Technical Guide to the Project Tracking Application

Last revised by James Midkiff – May 2019

# Table of Contents

* Introduction
* Main Menu
* Open Projects Form
* Consultants Form
* Clients Form
* Analysts Form
* Project History Form
* Update Project Form
* Add New Project Form
* Troubleshooting
* Glossary

# Introduction

## Purpose

This is an in-depth technical guide meant for analysts and managers to learn how the Project Tracking Application (PTA) actually works for the purposes of:

* Maintaining it
* Expanding its features and capabilities
* Debugging it

For a user-oriented guide to the Project Tracking Application, see the *Consultants’ Guide*.

## Formatting Used in this Technical Guide

* **Bold\_text(*parameter*) –** indicates a specific code function / statement with a parameter that you can input.
  + For example, **IsNull(*object*)** tests if the *object* is null
* *Italics* – Indicates a module name, a form name, subroutine name, a variable, a query, a table, or an optional parameter.
  + *Combo\_Entity\_AfterUpdate()* is a subroutine name.
* ‘xxxxx’ – Indicates a field/column name, button name, or option.
  + ‘Combo\_Entity’ is a button.
  + ‘Entity’ means the field ‘Entity’.
* “xxxxx” – Indicates a particular value.
  + The ‘Entity’ field has the value “EASI”.
* Underlines – indicates a particular point of emphasis or a piece of code to pay attention to.
* Arrows “>” indicate a progressive series of steps in a menu.
  + i.e. Tools > Options > Editor Format – means go to the Tools menu, then the Options sub-menu, then the Editor Format sub-menu.

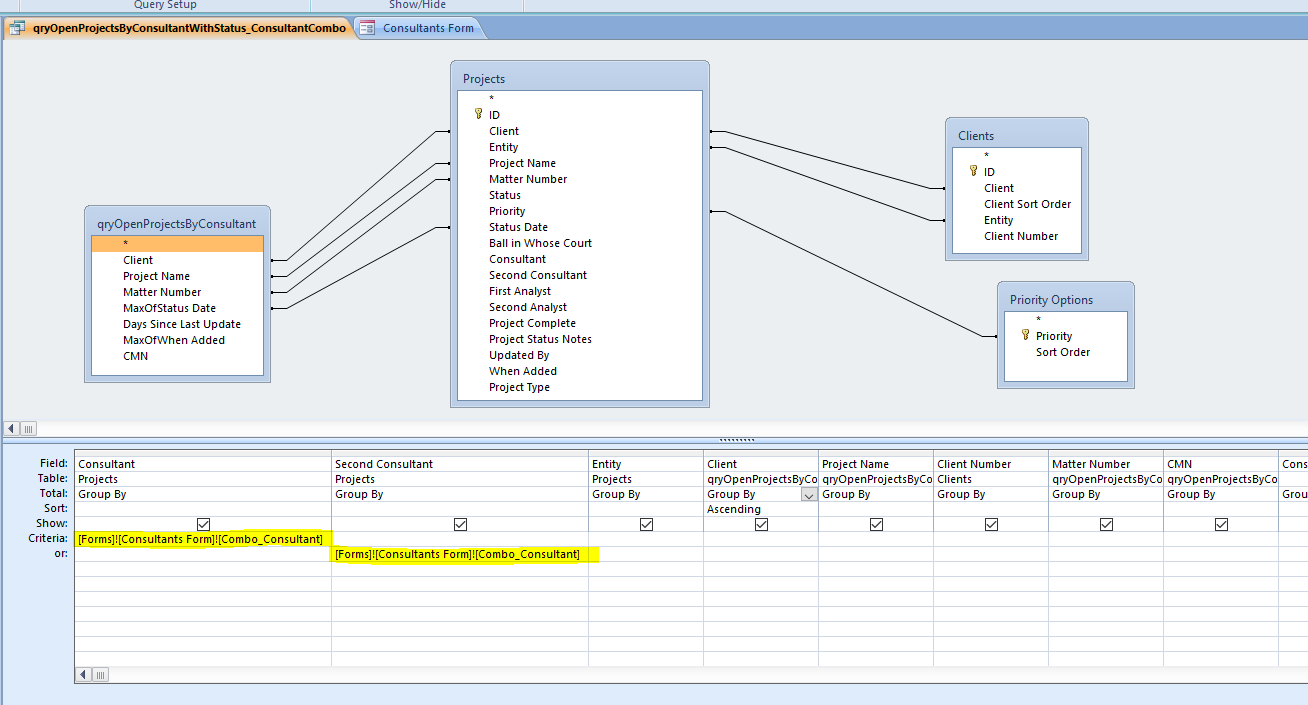
Most functions will appear with an explanation in the Glossary section at the end of this Technical Guide.

## Database

To access the inner workings of the database, after enabling the database click on the *Main Menu* button ‘Developers Only’.

The database is organized as such:

* All of the data and information is kept in the tables. The main table is *Projects*.
* The data is first transformed in the following order:
  + *qryOpenProjectsByConsultant\_Initial* gets the latest status update for each project
  + *qryOpenProjectsByConsultant* then adds an additional field ‘CMN’
  + *qryOpenProjectsByConsultantWithStatus* gathers all of the relevant fields and incorporates other information
  + Most forms then have their own unique query such as *qryOpenProjectsByConsultantWithStatus\_ConsultantCombo* that uses Combo Box selections as filtering criteria for the data.
  + I.e. below in the query *qryOpenProjectsByConsultantWithStatus\_ConsultantCombo*, the Record Source query will only show data where the ‘Consultant’ field is equal to the button ‘Combo\_Consultant’ OR the field ‘Second Consultant’ is equal to the button ‘Combo Consultant’.



* The forms have VBA code running behind them that takes the queries, searches them, modifies them, and then displays the appropriate data in the forms.
* The VBA code is kept in Class Modules (one module for each form), with various Subroutines that reference each other depending upon what the user does.
  + TO ACCESS THE VBA: Press Alt + F11, or go to Database Tools > Visual Basic.
  + The class modules are listed separately from the standalone module *Start-Up Code*.

Any time a project status is updated or a new project is added, that information is simply added to *Projects* and then the queries show that new record as the most up-to-date information. There is generally no need to delete or change anything from the tables unless there has been user error or some critical information has changed such as the project’s ‘Matter Number’.

This guide will go through each of the forms and the VBA associated with them by describing what each subroutine does. For additional questions, go to the glossary, google search, and/or figure it out yourself.

## VBA

VBA is not too complicated of a language especially if you have a working knowledge of R, Python, or the like. It does have a few quirks, so I will describe some important things to keep in mind with a screenshot. This technical guide and the code require a basic level of programming knowledge such as **If/Else** conditional logic, variable assignment, comments, and **For** loops.

In the below screenshot, I will note some of the basic elements of the VBA editor. The colors of your editor are likely different from what you have as the default on your system; I encourage you to customize your editor colors by going to Tools > Options > Editor Format. Doing so will make coding easier, more enjoyable, and more comfortable on the eyes.

See screenshot #9 in the Screenshots folder in the google drive link.

Following the numbers in the screenshot, here are some pointers to familiarize you with the basics

1. *Module Name –* This is the name of the VBA module you are currently working in. The modules can look very similar, so be sure to know which one you are working on. This module is named *Form\_Update Project Form*.
2. **Private Sub** ***module\_name()*** – VBA functions via subroutines (subs) which are a block of code that accomplishes a specific purpose.
   1. In the PTA subroutines can be called by another subroutine (see point 5) or by a user interacting with a form object that has code associated with it.
   2. Subroutines are very similar to functions except that subroutines cannot be called from queries and that [subroutines do not return a value](https://stackoverflow.com/questions/11539838/purpose-of-using-sub-routines-over-functions).
   3. The name of the subroutine generally gives a hint to either how it gets called or what it does. The subroutine *Combo\_Entity\_AfterUpdate()* is called from the button ‘Combo\_Entity’ after it has been updated.
   4. The **Private Sub** defines the scope of the subroutine, meaning that this sub can only be called by the form attached to this specific module and that the information created in this subroutine can’t be used outside of it. [See here](https://docs.microsoft.com/en-us/office/vba/language/concepts/getting-started/understanding-scope-and-visibility) for more information on scope.
   5. Subroutines must end with the statement **End Sub**.
3. Comments – An apostrophe is how VBA knows to signal a comment. VBA simply ignores comments when running.
4. [*object*].[*attribute*] *–* VBA functions similarly to Python in that periods (“.”) access the attributes of a particular object. Attributes are also known as Methods.
   1. For example, the statement **Check\_Project\_Complete.Value = 0** means that the Value of the checkbox ‘Check\_Project\_Complete’is now being set to zero.
   2. My naming convention for button prefixes:
      1. Check = Checkbox
      2. Combo = Combobox (i.e. a drop-down)
      3. Text = Textbox
      4. Command = A command button (a button that does something)
      5. Label = A text label
5. **Call [*subroutine*](*optional\_parameters*)** – This is how to the subroutines talk to each other, or call each other.
   1. **Call Restore\_Defaults(1)** will call the subroutine *Restore\_Defaults* with the parameter of 1. To see what the parameter means, you must find the called subroutine (in this case it will be the value for the paramater *Stage*.
   2. Unless the code indicates otherwise, all **Call** statements will call a subroutine within the same module.
6. **Debug.Print [*object*].[*attribute*]** – This runs the command **Debug.Print** on an object, printing the result in the Immediate Window (see point 8).
   1. This is the best way to check the value of an object’s attribute, though it may throw up an error if the object does not exist.
   2. For example, **Debug.Print ctrl.Name** will thus print the value of whatever the particular **ctrl** is at that moment (once the statement is uncommented). As this statement is in the middle of a **For** loop, each pass of the **For** loop will thus print the name of each **ctrl** that is affected.
   3. Be sure to uncomment or recomment Debug.Print statements as appropriate.
7. **Dim [*variable name*] As [*variable-type*]** – The top of each module should contain any options and variables you want available to the entire module. This again is an issue of variable scope.
   1. If you want to use assign a value to a variable in one subroutine, and then use that same variable and its value in another subroutine, you must define the variable at the top of the module with the statement **Dim** and then the variable type.
      1. Variable types include ‘String’, ‘Integer’, ‘Double’, ‘Variant’ (any type), among others.
8. Immediate Window – Functions similarly to the console in R but without showing what code is running. The immediate window is best used for viewing **Debug.Print** statements.
   1. To run a subroutine from the immediate window, type code similar to this and hit Enter:
   2. **Call [*Form\_Consultants Form*].*Temp*()**
      1. *[Form\_Consultants Form]* is the name of the form and *Temp()* is the name of the subroutine you want to run
   3. The form name must be surrounded in brackets if the name contains a space.

## Forms

The forms can be edited by going into the Form’s Design View. The Property Sheet, which is accessible via Alt + Enter or by going to Design > Property Sheet, contains the attributes for each object on the form. These attributes are the same ones controllable with VBA if a subroutine acts upon that object.

When working with most forms, the Record Source is the query or table that provides the information that populates the form. The modules for the forms *Open Projects Form*, *Consultants Form*, *Clients Form*, and *Analysts Form* change the Record Source with VBA code written in SQL format. IF YOU WANT TO CHANGE THE RECORD SOURCE, YOU MUST CHANGE IT THREE PLACES AND THE CODE MUST BE THE EXACT SAME SO THAT THEY ARE ALL CONSISTENT:

1. In whatever query is associated with the form
   1. I.e. If you’re changing the Record Source for *Consultants Form*, you should change the code in the query *qryOpenProjectsByConsultantWithStatus\_ConsultantCombo*.
   2. Changing the query first makes it easier to figure out what is the right SQL you will need for the VBA.
   3. Also if you don’t change it here, you won’t get the variables needed in your query.
   4. You can always keep the query open while working with the form (click on the query from the Shutter Bar on the left side, not from the Property Sheet in this case), though you will need to select values in the combo boxes of the form first to be able to see any data.
   5. Note that the Combo Boxes on a form are directly coded into the Record Source queries to filter for the correct data. The VBA and Design View Record Source must also reflect this.
2. Inside the form’s Design View
   1. If you don’t change it here, the variables may not come through.
   2. On the Property Sheet, ensure that ‘Form’ is the Selection type, then go to Data > Record Source.
3. Inside the VBA (look for **Me.RecordSource** and whatevery variables feed it)
   1. This is the part that will set the Record Source while the user is actually using the form.

The forms *Project History Form* and *Update Project Form* rely solely on a query as their Record Source and do not change the Record Source in the VBA. This is because these forms do not have any sorting features that the user can change.

The form *Add New Project Form* does not utilize a Record Source at all because the form instead relies exclusively on user input and **Dlookup()** functions.

# Main Menu

* *Form\_Load() –* This sub adjusts the Form Footer on the Main Menu for visibility’s sake. I copied it almost all from the hyperlink stated in the comment.
* *Command\_View\_Current\_Projects\_Click() –* If the *Command\_View\_Current\_Projects* button is clicked, this will open the *Open Projects Form*, and close and save the Main Menu. Each subsequent subroutine will do the same task but for a different button and form.
  + *Command\_View\_Consultants\_Projects\_Click()*
  + *Command\_View\_Clients\_Projects\_Click()*
  + *Command\_View\_Analyst\_Projects\_Click()*
  + *Command\_View\_Project\_History\_Click()*
  + *Command\_Update\_Project\_Status\_Click()*
  + *Command\_Add\_New\_Project\_Click()*
* *Command\_Developers\_Click()* – Unhide the Navigation Pane, unlock it, and unhide the ribbon.
* *Command\_Relaunch\_Auto\_Exec() –* Does not appear in this module. When the button *Command\_Relaunch\_Auto\_Exec* is clicked, it runs the macro *AutoExec*.
* *AutoExec* Macro – If the database is not yet trusted, this simply opens the Main Menu so that the user can see the directions to enable the database. Once the database is trusted, this macro calls the function*AutoExec()* which locks/hides the Navigation Pane, opens the Main Menu, and minimizes the Ribbon.
  + The function *AutoExec()* is available in the Module *Start-Up Code* which is in a separate location from the form modules which technically are considered “Class Objects”.
  + Any macro in Access and Excel that is titled *AutoExec()* will automatically run at start-up.

# Open Projects Form

* *Form\_Load() –* Does the following:
  + Sets the initial Record Source for the form including the sort order with the variable *Main\_SQL* and the initial sorting order *Initial\_Sort*.
  + Resets the Caption on each button control by setting it equal to the button’s ‘Tag’.
  + Calls the sub *Option\_Current\_Projects\_AfterUpdate*
  + Resets the focus to the button *Option\_Current\_Projects*
* *Command\_Refresh\_Click() –* When the Refresh command button is clicked, **Requery** the form, then call the *Form\_Load()* sub in order to reset the sorting.
* *Option\_Current\_Projects\_AfterUpdate() –* If someone updates the checkbox *Option\_Current\_Projects*, change the filter to filter for projects that are or are not complete.
* *Main\_Menu\_Button\_Click()* – Close the form and open the Main Menuform.
* *Command\_GoTo\_Update\_Project\_Form\_Click() –* When someone clicks this, this button will gather the ‘Entity’, ‘Client’, ‘Project Name’, and ‘Consultant’ (looked up through a **Dlookup**()), and then use that information to set up the *Update Project Form*.
  + Note that this sub will **Call** the sub *Update\_From\_Other\_Form* IN THE MODULE *[Form\_Update Project Form]*.
* *Command\_GoTo\_Project\_History\_Form\_Click() –* Similary to the above sub, this will gather the information to open and set up the *Project History Form*.
  + Some text manipulation was necessary once I combined the first consultant and second consultant together into a single field.
* *Command\_Sort\_Client\_Click() –* These are the subs that run when a user clicks on a column heading in the form to sort on. Each of these subs will do the following:
  + Designate which variable to sort on.
    - The variable must take the form of however it appears in the query.
    - The **Dim** statements are necessary to carry the value of this variable across subs.
  + **Call** the sub *Button\_Change()*
  + **Call** the sub *Sorting()*
  + And reset the focus to that same button.
    - *Command\_Sort\_Project\_Name\_Click*
    - *Command\_Sort\_Project\_Type\_Click*
    - *Command\_Sort\_CMN\_Click*
    - *Command\_Sort\_Consultants\_Click*
    - *Command\_Sort\_Status\_Click*
    - *Command\_Sort\_Priority\_Click*
    - *Command\_Sort\_Responsible\_Click*
      * For your info, ‘Responsible’ and ‘Ball in Whose Court’ refer to the same field. My bad on the name switch, though it was done due to running out of space on the forms itself with the column headers. It should all work together though.
    - *Command\_Sort\_Last\_Update\_Click*
    - *Command\_Sort\_Last\_Updated\_By\_Click*
    - *Command\_Sort\_Analysts\_Click*
    - *Command\_Sort\_Entity\_Click*
      * The text box and command box for ‘Entity’ are actually hidden in the form meaning the user cannot see them but they appear in Design View.
      * This is because Entity is not an important field for the user to see on this form (it’s discernible through the CMN) but I use the information when the sub *Command\_GoTo\_Update\_Project\_Form\_Click* is run.
    - *Command\_Sort\_Project\_Complete\_Click*
      * There is no sub for the ‘Project Status Notes’ field because Access does not like sorting on memo fields.
* *Button\_Change()* – This sub takes the button that the user clicked on and:
  + Changes the caption on the button to reflect the new sorting order by adding “ – Asc” or “ – Desc” to the button’s Caption, or by resetting the Caption to the button’s Tag.
  + Sets what the now sort order for that particular button will be
  + Runs a **For** loop that calculates the number of buttons currently being sorted on by detecting if the button Captions have “Asc” or “Desc” in them. The number of buttons now being sorted on is called *Total\_Sorts*
* *Sorting() –* This is the sub that actually changes the sorting on the form by changing the “ORDER BY ” in the SQL that makes up the form’s Record Source. I will explain this sub by working backwards:
  + The form’s Record Source is set equal to the variable *New\_RecordSource*
    - *Me.RecordSource = New\_RecordSource*
  + The variable *New\_RecordSource* is set equal to the original variable *Main\_SQL* and the new sorting variable *New\_Sort.* 
    - *New\_RecordSource = Main\_SQL & New\_Sort*
  + The sorting variable *New\_Sort* is set equal to the variable *Inner\_Sort\_Variables* unless there are no columns to sort on in the form, in which case *New\_Sort* is reset to *Initial\_Sort*.
  + *Inner\_Sort\_Variables* is constructed by concatenating the newest sorting variable with the previous sorting variables.
  + There are a lot of text clean-up statements throughout due to the concatenating.
  + READ AND USE THE **DEBUG.PRINT** STATEMENTS IN THIS SUB

# Consultants Form

* The subroutines on the this form below almost exactly mirror those on the *Open Projects Form*, so refer to that section to get a more detailed description of these subs, though there may be slight name and code differences:
  + *Form\_Load()*
  + *Command\_Reload\_Click()*
  + *Main\_Menu\_Button\_Click()*
  + *Option\_Current\_Projects\_AfterUpdate()*
  + *Command\_GoTo\_Update\_Project\_Form\_Click()*
  + *Command\_GoTo\_Project\_History\_Form\_Click()*
  + *Command\_Sort\_Client\_Click()*
  + *Command\_Sort\_Project\_Name\_Click()*
  + *Command\_Sort\_Project\_Type\_Click()*
  + *Command\_Sort\_CMN\_Click()*
  + *Command\_Sort\_Consultants\_Click()*
  + *Command\_Sort\_Status\_Click()*
  + *Command\_Sort\_Priority\_Click()*
  + *Command\_Sort\_Responsible\_Click()*
  + *Command\_Sort\_Last\_Update\_Click()*
  + *Command\_Sort\_Days\_Elapsed\_Click()*
  + *Command\_Sort\_Last\_Updated\_By\_Click()*
  + *Command\_Sort\_Analysts\_Click()*
  + *Command\_Sort\_Entity\_Click()*
  + *Command\_Sort\_Project\_Complete\_Click()*
  + *Button\_Change()*
  + *Sorting()*
* *Combo\_Consultant\_AfterUpdate() –* When someone chooses a new value for the ‘Combo\_Consultant’ button, reset the ‘Opton\_Current\_Projects’ checkbox, **Call** the *Form\_Load* sub to reset the Record Source and sorting, refresh the queries, and place the focus back on ‘Combo\_Consultant’.

# Clients Form

* The subroutines on the this form below almost exactly mirror those on the *Open Projects Form*, so refer to that section to get a more detailed description of these subs, though there may be slight name and code differences:
  + *Form\_Load()*
  + *Command\_Reload\_Click()*
  + *Main\_Menu\_Button\_Click()*
  + *Option\_Current\_Projects\_AfterUpdate()*
  + *Command\_GoTo\_Update\_Project\_Form\_Click()*
  + *Command\_GoTo\_Project\_History\_Form\_Click()*
  + *Command\_Sort\_Client\_Click()*
  + *Command\_Sort\_Project\_Name\_Click()*
  + *Command\_Sort\_Project\_Type\_Click()*
  + *Command\_Sort\_CMN\_Click()*
  + *Command\_Sort\_Consultants\_Click()*
  + *Command\_Sort\_Status\_Click()*
  + *Command\_Sort\_Priority\_Click()*
  + *Command\_Sort\_Responsible\_Click()*
  + *Command\_Sort\_Last\_Update\_Click()*
  + *Command\_Sort\_Days\_Elapsed\_Click()*
  + *Command\_Sort\_Last\_Updated\_By\_Click()*
  + *Command\_Sort\_Analysts\_Click()*
  + *Command\_Sort\_Entity\_Click()*
  + *Command\_Sort\_Project\_Complete\_Click()*
  + *Button\_Change()*
  + *Sorting()*
* *Combo\_Consultant\_AfterUpdate()*
  + Sets the Row Source for the button ‘Combo\_Client’ and resets its value
  + **Call** *Combo\_Client\_AfterUpdate*
* *Combo\_Client\_AfterUpdate()*
  + Reset the ‘Option\_Current\_Projects’ checkbox
  + **Refresh** and **Requery** the form
  + **Call** *Option\_Current\_Projects\_AfterUpdate*
    - Keep in mind this will also call *Form\_Load()* again because if someone changes an option in one of the combo boxes, the form should effectively reset

# Analysts Form

* The subroutines on the this form below almost exactly mirror those on the *Open Projects Form*, so refer to that section to get a more detailed description of these subs, though there may be slight name and code differences:
  + *Form\_Load()*
  + *Command\_Reload\_Click()*
  + *Main\_Menu\_Button\_Click()*
  + *Option\_Current\_Projects\_AfterUpdate()*
  + *Command\_GoTo\_Update\_Project\_Form\_Click()*
  + *Command\_GoTo\_Project\_History\_Form\_Click()*
  + *Command\_Sort\_Client\_Click()*
  + *Command\_Sort\_Project\_Name\_Click()*
  + *Command\_Sort\_Project\_Type\_Click()*
  + *Command\_Sort\_CMN\_Click()*
  + *Command\_Sort\_Consultants\_Click()*
  + *Command\_Sort\_Status\_Click()*
  + *Command\_Sort\_Priority\_Click()*
  + *Command\_Sort\_Responsible\_Click()*
  + *Command\_Sort\_Last\_Update\_Click()*
  + *Command\_Sort\_Days\_Elapsed\_Click()*
  + *Command\_Sort\_Last\_Updated\_By\_Click()*
  + *Command\_Sort\_Analysts\_Click()*
  + *Command\_Sort\_Entity\_Click()*
  + *Command\_Sort\_Project\_Complete\_Click()*
  + *Button\_Change()*
  + *Sorting()*
* *Analyst\_Selection\_Combo\_AfterUpdate()*
  + Reset ‘Opton\_Current\_Projects’
  + **Refresh** and **Requery** the form
  + **Call** *Form\_Load*
  + Set the focus on the combobox ‘Analyst\_Selection\_Combo’

# Project History Form

* *Combo\_Entity\_AfterUpdate()*
  + Sets the Row Source for the combobox ‘Combo\_Consultant’.
  + **Call *Restore\_Defaults***with a parameter of 1.
    - This will reset the value of the combo box ‘Combo\_Consultant’ and will reset the values and row sources of the combo boxes ‘Combo\_Client’ and ‘Combo\_Project\_Name’
* *Combo\_Consultant\_AfterUpdate()*
  + Sets the Row Source for the combobox ‘Combo\_Client’.
  + **Call *Restore\_Defaults***with a parameter of 2.
    - This will reset the value of the combo box ‘Combo\_Client’and will reset the value and row sources of the combo box ‘Combo\_Project\_Name’
* *Combo\_Client\_AfterUpdate()*
  + Sets the Row Source for the combobox ‘Combo\_Project\_Name’.
  + **Call *Restore\_Defaults***with a parameter of 3.
    - This will reset the value of the combo box ‘Combo\_Project\_Name’
* *Combo\_Project\_Name\_AfterUpdate()*
  + **Refresh** and **Requery** the form
  + No data will appear until all four combo boxes are filled in because the Record Source for the form, *qryProjectHistory\_Combos*, requires each of the boxes.
  + See the query to understand.
* *Restore\_Defaults(Stage)*
  + Resets the appropriate combo box according to how many the user has filled in so far. See above.
* *History\_From\_Other\_Form() –* When someone has clicked the ‘H’ button on another form, that will auto-fill in the values for ‘Combo\_Entity’, ‘Combo\_Consultant’, ‘Combo\_Client’, and ‘Combo\_Project\_Name’. The code for that is on the other form.
  + When that other subroutine calls *History\_From\_Other\_Form()*, this sub simply will
  + **Call** *Combo\_Project\_Name\_AfterUpdate* in order to **Refresh** and **Requery** the form
  + Set the Row Sources for ‘Combo\_Consultant’, ‘Combo\_Client’, and ‘Combo\_Project\_Name’.
* The subroutines below almost exactly mirror those on the *Open Projects Form*, so refer to that section to get a more detailed description of these subs, though there may be slight name and code differences:
  + *Command\_GoTo\_Update\_Project\_Form\_Click()*
  + *Command\_Reload\_Click()*
  + *Main\_Menu\_Button\_Click()*

# Update Project Form

This form makes use of various controls’ Tag attributes for the purpose of requiring the user to fill in certain fields before updating a project’s status. These are:

* “Required” – The user must have input information into these fields.
  + ‘Combo\_Entity’
  + ‘Combo\_Consultant’
  + ‘Combo\_Client’
  + ‘Combo\_Project\_Name’
  + ‘Upd\_Status’
  + ‘Upd\_Ball\_In\_Whose\_Court’
  + ‘Upd\_Initials’
  + ‘Upd\_Priority’
  + ‘Upd\_Date’
* The following subroutines mirror those in the module for the *Project History Form* with differences due mainly to the other controls throughout the form:
  + *Combo\_Entity\_AfterUpdate()*
  + *Combo\_Consultant\_AfterUpdate()*
  + *Combo\_Client\_AfterUpdate()*
  + *Restore\_Defaults(Stage)*
* *Combo\_Project\_Name\_AfterUpdate() –* This subroutine does the following:
  + Defines the variable *filtering\_text*
  + Resets the Value of all Text Boxes and Combo Boxes that are in the Detail section of the form.
  + Uses *filtering\_text* in a series of **Dlookup()** functions to lookup all of the values for the project that the user will be updating.
  + These looked-up values all come from *qryOpenProjectsByConsultantWithStatus*
  + Sets the Row Source for the combo box ‘Upd\_Status’ depending upon the project’s ‘Project Type’.
* *Command\_Update\_Project\_Status\_Click() –* This is subroutine that actually adds the new status once the user clicks ‘Command\_Update\_Project\_Status’:
  + First, certain combo boxes have the Tag “Required”. If any of these are blank, have an apostrophe, or have the words “Select” in them then throw up a **MsgBox**.
  + Then if everything’s ok, pop up a **MsgBox** that asks if the user truly wants to update this project with the information they wrote.
  + Run a series of **DLookup()** functions to supplement what the user has input in the form in order to be able to add the status to the *Projects* table.
  + Depending upon what the user clicks on in the ‘Save\_Button’, create and execute an SQL string to add the status to the *Projects* table.
  + **Refresh** and **Requery**
  + **Call *Restore\_Defaults*(0)** to restore the default values for all text in combo and text boxes.
* *Update\_From\_Other\_Form() -* When someone has clicked the ‘!’ button on another form, that will auto-fill in the values for ‘Combo\_Entity’, ‘Combo\_Consultant’, ‘Combo\_Client’, and ‘Combo\_Project\_Name’. The code for that is on the other form.
  + When that other subroutine calls *History\_From\_Other\_Form()*, this sub simply will
  + **Call** *Combo\_Project\_Name\_AfterUpdate* in order to **Refresh** and **Requery** the form
  + Set the Row Sources for ‘Combo\_Consultant’, ‘Combo\_Client’, and ‘Combo\_Project\_Name’.
* The subroutines below almost exactly mirror those on the *Open Projects Form*, so refer to that section to get a more detailed description of these subs, though there may be slight name and code differences:
  + *Command\_GoTo\_Project\_History\_Form\_Click()*
  + *Main\_Menu\_Button\_Click()*

# Add New Project Form

This form makes use of various controls’ Tag attributes for the purpose of making some visible/invisible and editable/not editable principally depending upon if the user selects “New Client” or “Existing Client” from the combo ‘Combo\_New\_Existing\_Client’. Here are the following Tags used:

* “Don’t Change” – Used to denote that the values for this control should not be changed by the code, though by the user is ok.
  + ‘Combo\_New\_Existing\_Client’
* “Existing” – Will appear and matter only if the user has selected to add a new project to an “Existing Client”.
  + ‘Label\_Existing\_Client\_Name’
  + ‘Combo\_Existing\_Client\_Name’
* “New” – Will appear and matter only if the user has selected to add a new project to an “New Client”.
  + ‘Text\_New\_Client\_Name’
  + ‘Label\_New\_Client\_Name’
* “Optional” – Can be left blank by the user and will not stop the project from being added to the database (i.e. all the user-updated fields that don’t have a “\*” in the label).
  + ‘Combo\_Second\_Consultant’
  + ‘Combo\_First\_Analyst’
  + ‘Combo\_Second\_Analyst’
  + ‘Combo\_Project\_Status\_Notes’

As to the subroutines:

* *Form\_Load() –* 
  + **Call *Client\_Number\_Toggling*** for the purpose of dealing with the Visibility of certain Visible controls on the form.
* *Combo\_New\_Existing\_Client\_AfterUpdate() –*
  + Sets the value to Null for all controls except for itself, ‘Combo\_New\_Existing\_Client’
  + **Call *Reset\_to\_Default(1)***to reset the value to “Select” for ‘Combo\_Entity’ and ‘Combo\_Existing\_Client’.
  + **Call *Client\_Number\_Toggling***to control that control’s visibility
* *Combo\_Entity\_AfterUpdate() –* 
  + Sets the RowSource for ‘Combo\_Existing\_Client\_Name’
  + Similarly to *Combo\_New\_Existing\_Client\_AfterUpdate()*, sets the value for certain controls to Null and others to “Select…" via **Call *Client\_Number\_Toggling*** and **Call *Reset\_to\_Default(2)***
* *Combo\_Existing\_Client\_Name\_AfterUpdate() –* This button and its label are hidden (i.e. not visibile to the user unless the user has chosen “Existing Client” in ‘Combo\_New\_Existing\_Client’. You can always see any hidden buttons in Design View.
  + Sets the value for certain controls to Null and others to “Select…" via **Call *Client\_Number\_Toggling*** and **Call** ***Reset\_to\_Default(3)***
  + Uses **Dlookup()** to lookup and fill in the ‘Client Number’ for the textbox ‘Text\_Existing\_Client\_Number’.
* *Client\_Number\_Toggling() –* This sub determines the visibility and appearance of ‘Label\_Existing\_Client\_Name’, ‘Combo\_Existing\_Client\_Name’, ‘Label\_Client\_Number’ and ‘Text\_Existing\_Client\_Number’, ‘Label\_New\_Client\_Name’, and ‘Text\_New\_Client\_Name’ depending upon the selection in ‘Combo\_New\_Existing\_Client’ and each control’s Tag attributes.
  + If ‘Client\_New\_Existing\_Client’ has the value of “Existing Client” or it’s blank because the form just opened:
    - The “Existing” Tag controls are made visible and the “New” ones are invisible
    - The ‘Label\_Client\_Number’ and ‘Text\_Existing\_Client\_Number’ are greyed out and made ineditable.
  + If ‘Client\_New\_Existing\_Client’ has the value of “New Client”
    - The “New” Tag controls are made visible and the “Existing” ones are invisible
    - The ‘Label\_Client\_Number’ and ‘Text\_Existing\_Client\_Number’ now have a white background and are editable.
  + If ‘Client\_New\_Existing\_Client’ has the value of “New Client” and “Combo\_Entity” = “EASI”
    - The ‘Label\_Client\_Number’ and ‘Text\_Existing\_Client\_Number’ are greyed out and made ineditable because we already know the Client\_Number will be 1310.
* *Command\_Add\_New\_Project\_Click() –* This sub runs a bunch of code that determines what happens when the user clicks ‘Command\_Add\_New\_Project’.
  + First and foremost this sub has an Error Handler with the code **On Error GoTo *Error\_Handler****.* This is important because other subs will generate Errors that must be carried over to this sub so that the remainder of the code in this module is bypassed.
  + If any of the following conditions are true the sub will raise a **MsgBox** with a message about the error and **Exit Sub** to end code execution.
    - Non-Optional combo boxes are blank or have “[Select…]” as their value meaning the user didn’t change them.
    - Any of the combo boxes have an apostrophe in them “’”. Apostrophes mess with hard-coded SQL text, so they should not be used anywhere as this will cause problems.
    - FYI, **Exit Sub** is the code that stops the code execution. **MsgBox** merely throws up a message to the user.
  + **Call *Add\_Project***, which is the sub to actually add the new project.
    - If an **Error** occurs within the sub *Add\_Project*, that error is carried forward and the code **On Error GoTo *Error\_Handler*** will jump to *Error\_Handler* in this same module and bypass the code in between those two points.
  + If the code execution has gotten to this point, then the project has been added successfully, so pop up a **MsgBox** indicating as such.
  + **Call*****Combo\_New\_Existing\_Client\_AfterUpdate***to reset the form.
  + *Error\_Handler* – This section just needed to be populated for the custom errors to work. It uses **Case.**
* *Premature\_Save() –* Display a message box if the user has tried to add the new project before filling out information in all required fields.
  + Keep in mind though that this sub merely throws up a message box, and the **Exit Sub** in the **If** blocks in the subroutine *Command\_Add\_New\_Project\_Click()* are what prevents further code execution.
* *Reset\_to\_Default(Stage) –* As in the same sub in other modules, reset combo boxes to their correct values depending upon what was the last combo box filled in.
* *Add\_Project() –* This sub actually adds the project to the database using SQL text, but it also has a lot of code that raises custom errors that will prevent further code execution.
  + Create a **MsgBox** that asks if the user wants to save this project, and feed them the information they put in.
    - If the value of ‘Save\_Button’ = 7, which corresponds to “No”, then raise custom error 667 and **Exit** this sub. The error will stop code execution in the sub *Command\_Add\_New\_Project\_Click().*
  + If the project being added is to an existing client:
    - Execute one sql statement to add the project.
      * The sql statement can fail with custom error 668 if the ‘Entity’, ‘Client’, and ‘Project Name’ already exist.
      * The sql requires the code **CurrentDb.OpenRecordset([*object or query*]).EOF**
  + If the project being added is to a new client:
    - Execute one sql statement to add the new client to the *Clients* table
      * The sql statement can fail if the ‘Client’ and the ‘Client Number’ already exist (for NT Lakis clients) or if the ‘Client’ exists (for EASI clients).
    - Execute another sql statement to add the project to the *Projects* table
* *Main\_Menu\_Button\_Click() –* Close this form and open the *Main Menu*
* *Upd\_Project\_Type\_AfterUpdate() –* Limit the combo box ‘Upd\_Status’ depending upon the ‘Project Type’ that the user has selected in the field ‘Upd\_Project\_Type’.

# Troubleshooting

## General

1. If nothing works, have you enabled the database? You may need to click ‘Reload Application’ afterwards.
2. If project updates aren’t appearing, have you clicked ‘Reload Form’ on the Form you are working on?
3. The best troubleshooting option is to close the database and re-open it again.
4. If the issue persists despite utilizing the below information, document the issue and then reach out to James and Christina for assistance.

## Forms

1. If a button is not working, have you linked it to the appropriate code?
   1. Go to the form’s Design View
   2. Click on the appropriate button
   3. Go to the Property Sheet > Event tab
   4. Ensure that ‘On Click’, ‘After Update’, ‘On Load’ or whichever option has “[Event Procedure]” written, and that the name of the Subroutine with your code is titled [*Your\_Button*].*AfterUpdate*()or the like (the VBA should auto-specify the name of the sub).
2. Do the form, its relevant query, and/or the VBA in that form’s module conflict in some way, particularly regarding the Record Source? Check the section Introduction > Forms for more information.

## Code

1. If there is an Access error code, read it and google it.
2. Read through comments I have left in the code.
3. Uncomment **Debug.Print** statements or write your own to see the value of various objects and their attributes
4. In combination with #2, insert **Breakpoints** by clicking on the grey bar directly to the left of the line of code that you want to insert a break at.
   1. Additionally, you can then Right-Click and choose ‘Set Next Statement’ to select what line of code will run next.
5. Have you properly surrounded quotation marks in VBA hard-coded SQL text with an additional set of quotation marks?
   1. I.e. if in a query you could write
   2. IIf([Second Analyst] Is Null,[First Analyst],[First Analyst] & ", " & [Second Analyst])
   3. However in VBA, if you want to save that to a variable named ‘Variable’, you would write
   4. **Variable = “IIf([Second Analyst] Is Null,[First Analyst],[First Analyst] & "", "" & [Second Analyst])”**
   5. SQL text must also end with “;” and must not have any spacing issues in it.
6. However if you are trying to use the value of a control, such as *Combo\_Entity.Value*, as text in an SQL string you must surround it in single-quotes. Example:
   1. **strSQL = ...other text… & "VALUES ('" & Combo\_Entity.Value & "', '" & Lookup\_First\_Consultant & "');" & … other text …**
   2. However be sure to not put these single quotation marks around a value that you want to be used as a number, such as *Upd\_Project\_Complete.Value*
      1. Access codes a checkbox as either 1 or 0.
7. Do you need to run **Me.Refresh** and **Me.Requery** statements in the code? Try them anyway as they don’t hurt.
   1. You’ll particularly need this after having any run any **Dlookup** statements, changing the Record Source in a form, or changing the Row Source in a combo box.
8. Are there any issues with apostrophes being coded into the data? Or the word “Select”?
   1. I.e. having a ‘Consultant’ who is named “His Selected’s Excellency” would be a problem
   2. The problem with “Select” is that I assume that if “Select” is present in a combo box that means that the user has not changed the default “[Select a Status]” value
9. All subroutines must end with **End Sub**, all **For** and **For Each** loops must have a **Next** statement, all **If** statements must have a **Then** statement and must end with **End If.** 
   1. TIP: Keep your code very neat and tidy by using the same indentation style that I have, clearly naming variables and controls, using comments, and using **Debug.Print** statements.

# Glossary

This glossary gives a definition and where appropriate hyperlinks for statements or objects that I believe are useful to know. Some statements I copied from the internet to solve a particular problem and I am not sure what they exactly do; I have omitted these from the glossary.

* **“& \_”** – Wrap a string of text to the next line in VBA. Very useful for hard-coding SQL statements that you want to keep human readable.
* Breakpoints– These are used to stop code execution. Click on the bar on the left side of the VBA module to insert a breakpoint at that particular line. [Link](https://www.techonthenet.com/access/tutorials/vbadebug2010/debug01.php).
  + Use in conjunction with **Debug.Print** statements and ‘Set Next Statement’ accessed via right-click.
* **Call** [*object name*]– Call a subroutine from another subroutine. After the child subroutine has been run, code will resume at the same place in the parent subroutine. [Link](https://docs.microsoft.com/en-us/office/vba/language/concepts/getting-started/calling-sub-and-function-procedures).
* **Case –** Similar to Switch() in Microsoft Access. [Link](https://docs.microsoft.com/en-us/office/vba/language/reference/user-interface-help/select-case-statement).
* **Chr(*number*)** – Add a special VBA character depending upon the parameter.
  + Chr(13) adds a new-line character. Very useful for keeping hard-coded SQL statements human-readable.
* **Debug.Print** – Print the value of whatever object and attributes are stated.
* **Dim *variable* As *variable-type*** – Define variables that can exist outside of subroutines and modules. [Link](https://www.thespreadsheetguru.com/blog/2014/2/19/what-does-dim-mean-when-coding-vba-macros).
* **Dlookup() –** Similar to Vlookup in Excel, used to get a single value from a query or table. [Link](https://support.office.com/en-ie/article/dlookup-function-8896cb03-e31f-45d1-86db-bed10dca5937).
* **DoCmd.Close** – Close a form. You must specify whether or not to save the form as given by *acSaveYes* or *acSaveNo*. For your sanity’s sake, do the former.
* **DoCmd.OpenForm** – Open a form.
* **CurrentDb –** Deals with items related to the Current Database. Has many attributes, such as the following ones used:
  + **CurrentDb.OpenRecordset([*object or query*]).EOF –** Ask if the cursor is currently at the End Of File. Useful for determining if a new query is empty. [Link](https://docs.microsoft.com/en-us/office/client-developer/access/desktop-database-reference/recordset-eof-property-dao).
  + **CurrentDb.Execute –** Execute a custom query created in VBA. [Link](https://docs.microsoft.com/en-us/office/client-developer/access/desktop-database-reference/database-execute-method-dao).
* **Err** – The Error object in Microsoft Access. Programmers can create their own custom errors to reflect custom conditions. [Link](https://docs.microsoft.com/en-us/office/vba/language/reference/user-interface-help/err-object).
  + **Err.Raise** – Create a custom error.
  + **Err.Number** – Access that custom error.
  + **Err.Description** – Access the text of that custom error.
  + **On Error GoTo [*linelabel*]** – Allows for error handling.
    - Requires line labels. [Link](https://docs.microsoft.com/en-us/dotnet/visual-basic/programming-guide/program-structure/how-to-label-statements).
* Macro – A section of code pre-written by Microsoft that is used to accomplish a particular, common task. Any macro can be converted to VBA, but most VBA cannot be converted to macros. [Link](http://www.databasedev.co.uk/macros-in-access.html).
* **Me** – Refers to the form that the module is attached to -- [Link](https://docs.microsoft.com/en-us/office/vba/language/reference/user-interface-help/me-keyword). It has several important attributes, each with their own attributes:
  + **Me.ActiveControl** – The active button that the user has just interacted with.
  + **Me.Controls** – Refers to all controls in a form. Particularly useful for setting properties/attributes of all controls, such as the caption, color, width, etc. using a For loop.
    - Be careful though, because different controls will have different attributes, as a Combo Box and a Check Box have different properties.
    - [Link](https://docs.microsoft.com/en-us/office/vba/api/access.controls)
  + **Me.Refresh** and **Me.Requery** – Refresh the form and requery the Record Source, respectively. These are particularly needed after changing the Record Source, using **Dlookup()** functions, or executing any sql statements.
* **MsgBox() –** Pop up a dialogue box for the user to see. If you create save that information to a variable, you can then determine what happens depending upon the user’s choice. [Link](https://support.office.com/en-ie/article/msgbox-function-e23511c8-4157-4e09-93a6-ba9f1749f4c0)
  + I.e. **Save\_Button = MsgBox([Prompt], [Buttons], [Title])** will save the result of what *Button* the user selects to the variable ‘Save\_Button’.
  + The *Buttons* parameter is a series of numbers that are added to gether. I.e. 68 = 64 + 4, and “64” means the Button *vbCritical* while “4” means the Button *vbYesNo*. In VBA, hit F1 and then search “MsgBox” to see how this is done.
* Module – Where you write VBA code. Accessible via Alt + F11.
  + [General Link](https://www.techonthenet.com/access/modules/index.php)
  + [Module vs. Class Module Link](https://excelmacromastery.com/vba-class-modules/)
* Scope – [See here](https://docs.microsoft.com/en-us/office/vba/language/concepts/getting-started/understanding-scope-and-visibility) for more information on scope. It applies to variables, subroutines, and functions.
* Subroutine – Collection of VBA code. [Link](https://docs.microsoft.com/en-us/office/vba/language/concepts/getting-started/calling-sub-and-function-procedures).
  + **End Sub** – Required in order to end each subroutine.
  + **Exit Sub** – Used to leave a subroutine without further code execution. Very good to use with **If** conditions and **Err.Raise**.
* Tag – A useful way of grouping controls on a form, available in the Form’s Design View > Property Sheet > Other > Tag. [Link](https://docs.microsoft.com/en-us/office/vba/api/access.form.tag).