Introduction

Welcome to the IO Extension! I have spent hundreds of hours writing this library and believe it offers some very useful and interesting features that will help you, the programmer, to use Small Basic beyond its limited scope and develop rich, robust applications.

Getting started

To get started using the IO Extension, let's install the extension. Follow all the directions in the Prerequisites folder then copy `IOExtension.DLL` and `IOExtension.XML` into the "C:\Program Files (x86)\Microsoft\Small Basic\lib" folder (or "C:\Program Files\Microsoft\Small Basic" if you are on a 32-bit system) (also, replace "C:\" with the drive letter of your main hard disk if "C" is not the correct letter). If administrative permissions are necessary, type in the appropriate credentials and continue with the copy operation.

Once the copy is done, open Microsoft Small Basic. Verify that the method group "IOForms" is present by typing it into the code window and ensuring it appears in the IntelliSense wheel. If the method group is not present, close Small Basic, re-copy the files, then open Small Basic again. If the problem persists, please email me at gungan37@gmail.com or post on the Small Basic forums for assistance.

Now, type in the following code in the code editor:

```
IOForms.Setup()
IOForms.InitializeForm()
```

You should now see a blank window appear. Congratulations, you have successfully installed the IO Extension and are ready to make some cool programs! Let's add some things to the window. Now, modify the code to look like this:

```
IOForms.Setup()
IOForms.AddButton("Button1", "Hello, World!", "", 10, 10, 100,
50)
IOForms.InitializeForm()
```

Now, a window with a button appear. Let's take a look at what each parameter does:

1) `"Button1"`: this is the internal name of the control. This is the name you will use when programmatically addressing the control
2) `"Hello, World!"`: this is the text to display on the button
3) `""`: this is the path of the image to display on the button. Since we are only displaying text, we left this blank
4) `10`: this is the distance, in pixels, from the top of the window to the top of the control
5) `10`: this is the distance, in pixels, from the left side of the control to the left side of the control

6) `100`: this is the width of the control in pixels
7) `50`: this is the height of the control, in pixels

Don't let all these parameters scare you! They add a lot of functionality that you will (if you don't already) really want. Feel free to experiment with these parameters and see what you can do.

Now, we will add an event when the button is clicked. Modify the code in your code editor to look like this:

```
IOForms.Setup()
IOForms.AddButton("Button1", "Hello, World!", "", 10, 10, 100, 50)
IOForms.OnButtonClick=ButtonClick
IOForms.InitializeForm()

Sub ButtonClick
  If IOForms.LastButtonClicked="Button1" Then
    IOForms.ShowMessage("Button1 was clicked", "Event fired!", "OK",
"Information", "Button1")
  EndIf
EndSub
```

All we did was add connect the subroutine `ButtonClick` to the `IOForms.OnButtonClick` event. In that subroutine, we checked that our button was the one that was clicked (this actually only necessary when more than one button is used), then showed a dialog stating the button was clicked. (If you are curious about `IOForms.ShowMessage`, check out its IntelliSense documentation.

Now, you should be ready to play with other controls! They all follow the common parameter pattern we just used. All methods to add controls start with "`IOForms.Add…`" and all events begin with "`IOForms.On…`", and all methods to modify controls start with that control's name (for example, the method to change a `TextBox`'s text is `IOForms.TextBoxSetText`).

More fun!

There is much more to the IO Extension that IOForms! Check out the Samples folder for good samples of these features. Here is a complete listing of what method groups can do what:

- `IOAlgorithms` allows you to sort numbers, calculate a password's strength, or calculate the value of the mathematic constant pi to a specified number of digits
- `IOClock` allows you to get data from the system clock beyond what the `Clock` method group can do
- `IOFile` allows you to preform file operations beyond what the `File` method group can do (such as `FileSystemWatcher` events)
- `IOForms` allows you use (one or more) Windows Forms, 24 different controls such as `ListView`s, `TreeView`s and `CommandLink`s, use Office-2007-style `Ribbon`s, and use Windows common dialog boxes and preform advanced tasks such as listening to the window's message pump
- `IOGameControllers` allows you any type of game controller in Small Basic
- `IOPrinter` allows you use printers in Small Basic
- `IOSensors` allows you use sensors (requires Windows 7 or higher) in Small Basic (but is untested since my PC has no sensors) and to use the Google Maps API, such as getting Street View or Map imagery, geocoding or reverse geocoding, looking up elevation for points, getting directions to anywhere, and getting maps showing the routes of directions.
- `IOSpeech` allows you to synthesize speech with a variety of parameters
- `IOTCPServer` and `IOTCPClient` allow you to send messages between a server and a client (much like `NetworkServer` and `NetworkClient` in Oskariok's Data Extension)
- `IOTimers` allows you to create, manipulate (and remove) as many timers as you need

Known issues

- `IOAero.ExpandGlass()` does not correctly use margin values in Windows 8, so some experimenting is necessary to get the margins correct. This bug is possibly due to the fact that the VistaBridge library is designed for Windows Vista and has problems with current versions of DWM (**D**esktop **W**indow **M**anager) that are shipped with Windows 7/8.
- When collapsed to the point of becoming an `Overflow` tab, `RibbonTab`s do not expand correctly when their drop-down arrow is clicked. To work around this issue, set the `WindowMinWidth` of the form to a value at which the ribbon is fully expanded.
- `WebBrowser` controls require form UI threads to be `STA` (serial-threaded apartments) since ActiveX controls require STA, so when they are added to a form before it is initialized, they are added to a custom object called a `WebBrowserQueue` in which they are stored until the form is

initialized. Upon initialization, a new thread is created in which all the objects in the queue are converted into controls on the form and the window is initialized as an STA Thread.

Notes

Some features of the extension are hidden from the Intellisense system of Small Basic. Among the useful features that have been hidden (to prevent confusion) are:

- In `IOForms`, you can use `AddWindow(Primitive WindowName)` to add a window to the program, use `WindowSelector` to set all calls to go to that window, then use `ShowWindow(Primitive WindowName)` or `ShowWindowAsDialog(Primitive WindowName)` to show the window. You can change the selector property "MainForm" to direct calls back to the main form.
- `IOShapes` and certain features of `IOAero` are hidden from Intellisense. They are works in progress. Please, please do not use them! One such method contains a security vulnerability. I will remove the `[HideFromIntellisense]` attribute when they are ready for use.