

# Creating Dynamic User Interfaces in Liquid Handling using Visual Basic for Applications

Nicholas Lin, Caliper Life Sciences, Hopkinton, MA USA

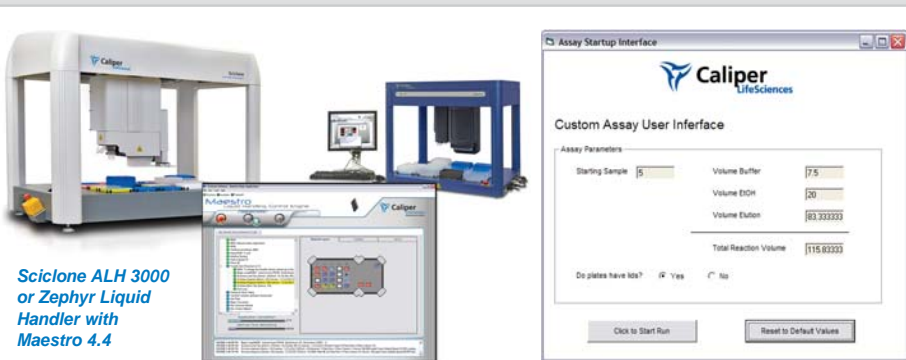
## Introduction

Liquid handling using laboratory automation today should be an easy and intuitive process for scientists and lab users to use in a production or lab environment. There may be times where liquid handling can appear to be complex because of exposed method parameters and variables that the end user visualizes while looking through a method. Simplicity and intuitiveness in an automation interface is important especially when more than one person in the lab is using the instrumentation.

Caliper Life Sciences' Maestro software, which drives both the Sciclone ALH 3000 and Zephyr liquid handlers, have tools built in that will enable the developer to create startup user interfaces to create an environment tailored to the specific needs of the end user. Visual Basic is embedded as part of the Maestro software and will give the developer the tools necessary to design and launch Graphical User Interfaces (GUI) directly from the liquid handling software.

With the ability to customize the look and feel of how users want to interact with programs, the process of starting the automated process becomes a lot easier without having the worry of main applications being accidentally modified.

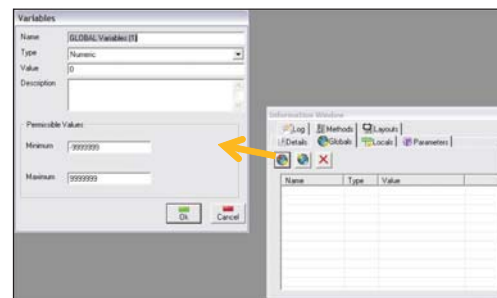
## Materials



## Methods

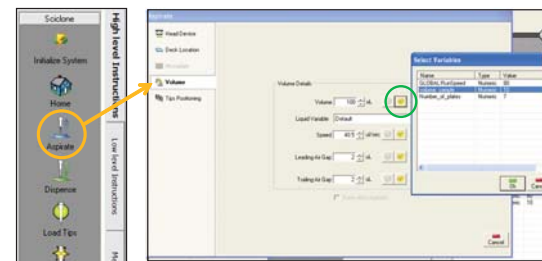
Those who are familiar with Visual Basic will find the integrated IDE environment makes programming and creating user forms a very simple and easy process. Those who are not familiar with Visual Basic can easily pick up how to create custom user interfaces by following examples given on this poster.

The purpose of creating custom user interfaces is to create an easy way for anyone to adjust volumes, speeds, repeating number of iterations, etc without having to edit their primary application. Once the application is validated it's best to leave it alone and change only specific parameters without risking accidental changing of methods.



## Methods (cont)

Prior to starting a Visual Basic form, a few variables will need to be added to the Maestro application. In this example, click on the global's variable tab on the "Information Window" and add a variable called "volume\_sample". The variable "volume\_sample" can be used throughout the application so that if the value required a change, the variable value can be modified without having to change the application steps. If there are any other variables you wish to create at this point that you would like to add to your Visual Basic user form you can add the variables now.



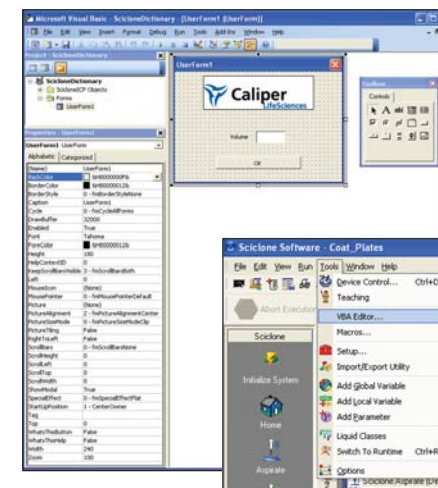
To apply variables created in the global variables tab to the method, a method instruction will need to be dragged over from the left window toolbar in Maestro to the method itself. In this example, drag an "Aspirate" step from the "High level Instructions" tab over to the method. When you get to the volume specification section tab, you will notice that there is a "V" button next to the volume input box. This "V" stands for variable. When you click on the "V", a new window will appear that contains any variables you created previously. By selecting the variable, any value assigned will be used dynamically in the "aspirate" step.

The user interface can be developed at this stage. With Maestro open, go to the tool bar and select "Tools" -> "VBA Editor". You will notice that a new window will appear over Maestro with the Visual Basic Development environment.

On the top left corner of the Visual Basic Application you will find the project explorer. The name of the project is ScicloneDictionary. By expanding the folder options in the Project Explorer, you will see ScicloneDictionary -> ScicloneICP Objects -> Sciclone. Double clicking the Sciclone entry will cause the coding window to appear in the window to the right. In the coding window a subroutine will need to be added once the form is generated.

To create a custom user interface, you need to go to the toolbar and select Insert -> UserForm. A blank form titled "UserForm1" will be visible in the right window.

You will also notice a Toolbox that appears. This Toolbox contains the tools needed to create titles, buttons or to display images you want the end user to see. By selecting a tool and dragging it to the form, you can easily build a user interface. For this example, drag 4 items from the toolbox to the form (Label, Command Button, Text Box and Image). The objects on the form will have generic names such as "CommandButton1" or "Label1". We can change the button's displayed value by single clicking on the button and modifying its properties located in the properties window below the project explorer. Change the default value of "CommandButton1" to "OK". If you click on the label object, a new set of properties will appear in the properties window. Each object on your form will have its own set of values. Change the "Caption" on the label tool to say "Volume". The last item to modify is the "Image" object. The image object is used to add graphics or a logo to your user form. If you have a company logo image, you can use this to make your user form more interesting. Selecting "Picture" in the properties window will allow you to browse and select an image to display. At this point you should have an interface that needs a little visual basic coding.



## Methods (cont)

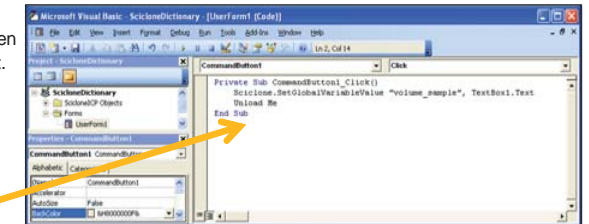
The coding will allow the user interface to pass the value in the "Volume" text box to the Maestro Global Variable list. To add code, double click the command button. When the coding window opens, the following should appear.

```
Public Sub CommandButton1_Click  
End Sub
```

This is the code that will run once the command button is pressed. Since there is no code between Public Sub and End Sub nothing will run at this point. The objective is to pass the "Volume" value in the form to the variable in Maestro software.

To add code, type the following between the Public Sub and End Sub statement.

```
Public Sub CommandButton1_Click()  
Sciclone.SetGlobalVariable  
"volume_sample", TextBox1.Text  
Unload Me  
End Sub
```



End Sub

The "Sciclone.SetGlobalVariable "volume\_sample", TextBox1.Text" sets the global variable "volume\_sample" to the value of what the user puts into the textbox in the user interface. The "Unload Me" statement unloads the user form so that the user does not see the form once the command button is pressed.

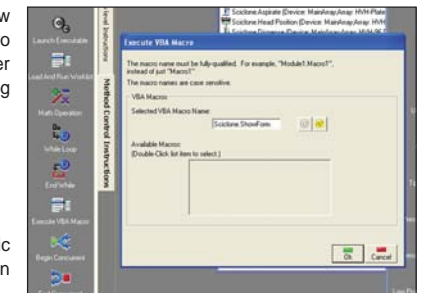
The final coding step is adding a subroutine that will allow your form to appear when you run the macro in the Maestro application. Double click "Sciclone" in the project explorer and add the following to the very bottom of the coding window.

```
Public Sub ShowForm()  
Userform1.Show  
End Sub
```

This is all the coding you have to do in the Visual Basic editor. The only step left is to call this macro "ShowForm" in the Maestro application.

Once everything is saved, the Visual Basic Editor windows can be closed or minimized. In Maestro select the bottom tab labeled "Method Control Instructions". You will see a "Run Macro" tool. You will want to drag this into your method. When the window for "Run Macro" appears, type the following into the box "Sciclone.ShowForm".

To test your user form, click the "Play" button in Maestro. The form should become visible and can be interacted with. Once you type a value into the volume box and click the command button, you should notice that the variable in the Global Variables box change value.



## Conclusion

With the flexibility of creating custom user interfaces, developers can control what the users can or cannot modify in a validated application. This feature is designed to ensure the smoothest process in an environment where many people use the instrumentation. By taking a few steps to create a user interface, it will increase productivity by reducing the time it takes to run an applications that has variability as well as reducing the downtime because of mistakes. In the example shown, volume was the only parameter changed. There are many other features that can be added such as logging users who run the program to error recovery options. Because this uses Visual Basic there are endless possibilities.

I would like to express my gratitude to Raymond Basherad for his assistance and support.