David Katherman
ISYS 540 – VBA

Dynamic Database

ISYS 540 Final Project

Executive Summary

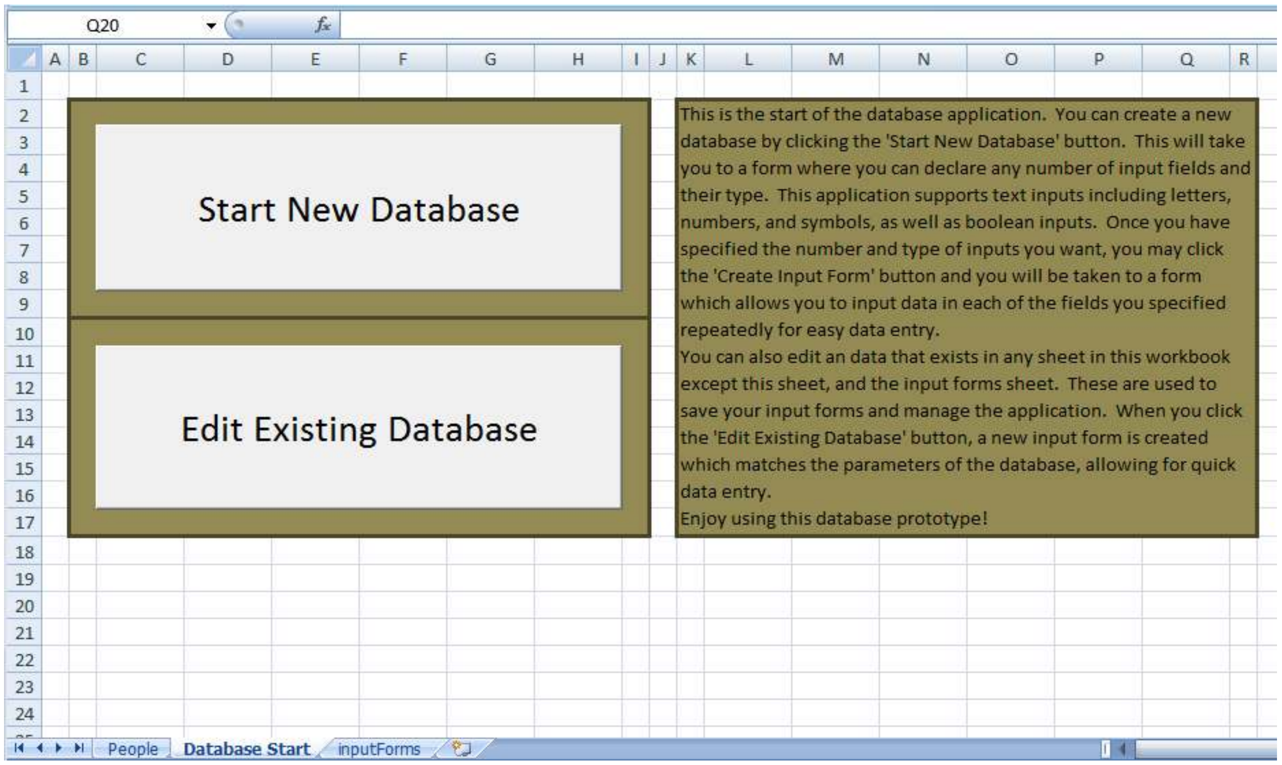
This project was originally conceived as a ‘pocket database’ application for a mobile platform, allowing a user to dynamically build, update, and search small databases on the go at worksites or business meetings. The version of the project explained below is a prototype of the pocket database system, built in Microsoft Excel using visual basic for applications and called ‘Dynamic Database’.
The Dynamic Database program allows a user to create a database composed of values held in a specific worksheet in the workbook. The user may specify what fields he wishes the database to take in. Dynamic Database will take the specified information and dynamically create an input form by which the user may quickly add information to the sheet containing the database.

Dynamic Database also allows the user to edit an existing database. The user may select the existing database from a list, and the program dynamically recreates the input form for that database from the saved input form information, kept on a separate sheet and inaccessible to the user. The user may then use the input form to quickly add more data to the database.

While a user could manually create the sheets and column headers and manually search and edit the information contained thereon, Dynamic Database was conceived to help me work through the logic of dynamically creating userforms and form controls from a series of user inputs, and then successfully placing the information entered into those form controls into a file. Documentation and forums suggest it is not possible in vba, where it is discussed at all, but I have found that it is possible, although it is not a small undertaking.   Dynamic Database is described in detail below.

Implementation Documentation

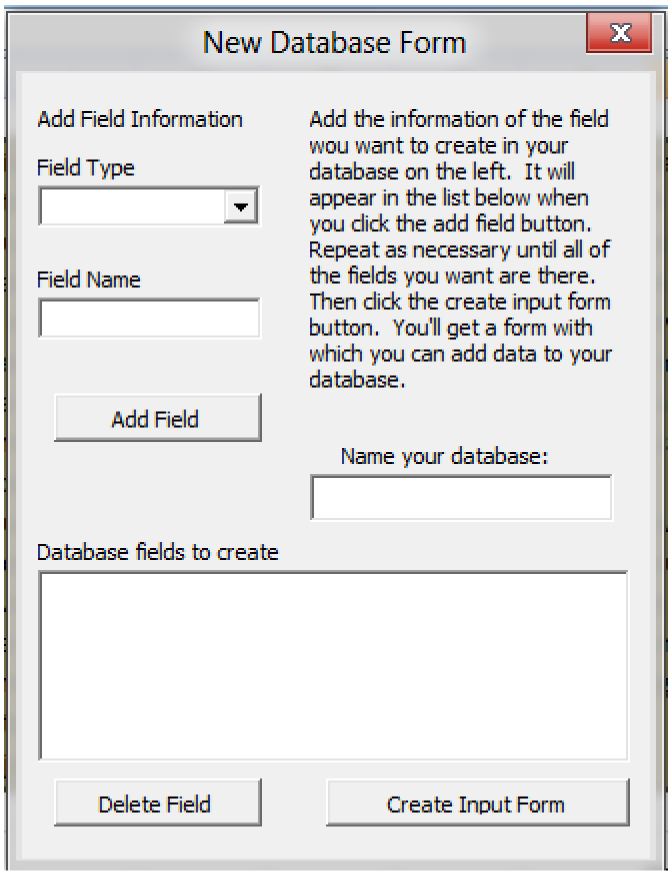
The Excel file containing Dynamic Database has two worksheets by default. When Dynamic Database is opened a subroutine is automatically triggered which selects the first of these two worksheets, called 'Datbase Start' which is shown in Figure 1.



This is the start of the database application. You can create a new database by clicking the 'Start New Database' button. This will take you to a form where you can declare any number of input fields and their type. This application supports text inputs including letters, numbers, and symbols, as well as boolean inputs. Once you have specified the number and type of inputs you want, you may click the 'Create Input Form' button and you will be taken to a form which allows you to input data in each of the fields you specified repeatedly for easy data entry.
You can also edit an data that exists in any sheet in this workbook except this sheet, and the input forms sheet. These are used to save your input forms and manage the application. When you click the 'Edit Existing Database' button, a new input form is created which matches the parameters of the database, allowing for quick data entry.
Enjoy using this database prototype!

Figure 1

Database Start contains the basic instructions for utilizing the database, as well as two buttons which launch the program. The button labeled 'Start New Database' links to a subroutine which simply initializes a vba userform called 'newDBForm.' NewDBForm contains more instructions on the use of the form and input controls which allow the user to specify the type of field he wants the database table to accept and name it. A button below these inputs will add the specified type of field and name to a listbox control as a string of the form 'type, name.' As soon as the string is added to the listbox control the inputs that were used to construct it are cleared to accept new data. NewDBForm is shown in figure 2 below.



Add the information of the field wou want to create in your database on the left. It will appear in the list below when you click the add field button. Repeat as necessary until all of the fields you want are there. Then click the create input form button. You'll get a form with which you can add data to your database.

Figure 2

The form controls are coded so that none of the above functions of the form will execute if a needed input is missing or blank. In such a case a message box will appear and inform the user of their error and exit the click action subroutine.

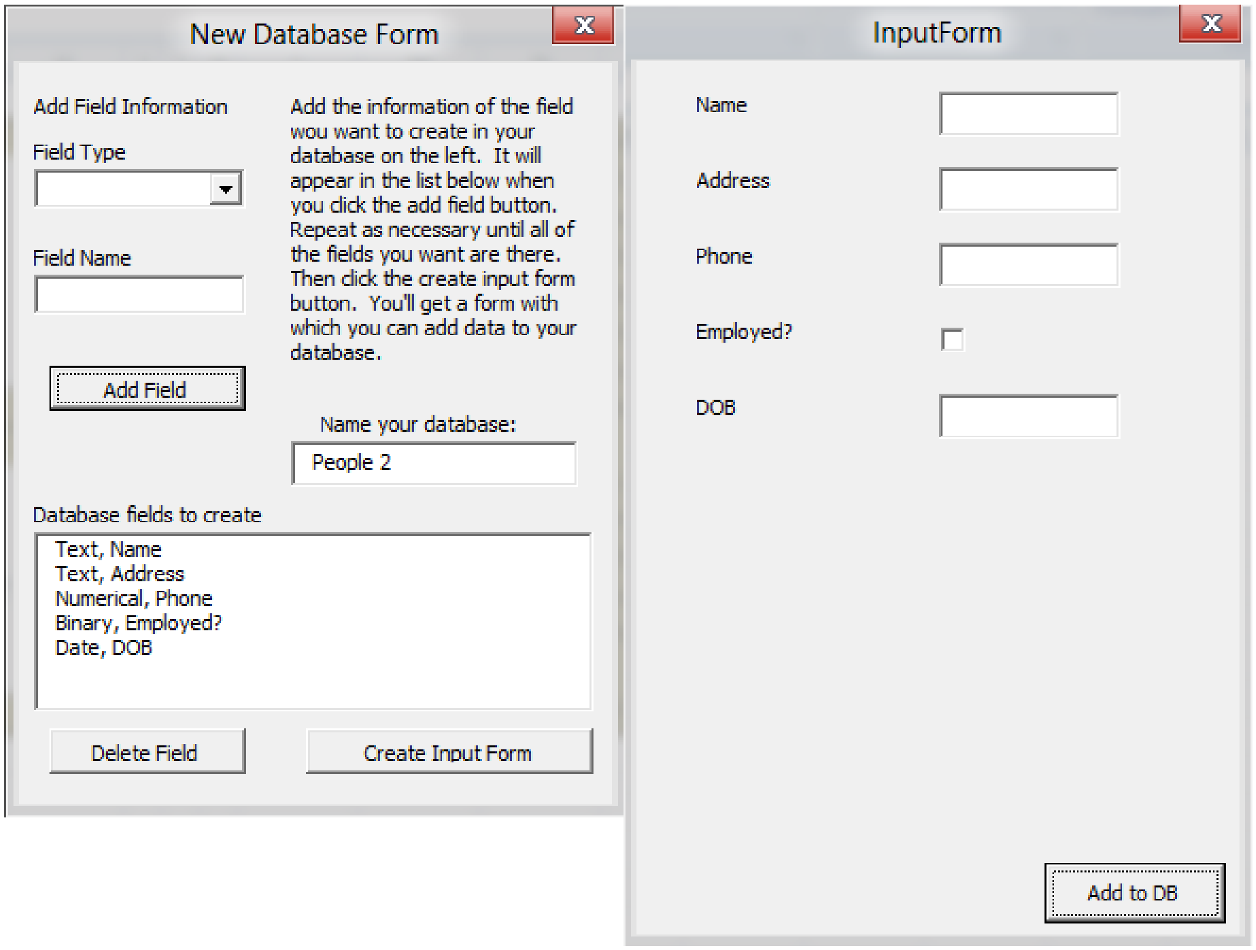
Finally, this form allows the user to input the name of the database. Once all of the fields which will accept data in the database table are added to the listbox control and the database is named, the user can click the 'Create Input Form' button.

Once the create input form button is clicked, the items in the listbox control are added to an array that is dynamically sized to fit exactly the number of fields that the user specified. This is done using the redim statement. Inside a loop, each string is then parsed into separate variables holding the type and name of the controls the user wants to create. If statements then query the variable holding the type string and a variable that was created as a control is set to be a form control of the given type, and is attached to a userform called 'inputForm' which is the blank slate to which the program adds each input control to create the input form dynamically. InputForm is created with a single button by default. In the same loop a label control is created and attached to the userform using a different control variable. This is to identify the input control the program just created on the form and the caption is set to the name string. Each label is added to a separate array for later use.

The variable that was created as a control, which is used to hold the control while it is attached to the form is then re-used each time the loop repeats as a new ‘type, name’ pair is pulled from the array and parsed into separate strings to determine the type of input control to create. A part of the process is shown in the code snippet below:

```
For i = 0 To UBound(fields) - 1
    fieldType = Mid(fields(i), 1, InStr(1, fields(i), ",") - 1)
    fieldName = Mid(fields(i), InStr(1, fields(i), ",") + 2)

    If fieldType = "Text" Or _
    fieldType = "Numerical" Or _
    fieldType = "Date" Then
      Set label = inputForm.controls.Add("forms.Label.1")
      With label
        .Caption = fieldName
        .Left = 24
        .Top = (12 + (30 * i))
      End With

      labels(i) = label

      Set textbox = inputForm.controls.Add("forms.TextBox.1", fieldName)
      With textbox
        .Left = 120
        .Top = (12 + (30 * i))
      End With
```

Note that the form creation code (Set textbox = inputForm.controls.Add("forms.TextBox.1", fieldName)) includes the name of the field, this is so the control can be accessed for its value later, when the data needs to be written to the database table. Note also that the complete source code can be found in Appendix 1. Finally, the database name taken from the newDBForm is written to a global variable for storage and later use. The relationship between the newDBForm with the list of user specified fields, and the actual input form that is created from that information is shown in Figure 3.



Add the information of the field wou want to create in your database on the left. It will appear in the list below when you click the add field button. Repeat as necessary until all of the fields you want are there. Then click the create input form button. You'll get a form with which you can add data to your database.

People 2

Text, Name
Text, Address
Numerical, Phone
Binary, Employed?
Date, DOB

Figure 3

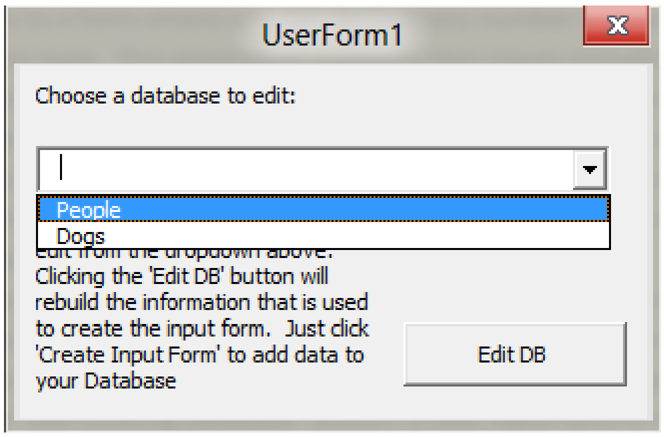
Once the inputForm is completely created, the newDBform is unloaded. In order to save the form for later use, the string values of the 'type, name' pairs (which were written to an array when the program started to create the inputForm) are written to the second of the two default worksheets in Dynamic Database, called 'inputForms.' Maintenance of the saved information in this sheet is discussed later.

The user can now use the fields in the dynamically created input form to accept inputs. Once the 'Add to DB' button is pressed, the values in the input controls in inputForm are written to the database worksheet. The subroutine first checks that the name of the database in the global variable exists as the name of a worksheet in the sheets collection. If not, a worksheet with the name of the database is created and the column headers are added by looping through the labels array to get the name of each input. Once the column headers are added to the sheet, the values from the input controls are added to the sheet under their respective column headers. This is accomplished through code like that in the snippet below:

```
Sheets.Add.Name = dataBaseName
        ' enter data into the form
        Sheets(dataBaseName).Select
        Range("A1").Select

        ' add the column headers to the DB sheet
        For i = 0 To UBound(labels) - 1
            Selection.Value = labels(i)
            Selection.Font.Bold = True
            Selection.Offset(0, 1).Select
        Next

        Range("A1").Select

        ' find a blank row
        Do While Selection.Value > ""
            Selection.Offset(1, 0).Select
        Loop

        ' add the data from the input form to the blank row
        For i = 0 To UBound(labels) - 1
            Selection.Value = inputForm.controls.Item(labels(i))
            Selection.EntireColumn.AutoFit
            Selection.Offset(0, 1).Select
        Next

    End If
```

If the sheet that holds the database table already exists (as in the second time data is added to the sheet or if the database is edited later) then the proper sheet is simply selected and the data is added to the first blank row that is found. The user can then use the input form to add as much data as needed before exiting the form and saving the workbook.

In order to edit the database the user created, the second button on the previously mentioned 'Database Start' sheet is labeled 'Edit Existing Database.' This button is linked to a simple macro that initializes a third userform called 'editForm.' This form is shown in Figure 4.



Edit from the dropdown above. Clicking the 'Edit DB' button will rebuild the information that is used to create the input form. Just click 'Create Input Form' to add data to your Database

Figure 4

EditForm contains instructions on how to edit an existing database and a combobox that is populated with the names of all of the sheets holding database information. The user may select the database from this combobox and click the 'Edit DB' button, which calls a subroutine that finds the name of the database and the field information in the inputForms sheet where the information needed to recreate the input form is stored. This subroutine then initializes the newDBform and loads each of the saved 'type, name' pairs into the listbox control. The user can then click the 'Create Input Form' button to recreate the proper input form. Because the database in question already exists, any data entered into the form is simply inserted in the proper order into the first blank row in the database sheet.

Because the application was designed to be a prototype of the database only and was intended to provide learning on the creation of dynamically created user forms, this is the limit of its functionality.

Learning Concepts, Difficulties, and Assistance

I genuinely struggled with several aspects of this project. The first and foremost was the creation of dynamic forms. I spent two days researching on multiple forums using many different search strings but was unable to find any forum in which the experts said it was possible (or recommended) to create dynamic form controls.

What little I did manage to find suggested, at least to my mind, that I would need a separate variable to hold each form control. Operating under this erroneous assumption I began to research how to create variables dynamically. There was even less information about this. What I thought I needed was to write code that could write code; something that could dim a set of new variables on the fly as needed.

I discovered that facilities existed in other programming language to do things like this. For example Java has the eval statement, which would look something like:

```
int i = 0
String variableName = “VariableName”

For(i = 0; i<array.len; i++){
        Eval(variableName + int.toString())
}
```

Unfortunately, I couldn’t find any such facility in vba. As a matter of fact, most of what I was finding in the forums directed me to use an array, but that array still needed to be filled with something. I eventually worked out the idea that I could place a dim statement within a loop to create the variable type I would need to create the form control. I discovered that this would be the equivalent of trying to dim the same variable twice in the same program and would crash it. I could have created the variable in the beginning of the subroutine, but I was convinced that without the capacity to create multiple variables, I would have to re-use that single control variable. I was further convinced that re-using that variable would overwrite the same spot in memory, leaving me with a single control instead of the multiple controls I would need.

By then I had the code I needed to programmatically create the proper controls, and I knew the logic that I needed. I just didn't think I could create all of them with a single variable. I expressed my problem to William Day, who graduated MISM from BYU in 2008 and is a very good friend and confidant of mine. I expressed my ideas and my concerns about the single control variable problem, and he taught me a lesson.

He wrote a simple program that took in a number, and dynamically created a number of fields equal to that number on a new form. He did it using a single control variable to create the control, which was re-used each time the loop reset, and it worked. I had so convinced myself of what I thought the code would do that I didn't bother to code a quick test to be sure. If I had I could have saved myself a whole day, and several good natured jibes from Will.

Will sent me his program and it matched almost exactly the logic I had previously worked out, only with the single, re-useable control variable. I coded my own program with a single variable (like I had neglected to test) and it worked like a charm. I did not receive any other substantial assistance in this project.

The second thing that I struggled with was finding a way to access each of the controls I created so that I could get the value of that item. My original thought was to put them all into a global array at creation, which I would then be able to loop through to get each control back out. This didn't work for two reasons. First and foremost, if I had two different kinds of controls I would end up with a type mismatch when attempting to add one or the other to the array. For example, if I had both text boxes and boolean checkboxes, then I could not add both to the array. Second, that global array would be destroyed when the program ended, meaning I could not access the form controls coming back to the program after it had ended, or if I were working with more than one database. I worked on it for hours, but clearly this wasn't the answer.

As I dug back into the methods that create the form controls I realized that I could name the control at creation, and then access it at any time during execution by name. This allowed me to get the data from the input forms and write it to the proper spreadsheet.

The major lesson in all of this was that there is no harm in coding a short program to be sure a program will do exactly what you think it will do.   Chances are you'll be surprised when it doesn't. If I had just tried what I thought wouldn't work to be sure I could have saved myself a great deal of time, and in the future I will definitely do more coding and less thinking about what might go wrong. I'll get more accurate results from coding the idea than what I think it might do.

Appendix 1

Source Code

Database Start Module

```
Sub startNewDatabase_Click()

   newDBForm.Show

End Sub

Sub editExistingDatabase_Click()
   editForm.Show

End Sub
```

newDBForm

```
Private Sub addFieldButton_Click()

   Dim fieldType As String
   Dim fieldName As String

   fieldType = fieldTypeCombobox.Value
   fieldName = fieldNameTextbox.Value

   If fieldType > "" And fieldName > "" Then
      fieldsList.AddItem (fieldType & ", " & fieldName)
      fieldTypeCombobox.Value = ""
      fieldNameTextbox.Value = ""
   Else
      MsgBox ("You tried to add a blank field... You can't do that.  You know that don't you? Try again.")
   End If

End Sub


Private Sub createInputFormButton_Click()
   inputForm.Show
End Sub

Private Sub deleteFieldButton_Click()
```

```
    If fieldsList.ListIndex = -1 Then
        MsgBox ("You have to select a field in order to remove it.  You're seriously not very good at this are
you?")
    End If
    On Error Resume Next
    fieldsList.RemoveItem (fieldsList.ListIndex)
End Sub

Private Sub UserForm_Initialize()
    With fieldTypeCombobox
        .AddItem ("Text")
        .AddItem ("Binary")
        .AddItem ("Numerical")
        .AddItem ("Date")
    End With

    fieldTypeCombobox.Value = ""

End Sub
```

inputForm

```
    Dim labels() As Variant
    Dim inputs() As Variant
    Dim dataBaseName As String

Private Sub addToDB_Click()
    Dim i As Integer
    Dim sheetExists As Boolean
    Dim ctl As Control
    Dim fields() As Variant

    ' redim the size of the fields array to hold the number
    ' of fields the user created
    ReDim fields(0 To newDBForm.fieldsList.ListCount)

    ' fill the fields array with the fields user created
    For i = 0 To newDBForm.fieldsList.ListCount - 1
        fields(i) = newDBForm.fieldsList.List(i)
    Next

    ' check the sheets to see if the database the user named
    ' already exists
    For i = 1 To Sheets.Count
        If Sheets(i).Name = dataBaseName Then
            sheetExists = True
        End If
    Next

    ' add the data to the specified sheet because it exists already
    If sheetExists Then
        Sheets(dataBaseName).Select
        Range("A1").Select

        Do While Selection.Value > ""
            Selection.Offset(1, 0).Select
        Loop

        For i = 0 To UBound(inputs) - 1
            Selection.Value = inputForm.controls.Item(labels(i))
            Selection.EntireColumn.AutoFit
            Selection.Offset(0, 1).Select
        Next

    ' if sheet doesn't exist, we'll need to create it, then
    ' add the column headers, and the corresponding data below them
    Else
        Sheets.Add.Name = dataBaseName

        ' enter data into the form
        Sheets(dataBaseName).Select
        Range("A1").Select

        ' add the column headers to the DB sheet
        For i = 0 To UBound(labels) - 1
            Selection.Value = labels(i)
            Selection.Font.Bold = True
            Selection.Offset(0, 1).Select
        Next

        Range("A1").Select

        ' find a blank row
        Do While Selection.Value > ""
            Selection.Offset(1, 0).Select
        Loop

        ' add the data from the input form to the blank row
        For i = 0 To UBound(labels) - 1
            Selection.Value = inputForm.controls.Item(labels(i))
            Selection.EntireColumn.AutoFit
            Selection.Offset(0, 1).Select
        Next
        End If

        ' clear out the inputs on the form so they are ready for
        ' more input
        For Each ctl In inputForm.controls
            Select Case TypeName(ctl)
                Case "TextBox"
                    ctl.Text = ""
                Case "CheckBox"
                    ctl.Value = False
                Case "ComboBox"
                    ctl.Value = ""
                Case "ListBox"
                    ctl.Value = ""
                Case "OptionButton"
                    ctl.Value = False
                Case "ToggleButton"
                    ctl.Value = False
            End Select
        Next ctl

    End Sub

    Private Sub UserForm_Initialize()
        Dim fields() As Variant
        Dim i As Integer
        Dim j As Integer
        Dim temp As String
        Dim label As MSForms.label
        Dim textbox As MSForms.textbox
        Dim checkbox As MSForms.checkbox
        Dim fieldType As String
        Dim fieldName As String
        Dim exists As Boolean
        Dim sht As Worksheet

        ReDim fields(0 To newDBForm.fieldsList.ListCount)
        ReDim labels(0 To UBound(fields))
        ReDim inputs(0 To UBound(fields))

        dataBaseName = newDBForm.DBName.Value

        For i = 0 To newDBForm.fieldsList.ListCount - 1
            fields(i) = newDBForm.fieldsList.List(i)
        Next
```

```
For i = 0 To UBound(fields) - 1
    fieldType = Mid(fields(i), 1, InStr(1, fields(i), ",") - 1)
    fieldName = Mid(fields(i), InStr(1, fields(i), ",") + 2)

    If fieldType = "Text" Or _
    fieldType = "Numerical" Or _
    fieldType = "Date" Then
        Set label = inputForm.controls.Add("forms.Label.1")
        With label
            .Caption = fieldName
            .Left = 24
            .Top = (12 + (30 * i))
        End With

        labels(i) = label

        Set textbox = inputForm.controls.Add("forms.TextBox.1", fieldName)
        With textbox
            .Left = 120
            .Top = (12 + (30 * i))
        End With

    ElseIf fieldType = "Binary" Then
        Set label = inputForm.controls.Add("forms.Label.1")
        With label
            .Caption = fieldName
            .Left = 24
            .Top = (12 + (30 * i))
        End With

        labels(i) = label

        Set checkbox = inputForm.controls.Add("forms.CheckBox.1", fieldName)
        With checkbox
            .Left = 120
            .Top = (12 + (30 * i))
        End With

    End If
Next

' we need to save the form
    ' - take the fields array and write its contents to
    ' a blank row in the hidden inputForms sheet
Sheets("inputForms").Select
Range("A1").Select
```

```
    exists = False

    Do While Selection.Value > ""

        For Each sht In Sheets
            If dataBaseName = Selection.Value Then exists = True
        Next

        If exists = True Then
            Exit Do
        End If

        Selection.Offset(1, 0).Select
    Loop

    If exists = False Then
        'find an empty row
        Do While Selection.Value > ""
            Selection.Offset(1, 0).Select
        Loop

        Selection.Value = dataBaseName
        Selection.Offset(0, 1).Select

        For i = 0 To UBound(fields) - 1
            Selection.Value = fields(i)
            Selection.Offset(0, 1).Select
        Next
    End If

    Unload newDBForm

End Sub
```

EditForm

```
Dim labels() As Variant

Private Sub editDBButton_Click()
    Dim fields() As Variant
    Dim i As Integer
    Dim j As Integer
    Dim fieldSize As Integer
    Dim temp As String
    Dim label As MSForms.label
    Dim textbox As MSForms.textbox
Dim checkbox As MSForms.checkbox
Dim fieldType As String
Dim fieldName As String

'find the number of fields so we can resize the array
Application.ScreenUpdating = False
temp = editForm.dbListCombo.Value

Sheets("inputForms").Select
Range("A1").Select

Do While Selection.Value > ""
  If Selection.Value = temp Then
    Selection.Offset(0, 1).Select
    Exit Do
  End If
Loop

Do While Selection.Value > ""
  If Selection.Value > "" Then
    fieldSize = fieldSize + 1
    Selection.Offset(0, 1).Select
  End If
Loop

ReDim fields(0 To fieldSize)
ReDim labels(0 To UBound(fields))

dataBaseName = editForm.dbListCombo.Value

' load the fields array with the saved data in inputforms
Range("A1").Select

Do While Selection.Value > ""
  If Selection.Value = temp Then
    Selection.Offset(0, 1).Select
    Exit Do
  End If
Loop

For i = 0 To fieldSize - 1
  If Selection.Value > "" Then
    fields(i) = Selection.Value
    Selection.Offset(0, 1).Select
  End If
Next
```

```
For i = 0 To fieldSize - 1
   newDBForm.fieldsList.AddItem (fields(i))
Next

newDBForm.DBName.Value = dataBaseName

Application.ScreenUpdating = True

Unload editForm

Sheets(dataBaseName).Select

newDBForm.Show
' inputForm.Show
' newDBForm.Hide

' Build the form
'For i = 0 To UBound(fields) - 1
   'fieldType = Mid(fields(i), 1, InStr(1, fields(i), ",") - 1)
   'fieldName = Mid(fields(i), InStr(1, fields(i), ",") + 2)

   'If fieldType = "Text" Or _
   'fieldType = "Numerical" Or _
   'fieldType = "date" Then
      'Set label = inputForm.controls.Add("forms.Label.1")
      'With label
         '.Caption = fieldName
         '.Left = 24
         '.Top = (12 + (30 * i))
      'End With

      'labels(i) = label

      'Set textbox = inputForm.controls.Add("forms.TextBox.1", fieldName)
      'With textbox
         '.Left = 120
         '.Top = (12 + (30 * i))
      'End With

   'ElseIf fieldType = "Binary" Then
      'Set label = inputForm.controls.Add("forms.Label.1")
      'With label
         '.Caption = fieldName
         '.Left = 24
         '.Top = (12 + (30 * i))
      ''End With
```

```
        'labels(i) = label

        'Set checkbox = inputForm.controls.Add("forms.CheckBox.1", fieldName)
        'With checkbox
          '.Left = 120
          '.Top = (12 + (30 * i))
        'End With
      'End If
    'Next

    ' inputForm.Visible = True
End Sub

Private Sub UserForm_Initialize()
    Dim sht As Worksheet
    Dim inSheets As Boolean

    ' add all the databases in existence to the combobox for selection
    For Each sht In Sheets
      If sht.Name <> "Database Start" And sht.Name <> "inputForms" Then
        dbListCombo.AddItem (sht.Name)
      End If
    Next

    Application.ScreenUpdating = False
    Sheets("inputForms").Select
    Range("A1").Select

    ' find form info that doesn't have a database and delete it
    Do While Selection.Value > ""

      inSheets = False

      For Each sht In Sheets
        If sht.Name = Selection.Value Then inSheets = True
      Next

      If inSheets = False Then
        Selection.EntireRow.Delete
      End If

      Selection.Offset(1, 0).Select
    Loop

End Sub
```