
      .SetBounds(90, 25, 70, 30)
      .OnClick = @frmMain_cmdCancel_Click
   end with
   this.Controls.add(controltype.Button, @this.cmdCancel)

   with this.cmdSetup
      .Parent = @this
      .Name = "cmdSetup"
      .Text = "Setup"
      .SetBounds(170, 25, 70, 30)
      .OnClick = @frmMain_cmdSetup_Click
   end with

   this.Controls.add(controltype.Button, @this.cmdSetup)
```

```
END CONSTRUCTOR

''  Define a global variable that will allow access to the form and controls
Dim Shared frmMain as TFORMMAIN
```

frmMain.inc

```
''  CRYPTOX - MAIN FORM INCLUDE FILE
''
''
#Include once "frmMain.bi"

''
''  Define the various methods that the application will respond to.
''  User code will go into these methods in order to make the application work.
''
function frmMain_Click( byref sender as wfxForm, byref e as EventArgs ) as LRESULT
   ' Change the text for the button
   frmMain.cmdOK.Text = "New Text"
   function = 0
end function

function frmMain_cmdOK_Click( byref sender as wfxButton, byref e as EventArgs ) as LRESULT
   ' Your code to respond to the OK command button would go here.
   function = 0
end function

function frmMain_cmdCancel_Click( byref sender as wfxButton, byref e as EventArgs ) as LRESULT
   ' Your code to respond to the Cancel command button would go here.
   function = 0
end function

function frmMain_cmdSetup_Click( byref sender as wfxButton, byref e as EventArgs ) as LRESULT
   ' Your code to respond to the Setup command button would go here.
   function = 0
end function
```

That was a lot to understand!

Yes, for sure, that certainly looks like a lot of code just to display a form and some buttons but I assure you that it is actually very small compared to the work that goes on behind the scenes with WinFormsX classes, WinFBX library functions, and calls to the underlying Win32 API.

In practice, the only file that you will have to regularly interact with will be the form's .inc file. The definition file will normally be automatically generated by a visual designer such as the one built into the WinFBE Editor. If that's the case, then all you need to do is to write code that responds to the various events that occur during the running of your program, for example, the clicking of the *OK* button. That's pretty simple.

Let's look parts of the code that require a deeper understanding.

Event Handlers

```
declare function frmMain_Click( byref sender as wfxForm, byref e as EventArgs ) as LRESULT
```

This is a declaration for an *Event Handler*. Event handlers are functions that WinFormsX calls in response to something occurring within the operating system or in response to an action by the user. For example, if the user clicks on a button or moves their mouse cursor over an area of the form or a control then an *event* is triggered. In Windows API language this is actually a message or notification generated by the operating system that then flows through the application's message pump eventually finding its way to the procedure handler for the application. If you have ever seen Win32 API style programs then you will recognize functions containing large SELECT CASE statements having such things as WM_COMMAND, WM_NOTIFY, WM_PAINT, WM_SIZE, etc. WinFormsX hides all of the complexity of message pumps and that SELECT CASE from you by instead redirecting the message to an Event Handler that you define yourself.

Every Event Handler has the exact same format and is comprised of two parameters that must be defined as BYREF.

frmMain_Click: The portion before the underscore is the form name and the portion afterwards is the name of the event being responded to. In this case, the function is in response to someone clicking on an area of the form. The first parameter, *sender*, is a variable representing the form where the click occurred (eg. frmMain and is of the wfxForm class). You can interact with that variable and manipulate properties of the form:

sender.Text = "My new caption for the form's title bar!"

This is actually equivalent to using the form's explicitly defined shared variable and both are acceptable.

frmMain.Text = "My new caption for the form's title bar!"

When dealing with controls on a form, the syntax is only slightly different and pretty self-explanatory.

```
declare function frmMain_cmdOK_Click( byref sender as wfxButton, byref e as EventArgs ) as LRESULT
```

Of note is that *sender* is now of type *wfxButton* rather than *wfxForm*. This is because the function is in response to the user clicking on the OK button rather than the form itself.

In the *frmMain.inc* file you will find the Event Handler function and this is the spot where you would write the actual code to respond to the Click event.

```
function frmMain_Click( byref sender as wfxForm, byref e as EventArgs ) as LRESULT
   ' Change the text for the button
   frmMain.cmdOK.Text = "New Text"
   function = 0
end function
```

The second parameter, *EventArgs*, is short for Event Arguments and is a variable that contains additional information related to the event being handled. For example, when handling mouse movement the EventArgs variable will contain the client coordinates of the mouse cursor and which (if any) mouse buttons are pressed. Likewise, responding to a KeyPress event the EventArgs variable would contain the character that was pressed and whether any Alt, Shift, or Ctrl keys are pressed. Get to know EventArgs because it is extremely convenient.

The Form TYPE structure

Every form is represented by a standard TYPE structure. You can code this by hand but realistically it will be generated by the visual designer for convenience.

```
TYPE TFORMMAIN extends wfxForm
   private:

   public:
      ' Controls
      cmdOK as wfxButton
      cmdCancel as wfxButton
      cmdSetup as wfxButton

      declare constructor
END TYPE

Dim Shared frmMain as TFORMMAIN
```

Take notice of the *extends wfxForm* portion. This means that the form you are creating is derived from a standard form class that is built into WinFormsX called *wfxForm*. That class contains properties and event handlers that are specific to forms (much like wfxButton and wfxTextBox are specific to buttons and textboxes). As an aside, wfxForm is derived from an even lower level class called *wfxControl* but that is not important for our discussion here.

The TYPE further defines what controls exist on the form. In this case we have three buttons, all derived from the *wfxButton* class.

Following the TYPE we have a variable called *frmMain* that is declared as SHARED so that it can be accessed from anywhere from within the application. This allows you to do things like this:

```
frmMain.cmdCancel.Text = "Cancel"
```

The Form TYPE's Constructor

The Constructor for the TYPE is where you define the default properties of the form and any controls. Normally, the visual designer will output these values for the sake of convenience.

```
   with this
      .Size          = 600, 400
      .StartPosition = FormStartPosition.CenterScreen
      .Text          = "CryptoX - Password Manager"
      .Name          = "frmMain"
      .Background    = Colors.SystemButtonFace
      .OnClick       = @frmMain_Click
   end with
   ' Add the form to the global application collection
   Application.Forms.Add(ControlType.Form, @this)
```

It is important that the *Application.Forms.Add* is called after the form's properties are set otherwise the application will not know that it exists and your application will not function correctly.

```
    with this.cmdOK
       .Parent = @this
       .Name = "cmdOK"
       .Text = "OK"
       .SetBounds(10, 25, 70, 30)
       .Selected      = true    ' set initial focus to this control
       .TextAlign     = ButtonAlignment.MiddleCenter
       .OnClick       = @frmMain_cmdOK_Click
    end with
    this.Controls.Add(controltype.Button, @this.cmdOK)
```

Once you define properties for a control it is very important to add that control to the form's collection of control. You do this via *this.Controls.Add* code as seen in the example above. It is not sufficient to simply have defined the control variable within the form's TYPE - you *must* also add it to the form's control collection.

Assigning Events to Event Handlers

In the TYPE constructor you will see code like the following:

```
   .OnClick = @frmMain_cmdOK_Click
```

This is the way that you "wire" an Event Handler to the form or control itself. It is the way that you physically tell WinFormsX to call a specific function for a specific event. In the case of the above code, we are telling WinFormsX that when the "Click" event occurs for the OK button, call the function *frmMain_cmdOK_Click*. This is a major underpinning of the WinFormsX library that creates a degree of logical separation of code that makes code maintenance much easier of the long run.

There are many different events that can be responded to and it greatly depends on the form or control that you are interested in. For example, forms can have specific form events such as OnFormClosing and OnFormClosed in addition to common events such as OnMouseMove which is available to all controls. Refer to the documentation specific to the control you are interested in to learn about what events exist for it.

Conclusion

As you can see, WinFormsX is pretty straightforward to use once you learn the few fundamental basics of how it operates. The best way to learn is by doing. Create a small application and experiment. Read the documentation and ask questions on the support forum http://www.planetsquires.com