This is a companion document to *Access VBA Techniques*.

Contents:

Getting Started3	
Macros & Modules3	
Writing Code4	
Object References4	
Debugging9	
Presentation of User Processes9	
Classification Scheme10	
Field-level Security10	
Application Security10	
Initial Value of Code in Data Table12	
Collect Statistics12	
Reference to File on File Server12	
When Date Function Returns 12/31/189913	
Dynamic Array13	
Data Types13	
Memo Fields14	
Empty Fields14	
Network Performance16	
Multi-User Access	
Data Type Mismatch16	
Arrays17	
ASCII Non-printing Carriage Control	
Characters18	
Units of Measure18	
Typography in Forms and Reports19	
31 8 1 3	
Using ActiveX Controls19	
Using ActiveX Controls	
Applications in Access21	
Applications in Access21 Startup Options21	
Applications in Access 21 Startup Options 21 Hiding Objects 22	
Applications in Access 21 Startup Options 21 Hiding Objects 22 AutoExec Macro 23	
Applications in Access 21 Startup Options 21 Hiding Objects 22 AutoExec Macro 23 Recovery 23	
Applications in Access 21 Startup Options 21 Hiding Objects 22 AutoExec Macro 23 Recovery 23 Suggestions 23	
Applications in Access 21 Startup Options 21 Hiding Objects 22 AutoExec Macro 23 Recovery 23 Suggestions 23 Bars: Menu Bars, Toolbars, and Shortcut Menus 24	
Applications in Access 21 Startup Options 21 Hiding Objects 22 AutoExec Macro 23 Recovery 23 Suggestions 23 Bars: Menu Bars, Toolbars, and Shortcut 24 Expressions and Functions 26	
Applications in Access 21 Startup Options 21 Hiding Objects 22 AutoExec Macro 23 Recovery 23 Suggestions 23 Bars: Menu Bars, Toolbars, and Shortcut Menus 24 Expressions and Functions 26 General 26	
Applications in Access 21 Startup Options 21 Hiding Objects 22 AutoExec Macro 23 Recovery 23 Suggestions 23 Bars: Menu Bars, Toolbars, and Shortcut Menus Menus 24 Expressions and Functions 26 General 26 Logical Operators 27	
Applications in Access 21 Startup Options 21 Hiding Objects 22 AutoExec Macro 23 Recovery 23 Suggestions 23 Bars: Menu Bars, Toolbars, and Shortcut Menus Menus 24 Expressions and Functions 26 General 26 Logical Operators 27 Comparison Operators 27	
Applications in Access 21 Startup Options 21 Hiding Objects 22 AutoExec Macro 23 Recovery 23 Suggestions 23 Bars: Menu Bars, Toolbars, and Shortcut Menus Menus 24 Expressions and Functions 26 General 26 Logical Operators 27 Comparison Operators 27 Date Functions 27	
Applications in Access 21 Startup Options 21 Hiding Objects 22 AutoExec Macro 23 Recovery 23 Suggestions 23 Bars: Menu Bars, Toolbars, and Shortcut Menus Menus 24 Expressions and Functions 26 General 26 Logical Operators 27 Comparison Operators 27 Date Functions 27 Arithmetic Functions 28	
Applications in Access 21 Startup Options 21 Hiding Objects 22 AutoExec Macro 23 Recovery 23 Suggestions 23 Bars: Menu Bars, Toolbars, and Shortcut Menus Menus 24 Expressions and Functions 26 General 26 Logical Operators 27 Comparison Operators 27 Date Functions 27 Arithmetic Functions 28 Aggregate Functions 28	
Applications in Access 21 Startup Options 21 Hiding Objects 22 AutoExec Macro 23 Recovery 23 Suggestions 23 Bars: Menu Bars, Toolbars, and Shortcut Menus Menus 24 Expressions and Functions 26 General 26 Logical Operators 27 Comparison Operators 27 Date Functions 27 Arithmetic Functions 28 Aggregate Functions 28 Lookup Function 29	
Applications in Access 21 Startup Options 21 Hiding Objects 22 AutoExec Macro 23 Recovery 23 Suggestions 23 Bars: Menu Bars, Toolbars, and Shortcut Menus Menus 24 Expressions and Functions 26 General 26 Logical Operators 27 Comparison Operators 27 Date Functions 27 Arithmetic Functions 28 Aggregate Functions 28 Lookup Function 29 Format Functions 30	
Applications in Access 21 Startup Options 21 Hiding Objects 22 AutoExec Macro 23 Recovery 23 Suggestions 23 Bars: Menu Bars, Toolbars, and Shortcut Menus Menus 24 Expressions and Functions 26 General 26 Logical Operators 27 Comparison Operators 27 Date Functions 27 Arithmetic Functions 28 Aggregate Functions 28 Lookup Function 29 Format Functions 30 Get Length of a Text/Memo Field 30	
Applications in Access 21 Startup Options 21 Hiding Objects 22 AutoExec Macro 23 Recovery 23 Suggestions 23 Bars: Menu Bars, Toolbars, and Shortcut Menus Menus 24 Expressions and Functions 26 General 26 Logical Operators 27 Comparison Operators 27 Date Functions 27 Arithmetic Functions 28 Aggregate Functions 28 Lookup Function 29 Format Functions 30 Get Length of a Text/Memo Field 30 Return One of Two Values 31	
Applications in Access 21 Startup Options 21 Hiding Objects 22 AutoExec Macro 23 Recovery 23 Suggestions 23 Bars: Menu Bars, Toolbars, and Shortcut Menus Menus 24 Expressions and Functions 26 General 26 Logical Operators 27 Comparison Operators 27 Date Functions 28 Aggregate Functions 28 Lookup Function 29 Format Functions 30 Get Length of a Text/Memo Field 30 Return One of Two Values 31 Carriage Control Functions 31	
Applications in Access 21 Startup Options 21 Hiding Objects 22 AutoExec Macro 23 Recovery 23 Suggestions 23 Bars: Menu Bars, Toolbars, and Shortcut Menus Menus 24 Expressions and Functions 26 General 26 Logical Operators 27 Comparison Operators 27 Date Functions 27 Arithmetic Functions 28 Aggregate Functions 28 Lookup Function 29 Format Functions 30 Get Length of a Text/Memo Field 30 Return One of Two Values 31	

TypeName Function	33
Constants	34
CallByName Function	34
Formatting	35
Format Function	35
Format Property	35
Format Property for Text and Memo Data	
<i>Types</i>	35
Format Property for Dates	
Function Int	
Table Stuff	37
Relational Database	37
Primary Key(s)	
Indexes	
Empty a Table	
Import Table Specification	
Import Excel Worksheet	
Link Table to DB2 Table	
Link Table to SQL Server	
Edit Data While Input	
Default Field Labels in Forms & Reports	
Table Definition: Lookup Values with Value	39
List	39
Table Definition: Lookup Values with Query	
Define Table Fields with Formatting	
Automatically Set Last Change Date in a	00
Record	39
Hyperlinks	
Hyperlinks in a Table Field: Open Any File	
Resetting Autonumbers to Zero	
Fixing a Table with an Autonumber Key	
Making Yes/No Type Fields Appear with	11
Checkbox	41
Form Stuff	
Types of Forms	
V1 V	
Application Forms	
Form-Process Interaction	
Useful Form Properties	
Change Form Properties	
Form Control Scrollbar	
Refresh Contents of Form	
Refer to Form Controls	
Is Form Open	
Multiple Instances of Forms	
Approach to Editing a Table With a Form	
,,	
Forms for Parent-Child Tables	52
Forms for Parent-Child Tables Navigation on Single Record Form	52 52
Forms for Parent-Child Tables	52 52 55

Protect Form Field From Update56
Miscellaneous Form Control Properties56
Command Buttons56
Option Buttons57
Memo Field in Text Box57
<i>Combo Box57</i>
<i>List Box</i>
Use List Box to Restrict Values to Lookup
<i>Table61</i>
Populate List Box with Query Result61
Populate List Box Containing Names of
Queries62
Scroll Down List Box to Show Pre-Selected
<i>Row</i> 62
Tab Control 63
Setting Field Value70
Put Record Count on Form71
Edit Data72
Find Individual Record on Form72
Block Record Additions/Deletions on Form72
<i>Print a Form72</i>
Initiate New Query/Report73
Update a Table From a Form Based on a
,
Update a Table From a Form Based on a
Update a Table From a Form Based on a Query73
Update a Table From a Form Based on aQuery
Update a Table From a Form Based on a 73 Query
Update a Table From a Form Based on a Query
Update a Table From a Form Based on a Query
Update a Table From a Form Based on a73Query73Edit Table with Two Forms73Subforms75Editing Data in a Child SubForm with Standard Navigation Controls76Link Subform to Main Form77
Update a Table From a Form Based on a Query
Update a Table From a Form Based on a Query
Update a Table From a Form Based on a Query
Update a Table From a Form Based on a Query
Update a Table From a Form Based on a Query
Update a Table From a Form Based on a Query
Update a Table From a Form Based on a Query
Update a Table From a Form Based on a Query
Update a Table From a Form Based on a Query
Update a Table From a Form Based on a Query
Update a Table From a Form Based on a Query
Update a Table From a Form Based on a Query

Form Opens Blank	87
Slow Opening Form	87
Form Suspends Itself	88
Controlling One Form From Another	88
Report Stuff	
Field Control Source	90
Grouping Records	90
Memo Fields	
Put Record Count on Report	
Adjusting Size of Controls to Data	92
Multi-Column Report	92
Presenting Data From Multiple Sources	92
Subreports	93
Child SubReport	93
Filter Report Data	93
Using Date Selection Criteria for a Report	93
Use Form to Select Data For Report	96
Using Date Selection Criteria for a Report	
with a Parameterized Query	97
Report Events	98
Macros	00
Wiacius	90
Queries	99
Queries Parameter Queries	99 99
Queries	99 99 1 00
Queries	99 99 100
Queries	99 99 100 100
Queries Parameter Queries SQL 1 SELECT 1 SELECT INTO 1 JOIN 1	99 99 100 100 101
Queries Parameter Queries SQL 1 SELECT 2 SELECT INTO 3 JOIN 3 UNION 3	99 99 100 100 101 101
Queries Parameter Queries SQL 1 SELECT 5 SELECT INTO 5 JOIN 1 UNION 1 UPDATE 5	99 99 100 100 101 101 102 103
Queries Parameter Queries SQL 1 SELECT 5 SELECT INTO 5 JOIN 1 UNION 1 UPDATE 1 INSERT 1	99 99 100 100 101 101 102 103 105
Queries Parameter Queries SQL 1 SELECT 5 SELECT INTO 1 JOIN 1 UNION 1 UPDATE 1 INSERT 1 DELETE 1	99 99 100 100 101 101 102 103 105 105
Queries Parameter Queries SQL 1 SELECT 1 SELECT INTO 1 JOIN 1 UNION 1 UPDATE 1 INSERT 1 DELETE 1 TRANSFORM 1	99 99 100 100 101 101 102 103 105 105
Queries Parameter Queries SQL 1 SELECT 5 SELECT INTO 1 JOIN 1 UNION 1 UPDATE 1 INSERT 1 DELETE 1 TRANSFORM 1 Correlation Names 1	99 99 100 100 101 101 102 103 105 105 105
Queries Parameter Queries SQL 1 SELECT 2 SELECT INTO 3 JOIN 4 UNION 4 UPDATE 5 INSERT 6 DELETE 7 TRANSFORM 6 Correlation Names 6 Using Comparisons in SQL 6	99 99 100 100 101 102 103 105 105 107 107
Queries Parameter Queries SQL 1 SELECT 1 SELECT INTO 1 JOIN 1 UNION 1 UPDATE 1 INSERT 1 DELETE 1 TRANSFORM 1 Correlation Names 1 Using Comparisons in SQL 1 Predicates 1	99 99 100 100 101 101 102 103 105 105 107 107 108
Queries Parameter Queries SQL 1 SELECT 5 SELECT INTO 5 JOIN 1 UNION 1 UPDATE 1 INSERT 1 DELETE 1 TRANSFORM 2 Correlation Names 1 Using Comparisons in SQL 1 Predicates 5 SELECT Predicates 1	99 99 100 100 101 102 103 105 105 107 107 108 110
Queries Parameter Queries SQL 1 SELECT 1 SELECT INTO 1 JOIN 1 UNION 1 UPDATE 1 INSERT 1 DELETE 1 TRANSFORM 2 Correlation Names 1 Using Comparisons in SQL 1 Predicates 3 SELECT Predicates 5 Subquery-Subselect 1	99 100 100 101 101 102 103 105 105 107 107 108 110
Queries Parameter Queries SQL 1 SELECT 5 SELECT INTO 1 JOIN 1 UNION 1 UPDATE 1 INSERT 1 DELETE 1 TRANSFORM 2 Correlation Names 1 Using Comparisons in SQL 2 Predicates 3 SELECT Predicates 3 Subquery-Subselect 4 Miscellaneous Query Techniques 3	99 100 100 101 101 102 103 105 105 107 107 108 110 110
Queries Parameter Queries SQL 1 SELECT 2 SELECT INTO 3 JOIN 4 UNION 4 UPDATE 5 INSERT 5 DELETE 7 TRANSFORM 6 Correlation Names 6 Using Comparisons in SQL 7 Predicates 5 SELECT Predicates 5 Subquery-Subselect 6 Miscellaneous Query Techniques 6 Character Replacement in SELECT 6	99 100 100 101 101 102 103 105 105 107 108 110 111 117
Queries Parameter Queries SQL 1 SELECT 5 SELECT INTO 1 JOIN 1 UNION 1 UPDATE 1 INSERT 1 DELETE 1 TRANSFORM 2 Correlation Names 1 Using Comparisons in SQL 2 Predicates 3 SELECT Predicates 3 Subquery-Subselect 4 Miscellaneous Query Techniques 3	99 99 100 100 101 102 103 105 105 107 107 108 110 111 117 118

Getting Started

In response to a request for an essay on using Access to solve some problem:

Data lives in *tables*. Design your tables with great care. Use SQL *queries* to manipulate data. Define user transactions. Use *forms* to facilitate user data entry, to initiate *reports*, and to provide access to user transactions (Microsoft calls this last a "switchboard", but I think they got their metaphors crossed). Use *reports* to present data in custom arrangements.

Plan your application: decide what Access components are needed and how they will be used. Work out the data flow and process details. When you have a complicated process, break it into several small simple and auditable steps. It can be helpful to use temporary tables to collect data for a form or report. You can chain queries together by having one dependent on the previous one. You can do a lot with queries and tables, but by no means everything. For the rest you will have to use code.

Work slowly and keep notes—it is easy to forget what you have already done and not done. It's also easy to create components that eventually go unused; delete them. Use the Documentor (menu <u>Tools, Analyze, Documentor</u>) to report the components of your database. Use the documentation code in this document to get more detail in a more usable format.

If you have trouble getting a form to work the way you want, it may be that your data model is wrong. This is especially true when a form handles more than one table.

If you are creating a multi-user application:

- Identify user roles and activities and use them to refine your user interface. Provide for user administration (file access and user-level security) and data administration (compact, backup, and restore the database).
- Build in security.
- Include features that will make it possible for you to troubleshoot the application. A user activity log makes a useful record of who did what when. A statistics table allows you to keep key counts in one place and see how they change over time. An error table is populated with custom error handling and lets you review the details on errors that occurred (especially useful when users experience errors and report them to you incompletely).
- Provide for ongoing bug fixes and development. The easiest way to do this is to have two databases: DEVelopment and PRODuction. You change and create components in DEV, test them thoroughly, them copy them to PROD.

Macros & Modules

There are two ways to implement "code": macros and modules. Macros have a limited set of possible actions; they are developed by the Macro Editor. Modules contain subroutines and functions coded in Visual Basic.

I've come to prefer using modules exclusively. Macros are appealing to the novice, but they are limited in what they can do. More importantly, they should not be used for applications used by people other than yourself as they provide security problems. Just learn Visual Basic!

Visual Basic is an object-oriented programming language. It is based on objects (such as a table) which have methods (such as **open**) and properties (such as **name**). You make things happen by reading and setting object properties and executing object methods. It's a pain but what can you do? In order to be successful, you must learn the object model. It contains the objects and their hierarchical relationships with each other as well as their properties and methods.

There are several object models which you may end up using: Access, Visual Basic, Visual Basic for Applications, DAO, and ADO. Enjoy!

The main problem with object models is that the object relationships are forced into hierarchies. And hierarchies do not always accurately reflect the true logical relationships. (This is why IBM's mainframe database IMS was replaced by DB2: the hierarchical model was not as useful as the relational model.)

Writing Code

Generally code is typed directly into a module in the Visual Basic Editor (VBE). When you do it that way, the VBE can detect which objects you are using and establish the necessary type library references.

When code is pasted into a module, VBE does not detect the objects and consequently, a reference will not be automatically set to the necessary type library. You will know a reference is missing when you compile or run the code: an error message will appear, often "undefined user type." The solution is to manually select the necessary reference.

See Set References on page Error! Bookmark not defined..

You will need at least one good reference book on Visual Basic in order to learn it and use it efficiently.

BE CAREFUL to use correct " and ' characters. If you copy code from Word, they may be wrong.

Object References

You must reference an object before you can use it¹. Establishing an object reference varies by type of object:

- Access objects: these include tables, queries, forms, and reports.
- Class objects: these correspond to class modules. Modules associated with forms and reports are also class objects and can have properties and methods (public procedures).
- Microsoft Office objects: for example Word.Application.
- Other OLE objects: for example DSOFile.

ActiveX objects can be class objects or Automation/OLE objects.

Object variables are references to software components.

1

¹ The sole exception is the For Each . . . Next loop which can iterate the object members of a collection.

Generally a variable is declared as an object after which it is associated with (bound to) a particular object. Creating a new instance of an object from a class is called *instantiation*. Several statements can be used:

Dim statement: Declares a variable, specifies its data type, and allocates storage space. In this discussion Dim is used interchangeably with Private and Public, which differ only by scope. **Set statement:** Assigns an object reference to a variable or property.

CreateObject function: Creates and returns a reference to an ActiveX object. Is used with the Set statement. The Dim statement is extraneous with CreateObject.

GetObject function: Accesses an ActiveX object from a file and assigns the object to an object variable. Is used with the Set statement.

Syntax:

- Dim [WithEvents] varname[([subscripts])] [As [New] type]
 - a. <u>WithEvents</u> is a keyword that specifies that varname is an object variable used to respond to events triggered by an ActiveX object. WithEvents is valid only in class modules. You can declare as many individual variables as you like using WithEvents, but you can't create arrays with WithEvents. You can't use New with WithEvents.
 - b. New is a keyword that enables implicit creation of an object. If you use New when declaring the object variable, a new instance of the object is created on first reference to it, so you don't have to use the Set statement to assign the object reference. The New keyword can't be used to declare variables of any intrinsic data type, can't be used to declare instances of dependent objects, and can't be used with WithEvents.
 - c. The <u>type</u> can be the generic "Object" or a specific LibName. ClassName. The second value requires adding a Reference to the type library (LibName) with the "References" dialog box or setting a reference in code (adding an item to the References collection).
- Set objectvar = {[New] objectexpression | Nothing}
 - a. New keyword is usually used during declaration to enable implicit object creation. When New is used with Set, it creates a new instance of the class. If objectvar contained a reference to an object, that reference is released when the new one is assigned. The New keyword can't be used to create new instances of any intrinsic data type and can't be used to create dependent objects.
 - Nothing keyword discontinues association of objectvar with any specific object.
 Assigning Nothing to objectvar releases all the system and memory resources associated with the previously referenced object when no other variable refers to it.
 - c. Generally, when you use Set to assign an object reference to a variable, no copy of the object is created for that variable. Instead, a reference to the object is created. More than one object variable can refer to the same object. Because such variables are references to the object rather than copies of the object, any change in the object is reflected in all variables that refer to it. However, when you use the New keyword in the Set statement, you are actually creating an instance of the object.
- CreateObject(class,[servername])
 - a. The class argument uses the syntax appname.objecttype and has these parts: <u>appname</u> is the name of the application providing the object. <u>objecttype</u> is the type or class of object to create.
 - b. The name of the network server where the object will be created. If servername is an empty string (""), the local machine is used.
- GetObject([pathname] [, class])

- a. The pathname argument contains the full path and name of the file containing the object to retrieve. If pathname is omitted, class is required.
- The class argument uses the syntax appname.objecttype and has these parts:
 <u>appname</u> is the name of the application providing the object.
 <u>objecttype</u> is the type or class of object to create.
- c. If you don't specify the object's class, Automation determines the application to start and the object to activate, based on the file name you provide. Some files, however, may support more than one class of object, in which case you must supply the class argument.

Examples:

```
Dim xlApp1 As Excel.Application
Set xlApp1 = Excel.Application 'assigns a reference to a real class
   instance
Dim xlBook1 As Excel.Workbook
Dim xlSheet1 As Excel.WorkSheet
Set xlApp1 = Nothing
Set xlApp2 = CreateObject("Excel.Application")
Set xlBook2 = xlApp2.Workbooks.Add
Set xlSheet2 = xlBook2.Worksheets(1)
Dim xlApp3 As Object
Dim DSO As DSOFile.OleDocumentProperties
Set DSO = New DSOFile.OleDocumentProperties
Set objFile = CreateObject("DSOFile.OleDocumentProperties")
Dim MyDoc as Object
Set MyDoc = GetObject(UNC) ' where UNC is complete filename of Word document
Forms ("frmSales"). Maximize ' invokes the form's method ( public procedure)
```

Early and Late Binding

Using objects requires the application to bind to the object's interface. When this binding is done at runtime, it is called *late binding*. When it is done at compile time, it is called *early binding*. The two kinds of binding have different performances, i.e., one is more efficient than the other; but as there are no hard and fast rules as to which is more efficient, this is best determined by careful planning and testing. It is commonly thought that early binding is more efficient.

Early binding is accomplished:

- by Set objvar = LibName.ClassName
- by Set objvar = New LibName.ClassName

Late binding is acccomplished:

by Dim objvar As Object

Using the Me Property

Me is a property that refers to the current object and can be quite convenient. In a form event, Me refers to the form. In a control property, Me refers to the control. You can refer to the parent object with the Parent property:

```
Me.Parent
```

Accessing parentage is often necessary when a form has controls nested in controls, such as a form with a subform with a text control, or a form with a tab control with a page. When Me is a

subform, Me.Parent returns the parent form. When Me is a tab control page, Me.Parent returns the tab control itself.

You cannot use Me in a non-form or non-report module.

Using the ! and Dot Operators in Expressions

The! and operators are used in an identifier to indicate the type of the item that immediately follows.

The! operator indicates that what follows is a user-defined item. It can be used to refer to an open form or report, a control on an open form or report, or a field in a table. Example:

Me!HeightLabel

Plants!CommonName

The dot (.) operator usually indicates that what follows is an item defined by Microsoft Access. It can be used to refer to a property of a form, report, or control. It can also be used to refer to a field value in a SQL statement, a VBA method, or a collection. Example:

frmA.Visible
Forms![Orders]![Orders subform].Form

Access Objects

Code modules are in scope for the duration of their VBA project. Consequently public variables are publicly visible while the MDB is open. Code modules do not need to be explicity loaded or instantiated.

You can refer to an Access object either implicity or explicitly; the former is faster in execution.

Some objects may be coded without reference to parent objects. For example, AllForms can be specified without reference to the parent application object (AllForms!formA.IsLoaded vs. app.AllForms!formA.IsLoaded). Omitting optional references makes for cleaner looking code. However, when you experience inconsistent errors, especially errors relating to missing object references, the solution is to add the reference to the parent object. These kinds of errors are especially likely to occur when using OLE Automation (e.g., manipulating XLS files with VBA code in an Access database).

There are several ways to refer to a form or report object.

Syntax	Example
Reports!reportname	Reports!OrderReport
Reports![report name]	Reports![OrderReport]
Reports("reportname")	Reports("OrderReport")
Reports(index)	Reports(0)

Screen. ActiveForm can only be used in a form module.

Square brackets are necessary when the name has embedded spaces.

Code that uses the object's Me property executes faster than code that uses a fully qualified object name; Me can only be used if you're referring to the form or report from code within the class module.

For controls within a form or report, the implicit reference is faster:

implicit	Reports!OrderReport!NewData
explicit	Reports!OrderReport.Controls!NewData
implicit	Me![NewData]
explicit	Me("NewData")
implicit, use index in the Controls collection	Me(0)
explicit	Me.Controls!NewData
explicit	Me.Controls("NewData")
explicit	Me.Controls(0)

Set form, report, and control properties in Visual Basic:

set a property of a form or report	Forms!Customers.Visible = True
set a property of a control	Me!CustomerID.Visible = True (faster) Forms!Customers.Controls!CustomerID.Visible = True
set a property of a form or report section	Me!Section(acPageHeader).Visible = False Forms!Customers.Section(3).Visible = False

Form Instances

- When you open a form you create an instance of the form.
- The first instance is the default instance.
- You can create 2 or more instances with the same object variable name:

in Declaration section

Private colForms As New Collection

in the event that opens the form, which can be run one or more times

```
Dim frmA As Form_formYes
Set frmA = New Form_frmYes
frmA.Caption = . . .
frmA.Visible = True
colForms.Add frmA ' creates the Collections object
```

You can create 2 or more instances with different object variable names:

```
Dim frmA As Form_formYes
Dim frmB As Form_formYes
Set frmA = New Form_frmYes
Set frmB = New Form_frmYes
frmA.Caption = . . .
frmB.Caption = . . .
frmB.Caption = . . .
frmB.Visible = True
```

Class Objects

Class modules provide templates for custom objects; the name of the module is the name of the object. An instance must be explicitly created before you can access its public properties and methods.

There are two ways to instantiate a class object:

faster

```
Dim abc As ABasicClass
Set abc = New ABasicClass
```

slower

Dim abc As New ABasicClass

Then invoke a method:

```
abc.ListNames
```

A module can use an existing object reference to a class module. Say one module created a public instance named abc. The second module can access a class method:

```
abc.ListNames
```

Microsoft Office Objects

Office automation objects can be referenced per the following examples:

OLE Objects

OLE objects can be referenced per the following examples:

```
Dim DSO As DSOFile.OleDocumentProperties
Set DSO = New DSOFile.OleDocumentProperties
Set objFile = CreateObject("DSOFile.OleDocumentProperties")
```

Debugging

The basic debugging tools of breakpoints and Step Into are activated in two ways:

- 1) Open the MDB file while holding down the [Shift] key. This deactivates the AutoExec macro and opens the file in design mode. So debug works.
- 2) In the <u>Tools, Startup</u> window, check the "Use Access Special Keys". This activates debug.

Presentation of User Processes

This can be done with one or more forms. Each form has a control button that corresponds to a work process. The process buttons can be grouped by user role. Each button runs a macro/event procedure that in turn opens a window, report, or query or runs a module, or some combination of these. Explanatory labels minimize the need for user training. Built-in Access user-level security can be applied to the macros. It's better to code your own security as described in the last bullet in *Field Level Security*.

Classification Scheme

Some application things are classified, such as a *type* of document. The best way to present to users (on forms and reports) the full text (description) of the classification is to utilize a code (1–3 characters). Store the code (now a foreign key) in the data table and have a separate lookup table with code and description.

- a. Values in data table are restricted by lookup table.
- b. Code is used in the data table (to conserve storage space).
- c. User chooses value based on description.
- d. Lookup drop-down list in form uses description only.
- e. Lookup drop-down list in datasheet view uses both code and description.

Implementation:

- a. Data table code field is defined as: Required = no, Allow Zero Length = yes, Indexed = no; Lookup Display Control = Combo Box, Row Source Type = Table/Query, Row Source = Analysis Status (lookup table or query), Bound Column = 1, Column Count = 2, Column Heads = no (yes if 3+ columns), Column Widths = 0.3"; 1.5", ListWidth = sum of column widths. If there are 3 or more columns, use column heads. If using Combo Box, set List Width = sum of column widths.
- b. Set form field as combo box with similar properties.

Field-level Security

- The user view of data is in a form based on a query. The query has RunPermissions property = Owner's.
- The form protects fields that should not be changed (with properties Enabled = Yes, Locked = Yes, TabStop = No).
- The end user has Read Data and Read Design permissions on the query, Open/Run permission on the form.
- Since this was written, I have changed my opinion of Access security and the use of Access permissions. Multi-user security depends on a separate workgroup file which must be opened as the MDB application is opened. Making this happen all the time is a problem. So I've come to prefer user permissions stored in their own Permissions table in the same MDB. Code applies the permissions at logon/start-up time.

Application Security

- You can implement security "from scratch." This approach uses a Permissions table. The key to the table is the user ID. If the user has no active record in the table, they are refused entry. The active state is defined by an effective and expiration date; a null expiration date means the effective period is open-ended. Each application transaction has a yes/no permission. When a person starts the application, transactions that are not permitted are disabled on the main (menu) form.
- Permissions table:

NetworkUserIDText

EffectiveDate DateTime (Short Date)
ExpirationDate DateTime (Short Date)

PermTransactionA Yes/No PermTransactionB Yes/No

. .

(one permission flag for each secured transaction)

• Check permissions at start up (logon) and enable-disable individual menu (transaction) controls based on the value of the permissions flags.

```
Sub CheckPermissions()
On Error GoTo ErrorHandler
Dim contact As String
contact = "Phil Smith"
Dim msg As String
Dim rstPerm As Recordset
strSQL = "SELECT * FROM Permissions WHERE NetworkUserId = '" & NetId & "' "
Set dbs = CurrentDb
Set rstPerm = dbs.OpenRecordset(strSQL)
With rstPerm
    If .EOF = True Then
       msq = "No permissions exist for user " & NetId & "."
        GoTo PermissionsError
    End If
    If .Fields("EffectiveDate") > Date Then
        msg = "Permissions for user " & NetId & " not yet in effect."
        GoTo PermissionsError
    End If
    If .Fields("ExpirationDate") = "" Or .Fields("ExpirationDate") < Date Then</pre>
        msg = "Permissions for user " & NetId & " have expired."
        GoTo PermissionsError
    End If
    Forms!Main!cmdeditUserPermissions.Enabled = .Fields("PermEditPerm")
    Forms!Main!cmdReportUsers.Enabled = .Fields("PermRptPerm")
    Forms!Main!cmdImportBRT.Enabled = .Fields("PermImportBRT")
    Forms!Main!cmdImportMessages.Enabled = .Fields("PermImportMsg")
    Forms!Main!cmdEditExceptionMessages.Enabled = .Fields("PermExcMsg")
    Forms!Main!cmdArchive.Enabled = .Fields("PermArchive")
    Forms!Main!cmdCollectStatistics.Enabled = .Fields("PermStatistics")
    Forms!Main!cmdReportRecordCounts.Enabled = .Fields("PermRptRecCnts")
    Forms!Main!cmdReportPerformance.Enabled = .Fields("PermRptPerf")
End With
rstPerm.Close
Exit Sub
PermissionsError:
msg = msg & " Application will close now. Contact " & contact & " for
assistance."
MsgBox msg, vbOKOnly, "ABC Exceptions Logon"
DoCmd.Close acForm, "Main", acSaveNo
rstPerm.Close
Application.Quit acExit
ErrorHandler:
Dim ErrNum As Integer, ErrDesc As String, title As String
Call TurnOffHourglass
Set dbs = CurrentDb
title = "ABC Exceptions: Check Permissions"
ErrNum = Err.Number
```

```
ErrDesc = Err.Description
msg = "ERROR! (" & ErrNum & "): " & ErrDesc & ." Please contact your local
technical support."
MsgBox msg, vbOKOnly, title
End Sub
```

Initial Value of Code in Data Table

This discussion applies to the initialization of a table field when the table is loaded from a text file that does not include the subject field. The original subject was a Message table with a AnalysisStatusCode field.

It may be easier to load Message records (presumably done only once) with code = null value.

Access does not allow a table to have a primary key with a null value. So choices are: (1) Make code optional in Message table, set it to initial value as a follow-on step. (2) Add separate primary key to lookup table, with autonumber value, and let code be null. This probably screws up lookup functionality on form and datasheet.

Choice (1) seems best. Query to find and replace all null values to the logical value for null (0) can be used more than once, especially after adding new messages from SPL. Same approach can be used for other non-SPL fields.

Collect Statistics

Use table Statistics with fields:

Statistic ID autonumber

Statistic Date-Time default value: =Now()

Statistic text (100)

Count number (long integer)

Use queries that create a table record for a particular statistic. Example:

```
INSERT INTO Statistics (Statistic, Count)
SELECT "Count all messages" AS Expr1, Count(*) AS Expr2 FROM Message;
```

- Use a procedure that runs each query in sequence.
- Use query to count records grouped by some field value.

```
INSERT INTO Statistics ( Statistic, Count )
SELECT "Count messages with analysis status = " & [Analysis Status].[Analysis
Status Descr] AS Expr2, Count(*) AS Expr3
FROM Message LEFT JOIN [Analysis Status] ON Message.[Analysis Status Code] =
[Analysis Status].[Analysis Status Code]
GROUP BY [Analysis Status].[Analysis Status Descr];
```

Reference to File on File Server

There are two ways (in Windows) with which to refer to files on the network:

- (1) with a network drive, e.g., G:\directoryA\fileB.doc
- (2) with Universal Naming Convention (UNC), e.g., \\server\share\\directoryA\\fileB.doc

The latter is preferable because drive mapping is specific to an individual user (each person's G: drive may be unique).

In Access, the format is: server\\share\directory1\directory2\filename. Example: G0301\\crtsys\sjda\myfile.txt

For FileScripting Object:

\\GO301\crtsys\sjda\myfile.txt

When Date Function Returns 12/31/1899

The Date() function is supposed to return the current date. However, there are occasions when it returns 12/31/1899: In a SQL statement in VB that uses the Date() function. The problem is solved by surrounding the function with the # sign:

```
strWhere = "((Exception.[Run Date-Time]=#" & Date & "#)"
strSQL2 = strSQL1 & strWhere
varB = LoadArrayFromRecordset(strSQL2)
```

Dynamic Array

A dynamic array can (1) have one or more dimensions that reflect the value of a variable and/or (2) can be resized at any time.

```
Dim array() As Integer
Dim d As Integer
. . .
ReDim array(1 To d)
```

Data Types

There seems to be different names. When you define a new table your choices are: Text, Memo, Number, Date/Time, Currency, AutoNumber, Yes/No, OLE Object, Hyperlink, Lookup Wizard.

Field Size is used to differentiate between the different types of Numbers:

```
Byte holds values 0-255
Integer holds values 32,767 + or - 100
Long Integer holds values 2M + or = 100
Decimal and others unneeded by me so far.
```

These data types do not necessarily map to the ones used by the SQL Data Definition Language.

Hyperlink: A data type for an Access database field that stores hyperlink addresses. An address can have up to four parts and is written using the following format:

displaytext#address#subaddress#. This data type cannot be set in DDL SQL.

Text: In a Microsoft Access database, this is a field data type. Text fields can contain up to 255 characters or the number of characters specified by the FieldSize property, whichever is less.

Memo: In a Microsoft Access database, this is a field data type. Memo fields can contain up to 65,535 characters.

Datetime: Stores date and time between years 100 and 9999.

Integer: Stores an long integer with a value between -2,147,483,648 and 2,147,483,647 in 8 bytes.

Decimal: Stores a decimal number in 17 bytes.

Money: Stores a number in 8 bytes.

Bit: Stores a binary (yes/no) value in 1 byte.

In all its glorious wisdom, the designers of Access define different kinds of Numbers.

Memo Fields

- Hold text data longer than 255 bytes.
- A query of a table can get all the contents of a memo field.
- A report based on a query truncates memo fields.
- A report based on a table does not truncate memo fields.
- A make-table query truncates memo fields.
- In order to report memo fields completely, you can use a query to collect the data, convert it to a make-table (select into) query, run it, change the text format from text to memo, delete all the records, change the make-table query to an append query, then run the append query. This populates a table with memo fields. Then run the report based on this table.
- In a query to select records having a memo field that has text in it: WHERE (((Len(LTrim([CCSP CI MSG L].[DESCRLONG OVRD])))>1))
- See carriage returns in a memo field in the Zoom window (opened with [Shift]+[F2]. The line breaks where the carriage return is located.
- Insert a carriage return into a memo field with [Ctrl]+[Enter]. You can do this in datasheet view, form view, and the Zoom window.
- Memo fields can hold ASCII and/or DOS non-printing control characters like line breaks; the DOS characters are typically inserted in text by other programs, after which the text is imported into an Access table. In a report, DOS characters appear as small squares; this is a "feature" of the report designer. You can report actual line breaks instead of the squares: use the Replace function to replace the DOS characters with VbCrLf (carriage return and line feed), Chr(10), and/or Chr(13) characters.

The main ASCII characters are: 10 (line feed/new line), 11 (vertical tab), 12 (form feed/new page), and 13 (carriage return). It is common for CR and LF to be combined as Chr(13)+Chr(10); this is true for Windows.

There is also a VB variable vbNewline.

Empty Fields

- There are different forms of empty fields:
- (a) having zero-length, ""
- (b) null (has a Null value)
- (c) empty, never having been assigned a value since the Variant variable was created (a Variant variable is created with Dim without the As datatype phrase).
- Related values:

Null, result of explicit assignment of Null to a variable

Empty, represented by zero in numeric context or zero-length string ("") in a string context

Table definition properties include:

AllowZeroLength

Required

The following table shows the results of combining the settings of the AllowZeroLength and Required properties.

AllowZeroLength	Required	User's action	Value stored
		Presses ENTER	Null
No	No	Presses SPACEBAR	Null
		Enters a zero-length string	(not allowed)
		Presses ENTER	Null
Yes	No	Presses SPACEBAR	Null
		Enters a zero-length string	Zero-length string
		Presses ENTER	(not allowed)
No	Yes	Presses SPACEBAR	(not allowed)
		Enters a zero-length string	(not allowed)
		Presses ENTER	(not allowed)
Yes	Yes	Presses SPACEBAR	Zero-length string
		Enters a zero-length string	Zero-length string

- A table's records are initialized as Null if they are not required. You might prefer them to be initialized as a zero length string, if so set Default Values = "" in the table design.
- If you need to determine exactly what a blank table field is, write a query like the following: SELECT Subject.ID, Subject.PortalID, Subject.ProductID, IsEmpty([ProductID]) AS Empty, IsNull([ProductID]) AS [Null], IIf([ProductID]="", "yes", "no") AS Zero FROM Subject;
- When no item is selected in a list box, the value of the bound column is Null.
- Some fields look empty but are not. They contain character code(s) that don't correspond to keyboard characters. Example:

Chr(10) linefeed

Chr(13) carriage return (does <u>not</u> advance to next line)

ASCII codes are documented at www.asciitable.com.

- An expression like "*" in a query criteria selects zero-length values but not Null values.
- Other expressions in query criteria:

Is Null

Is Not Null

You can analyze data values:

```
SELECT LEN(field) As Size, . . . FROM . . . ORDER BY 1
```

LEN is blank when field value is Null.

When you run the test query to determine what is in a field (empty, null, or zero string), be aware that:

When Boolean values are converted to other data types, False becomes 0 and True becomes -1.

Related functions:

Nz(*fieldname*): converts Null value to zero.

Chr(character code): returns a string

Asc(string): returns an integer representing the character code of the first letter in the string

IsEmpty(*expression*): Returns a Boolean value indicating whether a variable has been initialized. IsEmpty returns True if the variable is uninitialized, or is explicitly set to Empty; otherwise, it returns False. An Empty variable is represented as 0 in a numeric context or a zero-length string ("") in a string context.

IsNull(*expression*): Returns a Boolean value that indicates whether an expression contains no valid data (Null). IsNull returns True if expression is Null; otherwise, IsNull returns False.

A test query:

```
SELECT ID, Site, IsEmpty(Site) AS IsEmpty, IsNull(Site) AS IsNull, IIf(Site="","yes","no") AS Zero, IIf(Site Is Null, "yes", "no") AS [Is Null], IIf( Site Is Not Null, "yes", "no") AS [Is Not Null] FROM CandidateFiles;
```

reveals for Site, a text field:

Original Value	Changed Value	IsEmpty?	IsNull?	""?	Is Null?	Is Not Null?
V	(unchanged)	0 (false)	0 (false)	no	no	yes
	(unchanged)	0 (false)	-1 (true)	no	yes	no
V	[Spacebar]	0 (false)	-1 (true)	no	yes	no
V	[Del]	0 (false)	-1 (true)	no	yes	no
	V	0 (false)	0 (false)	no	no	yes

Network Performance

- Index all fields used in a join or query selection criterion.
- Compile all code.
- Put all non-table objects on the local workstation. The down side of this approach is publishing MDB updates.

Multi-User Access

Choose a record locking scheme and specify it in Access options and for each form. Set options at application startup time:

```
Application.SetOption "Default Record Locking", 2 Application.SetOption "Use Row Level Locking", True
```

• Individual queries can have their RecordLock property set to reflect the desired scheme.

Data Type Mismatch

Data type mismatch in criteria expression. (Error 3464)

The criteria expression in a Find method is attempting to compare a field with a value whose data type does not match the field's data type.

Function InStr can return a Null if the underlying field is Null, and when you compare Null with an integer you get a type mismatch error.

"From what you describe, I suspect that the calculation or the comparison returns an error for one or more records, but that the error-producing calculation is not actually performed until it

has to be -- this is what Jet usually does when a query calls a function, to save time. When you apply a criterion to the field, though, the calculation has to be evaluated for all records right up front."

I struggled with a query with a WHERE InStrRev() > 0 clause generated a data mismatch error. The solution was to use function InStr.

Arrays

An array is an in-memory table that holds temporary data. It can have one or more dimensions. An array can be static, meaning its size is fixed, or dynamic, meaning its size can be changed one or more times. An array can contain any single data type. An array is normally zero-based, meaning the first item has an index of zero. This can be changed in two ways.

An array is not an object. The things that act on arrays are statements.

Define an array:

J	
Dim myArray(10) As String	defines a static array of 11 string elements, zero-based; all elements are initialized to default value (zero-length string)
Dim myArray(0 To 10) As String	defines a static array of 11 string elements, zero-based; all elements are initialized to default value (zero-length string)
Dim myArray(1 To 10) As Integer	defines a static array of 10 integer elements, 1-based; all elements are initialized to default value (zero)
Dim myArray(6 To 10) As Integer	defines a static array of 5 integer elements, 6-based
Dim myArray(10, 3) As String	defines a static array of two dimensions, the first with 11 string elements and the second with 4 string elements, zero-based; all elements are initialized to default value (zero-length string)
Dim myArray() As String	defines a dynamic array of strings

Change the size of a dynamic array (after it has been defined)

ReDim myArray(20)	changes the number of array entries to 20, zero-
	based; any existing values are discarded
ReDim Preserve myArray(1 To maxArray)	changes the number of array entries to the
	value of the variable maxArray, 1-based; any
	existing values are preserved

Load data into an array:

<pre>myArray(x) = "Smith"</pre>	sets the value of the xth element to the literal
myArray(2, 5) = strSQL	sets the value of the element to the value of the
	variable

Read data in an array:

strVar = myArray(x)	retrieves the value of the xth element of the

	array
strVar = myArray(3, x)	retrieves the value of the (3, x)th element of the
	array

Get boundaries of an array:

Upper = UBound(myArray)	returns a Long containing the largest available subscript for the indicated dimension of the array
<pre>Upper = UBound(myArray, 3)</pre>	returns the largest available subscript for the 3rd dimension of the array
<pre>Upper = LBound(myArray)</pre>	returns a Long containing the smallest available subscript for the indicated dimension of the array
Lower = LBound(myArray, 3)	returns the smallest available subscript for the 3rd dimension of the array

Another way to change the lower bound for all arrays in the module is to include in the Declarations area:

Option Base 1

Can you set one array to another? No, but you can copy the contents of one array to another one cell at a time.

An array's memory space allocation is released when its lifetime expires. When an array is declared within a procedure, the array is released when the procedure ends. When an array is declared in a standard code module's Declaration section, the array is released only when Access closes. You can release a dynamic array early:

Erase myArray

When Erase is used on a static array, the array is reinitialized but not released.

ASCII Non-printing Carriage Control Characters

The vbCrLf et al constants are defined in the Visual Basic for Applications type library and can be used anywhere in your code in place of the actual values.

Access requires both a carriage return character (ASCII 13) and a line feed character (ASCII 10), in that order. Excel is content with just a line feed.

Include the following function in a SQL query: Replace([TextField], Chr(10), Chr(13) & Chr(10)). This does work; the later remark in section String Manipulation that says Replace cannot be used in SQL is in error.

Units of Measure

Form and report controls have some properties that specify space elements (such as margins and line spacing). These properties can be specified with one of several units of measure:

- inches (in or ")
- centimeters (cm)

- pixels (px)
- points (pt). There are 72 points in an inch, and approximately 28 in a centimeter.
- twips. There are 20 twips in a point, 1,440 in an inch, and 567 in a centimeter. Twips are entered without a unit of measure.

Examples:

LineSpacing = 0.02"

LineSpacing = 4cm

LineSpacing = 670

Typography in Forms and Reports

The following control properties can be used to set the typography of text.

Box properties:

Object	Property	Values
border	BorderColor	
	BorderStyle	Transparent Solid Double solid Dashed Short dashes Dots Sparse dots Dash Dot Dash dot dot
	BorderWidth	Hairline 1–6 pts
	BackStyle	Normal Transparent
inside margin	TopMargin	integer (UOM), e.g., 5 pt, 100 (twips)
	BottomMargin	integer (UOM)
	LeftMargin	integer (UOM)
	RightMargin	integer (UOM)

Text

Object	Property	Values
typeface	FontName	e.g., Palatino Linotype
	FontSize	integer, UOM = points
	FontWeight	Thin Extra Light Light Normal Semi-bold Bold Extra Bold Heavy
	FontBold	Yes/No; if yes FontWeight = Bold, else = Normal
	FontItalic	Yes/No
	FontUnderline	Yes/No
	ForeColor	
paragraph	LineSpacing	integer (UOM); e.g., 13 pt

Using ActiveX Controls

These are a mixed blessing. They offer more functionality than available in the basic Access controls. However not all ActiveX controls work well in Access (see Microsoft Help and Support Article 208283 for a list of Access-compatible controls, http://support.microsoft.com/kb/208283).

Some third-party controls are licensed only for use by end users, not developers. The technology has versioning and license problems, so much so that many developers avoid them.

ActiveX controls are .OCX files that must be registered on your computer before you can use them. Use the Access menu <u>Tools</u>, <u>ActiveX-Controls</u> to see what has been registered and to register new controls. Once a control is registered it shows up in the "Insert ActiveX Control" dialog box.

Applications in Access

Access is a great framework for developing and delivering applications consisting of a database and UI. Access applications run under Access, meaning the application windows open within the Access window.

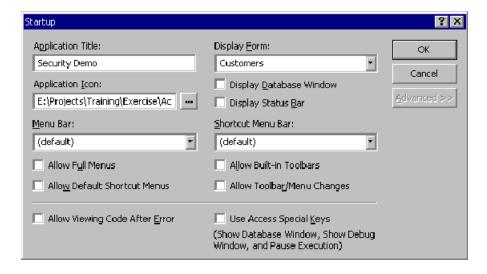
There are certain built-in Access features which you may want to hide and/or block user access to:

built-in menu bar built-in toolbar database window

The problem is that once these have been disabled, you will eventually need to get to them.

Startup Options

The Startup dialog box is used to define how an application starts up.



Control	Effect
Application Title	replaces the text "Microsoft Access" in the Access title bar with the entered text
Application Icon	replaces the icon in the Access title bar with the selected image (must be a .bmp or .ico file type)
Menu Bar	replaces the built-in menu bar with a custom one
⊠ Allow Full Menus	if "yes" lets the full built-in menu appear; if "no" hides the built-in menu items that can allow the user to access design views of the database objects and possibly change the design. If "no" the View, Insert, and Tools menus are hidden and some items on the other menus are hidden.
⊠ Allow Default Shortcut Menus	if "yes" allow the built-in shortcut menus

Control	Effect
⊠ Allow Viewing Code After Error	[this option is available not in my version of Access 2003]
Display Form/Page	name of an application form or page to open automatically when the application begins. Depending on the complexity of the application., this might be a control panel (or dashboard) or the primary data form.
⊠ Display Database Window	"no" prevents the database window from opening when the database opens. This window will still be available when (1) full menus are allowed (because Window, Unhide lets the user unhide the database window) or (2) Access special keys are allowed (because [F11] opens the database window).
⊠ Display Status Bar	"no" prevents the status bar from being visible; I do not know why this would be desirable
Shortcut Menu Bar	replaces built-in shortcut menus with your own
☑ Allow Built-in Toolbars	"no" prevents the built-in toolbar from being visible
⊠ Allow Toolbar/Menu Changes	"no" prevents the user from changing the toolbars or menu bars with the Customize dialog
☑ Use Access Special Keys	controls the action of four keys described in the table below

Special Key	Result when Key is Allowed
F11	Brings the Database window to the front (unhides it).
Ctrl+G	Opens up the VBE Immediate window.
Ctrl+F11	Toggles between the custom menu bar and the built-in menu bar.
Ctrl+Break	In a Microsoft Access project (that connects to a Microsoft SQL Server database), stops Access from retrieving records from the server.
Alt+F11	Starts the Microsoft Visual Basic Editor.

Hiding Objects

Another way to limit access to built-in Access features is to hide objects. To hide an object in the Database window (or Navigation pane), right-click it and choose Properties. Then, check the Hidden Attribute option.

Hidden objects can be unhidden. To view all hidden objects, you simply choose menu <u>Tools, Options</u>, click the View tab, and then select the Hidden Objects option in the Show section. (If you select an object's Hidden attribute, but the Database window still lists it, the Hidden Objects option is probably selected.) In Access 2007, right-click the Navigation pane's menu bar, select Navigation Options, select Show Hidden Objects, and click OK.

The built-in menu bar cannot be completely hidden, but it can be moved as a disguise.

AutoExec Macro

This macro is run automatically when the application is opened. It can be used to initialize startup options and limit users' access in different ways. For example, it can hide ALL menu bars and toolbars. If you do this, be sure you have a tried-and-true recovery method!

Recovery

If the startup options disable full menus, special keys, and the database window, there is not much the user can do to circumvent the normal expected use of the application.

To bypass the startup options that are set for the Access database, hold down the [Shift] key while you open the Access file. You can do this when you are opening the database from (1) a shortcut on the desktop, (2) a shortcut in Windows Explorer, (3) the filename in Windows Explorer, and (4) the filename on the <u>File</u> menu.

NOTE: The functionality of the [Shift] key that permits you to bypass the startup options can itself be disabled to enforce the startup options that are set for the Access database. To do this, in VBA code set the AllowBypassKey property to False. Don't do this unless you have a way to get around it!

The special keys described above can be helpful if they are not disabled. There is another special key:

Special Key	Result
F10	Makes the menu bar active. From the menu bar you can access the Startup options—but only if the Tools menu is visible, which it won't be if the Allow Full Menus startup option is active.

The <u>Window</u>, <u>Unhide</u> menu can be used to show the database window, but only if full menus are allowed.

Suggestions

You have to decide how dangerous it might be if users are allowed unconstrained access to the design tools. If you block all access to these tools, then you need to do a few other things in order to provide access to the application objects:

- locate the actual data in a separate file and link to the tables in the application file
- use a third file to hold the full application without access constraints
- when you need to change an object in the controlled application, export the changed version from the third file
- create a custom shortcut key that runs VBA code to show the built-in menu (from which you can access all tools)

It may be enough to divide users into two groups:

- common users: have limited access to Access tools and access to only some application objects
- administrators: have full access to both Access tools and to all application objects.

This can be accomplished in two ways:

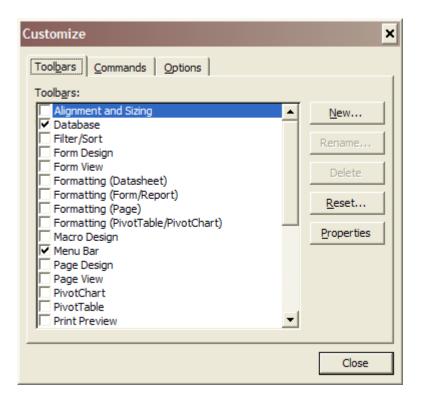
- a. Use three MDB files as described in the first three bullets above. The first file contains the UI objects needed by the common users. The third file contains all the UI objects.
- b. use two MDB files, one with only the actual tables, the other with startup options to hide the database window but allow the special keys. The file with the startup options opens with an application form from which only some objects can be accessed. The administrators can access all objects via the database window, which is made visible by [F11].

Bars: Menu Bars, Toolbars, and Shortcut Menus

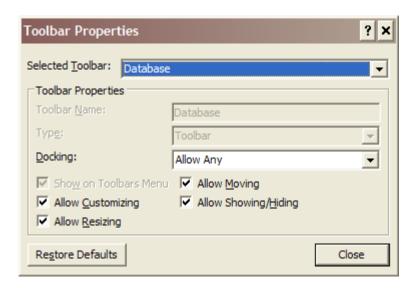
The menu bar is a cascading menu that normally appears on a horizontal bar at the top of the Access window. The built-in menu bar is movable and dockable (to the edge of the Access window).

A shortcut menu, or context menu, is a menu that shows a list of commands relevant to a particular item. To display a shortcut menu, right-click the item or press [Shift+F10].

Access allows you to customize these with the Customize dialog box (accessible from menu <u>Tools, Customize</u> or <u>View, Toolbars, Customize</u>). You may also create new ones. These customizations are specific to a particular application (database)—I'm really not sure about this. You can see from the image below that Access (actually Microsoft) considers all these things as "toolbars."



You can also change the properties with the dialog box pictured below.



A *global menu bar* is a special custom menu bar that replaces the built-in menu bar in all windows in your database application, except where you've specified a custom menu bar for a form or report. It can also be used to replace built-in shortcut menus for datasheets, forms, form controls, and reports.

Expressions and Functions

General

- When a field name is the same as a reserved word (such as a property), refer to it with the table name: [NameInfo]![Name]. Note this is the advice of the help file, but it did not work for [PLANT]![Height].
 - Properties to watch out for: Address, Bookmark, Caption, Column, Count, Cycle, DateCreated, DateModified, Default, Document, FileFormat, Format, FullName, FullPath, Height, Hyperlink, InsideHeight, InsideWidth, Kind, Name, Page, Pages, Path, Port, ProductCode, Type, Width.
- Neither should you use any names that are the same as the functions, statements, and methods in Visual Basic. To use an intrinsic language function, statement, or method that conflicts with an assigned name, you must explicitly identify it: Precede the intrinsic name with the name of the associated type library. For example, if you have a variable called Left, you can only invoke the Left function using VBA.Left.
- When you have a field name that is the same as an intrinsic VB thing, one workaround is to rename/alias the field in the report's record source query (fieldname as uniquename), then use the new name in the control's control source.
- Functions can be used just about everywhere: in table definitions, queries, forms, reports, and VB code. There are standard VB functions and you can code your own.
- See "Refer to Form Controls" section for specifics that apply to both forms and reports.
- Set the data source of a form/report control to the results of a function. In the following example, ExpandMsgText is a custom function with one parameter:

```
ControlSource =ExpandMsgText([Exception ID])
```

To set/compare value of Yes/No field, use True/False without double-quotes:

```
WHERE [Review Flag] = False
IsNumeric(expression) function returns True or False
IsDate(expression) function returns True or False
```

<u>Like</u> is an operator that performs wildcard comparison. See page Error! Bookmark not defined..

```
= DMax("InterviewDate", "Interview", "JobTitle = ' & JobTitle & " ' ")
BE CAREFUL to use correct " and ' characters. If you copy code from Word, they may be wrong.
```

Concatenate field(s) and literals. There are two concatenation operators: & and +. The
former performs basic concatenation. The latter combines two strings to form one string and
propagates null values (if one value is Null, the entire expression evaluates to Null).

```
=[Field1] & [Field2]
=[Field1] & "and"
=[This] + [That]
```

- If you define the ControlSource of a control on a form or report to include the name of a table field, the control name must be different from the table field name (or there will be a circular reference error).
- Date literals are bracketed by pound sign in code and SQL:

=#09/06/2009#

Logical Operators

Not is a logical operator that lets you toggle a binary property setting.

object.property = Not object.property

• **And** is a logical operator that performs a logical conjunction on two expressions. The result is True if both expressions are true. If either expression is False, the result is False.

result = expression1 And expression2

• **Or** is a logical operator that performs a logical disjunction on two expressions. If either or both expressions is True, the result is True.

result = expression1 Or expression2

• **Xor** is a logical operator that performs a logical disjunction on two expressions. If one, and only one, of the expressions is True, the result is True.

result = expression1 Xor expression2

• **Eqv** is a logical operator that performs a logical equivalence on two expressions. If either expression is Null, the result is also Null. If both expressions are True or False, the result is True. If one expression is True and the other is False, the result is False.

result = expression1 Eqv expression2

Comparison Operators

Date Functions

Also see **Formatting** in next section.

Now() current date-time
Date() current date
Time() current time

Day(*date*) returns day of month as integer

Month(*date*) returns an integer equal to the number of the month, no leading zero

Year(date) returns an integer equal to the year, e.g., 2004

DateValue(*date*) returns a Date variant

DateValue(*datetime*) returns a Date (drops the time)

DatePart(interval,date)

interval = "yyyy" year interval = "q" quarter interval = "m" month interval = "y" day of year

interval = "d" day interval = "w" weekday interval = "ww" week

```
interval = "h"
                      hour
    interval = "n"
                      minute
    interval = "s"
                      second
    example: DatePart("m", 25-Dec-1992) returns 12
    example: DatePart("w", [CompleteDate]) returns Monday
FormatDateTime(date,format)
                                     uses VB constants for the format choice
object.FormatDateTime(LastSaveDate, vbShortDate)
varRow(intArray) = FormatDateTime(varRow(intArray), vbLongTime)
Format(date, format)
                      this apparently has changed in Access 10
    format = "LongTime" ??
    format = "ShortDate" ??
    format = "hh:mm"
    format = "h:m:s"
```

- If you want dates to display as 4-15-2005 instead of 4/15/2005 you can do this with the Format property in the table definition: After selecting Data type = Date/Time, enter Format as m-d-yyyy. Then that format is reflected in queries, forms, and reports.
- If you are adding a literal to a date value at report time, specify that in the date's Format property on the report. Example: "Birth 1 "m-d-yyyy You can combine this with CanShrink to keep an absent date from appearing at all.

Arithmetic Functions

```
Sum: +
=[FieldA] + [FieldB]

Subtract: -
=[FieldA] - [FieldB]

Multiply: *
=[FieldA] * 1.06

Division: /
=[FieldA] / [FieldB]
```

Aggregate Functions

These functions apply to set of values of the named field in the dataset.

```
• Average: Avg()
```

=Avg([Freight]) Uses the Avg function to display the average of the values of the Freight control.

Count: Count()

=Count([OrderID]) Uses the Count function to display the number of records in the OrderID control.

Sum: Sum()

```
=Sum([Sales]) returns sum of the values of the Sales control
=Sum([Quantity]*[Price]) returns the sum of the product of the values
of the Quantity and Price controls
=[Sales]/Sum([Sales])*100 returns the percentage of sales, determined by
dividing the value of the Sales control by the sum of all the values of the
Sales control
```

Access provides domain aggregate functions. These are often used in VBA procedures.

```
Domain maximum: DMax(expression, domain[, criteria])
=DMax("[Freight]", "Orders", "[ShipRegion] = 'CA'")
```

The DMin and DMax functions return the minimum and maximum values that satisfy criteria. If *expression* identifies numeric data, the DMin and DMax functions return numeric values. If *expression* identifies string data, they return the string that is first or last **alphabetically**.

The *expression* can be the name of a form control or table field; it can also be a constant or function. *domain* can be the name of a table or query but not be SQL. *criteria* can be string variable that is a WHERE clause without the WHERE. In VBA variables can be used to hold the value of one or more parameters.

The DMin and DMax functions ignore Null values in the field referenced by *expression*. However, if no record satisfies criteria or if *domain* contains no records, the DMin and DMax functions return a Null. Be sure the receiving object can handle a Null value, if a variable declare it as a Variant.

Domain count: DCount(expression, domain[, criteria]) expression can be the name of a form control or table field; it can also be a constant or function; domain can be the name of a table or query but not be SQL, criteria can be string variable that is a WHERE clause without the WHERE.

```
=DCount("[OrderID]", "Orders", "[ShipRegion] = 'CA'") Uses the DCount function to display the number of records in the OrderID control whose ShipRegion is CA.
```

- Domain average: DAvg (expression, domain[, criteria])
 =DAvg ("[Freight]", "Orders", "[ShipRegion] = 'CA'")
- Domain first/last -- random: DFirst (expression, domain[, criteria]) and DLast(...)
 DFirst("[Orders]![OrderDate]", "[Orders]") to return a random record from a particular field in a table or query
- Domain maximum: DMax (expression, domain[, criteria])
 =DMax("[Freight]", "Orders", "[ShipRegion] = 'CA'")
- Domain minimum: DMin (expression, domain[, criteria])
 =DMin("[Freight]", "Orders", "[ShipRegion] = 'CA'")
- Domain sum: DSum (expression, domain[, criteria])
 =DSum("[Freight]", "Orders", "[ShipRegion] = 'CA'")

Lookup Function

Domain lookup: DLookup: expression, domain[, criteria])

Returns a value from a specified set of records (or domain).

expression is a field, calculation, control on a form, or function that you wish to return. domain is the set of records. This can be a table or a query name.

criteria is optional. It is the WHERE clause to apply to the domain.

The function is the equivalent of a SQL query like: SELECT expression FROM domain WHERE criteria.

The function can be used as the source of a calculated contro, especially when the field that you need to display isn't in the record source on which your form or report is based. It can also be used in a query.

The DLookup() function returns one value from a single field even if more than one record satisfies the criteria. If no record satisfies the criteria, or if the domain contains no records, DLookup() returns a Null.

```
=DLookUp("[LastName]", "Employees") ' returns LastName field of first record =DLookup("[Freight]", "Orders", "[ShipRegion] = 'CA'") =DLookup("UnitPrice * Quantity", "Order Details", "OrderID = 10248") =DLookup("CustomerID", "Orders", "OrderID = " & Forms![Orders]!OrderID) =DLookUp("Directions", "JETypeDirections", "[Type] = " & "JEType")
```

Format Functions

Some of these may be new to Access 10 (2003).

```
FormatDateTime(Date[,NamedFormat])
```

where the named formats are expressed by constants: vbGeneralDate, vbLongDate, vbShortDate, vbLongTime, vbShortTime.

I was unable to use FormatDateTime successfully in a report property source statement; Access did not like the constant. Use function DateValue instead!!

```
FormatNumber(Expression[,NumDigitsAfterDecimal [,IncludeLeadingDigit [,UseParensForNegativeNumbers [,GroupDigits]]])

FormatCurrency(Expression[,NumDigitsAfterDecimal [,IncludeLeadingDigit [,UseParensForNegativeNumbers [,GroupDigits]]])
```

Get Length of a Text/Memo Field

Use LEN(string) function.

Returns a Long containing the number of characters in a string or the number of bytes required to store a variable. The thing to remember is that if *string* contains Null, Null is returned. Be sure the receiving object can handle a Null value, if a variable declare it as a Variant.

To get the maximum length of the values in a text table field, combine Len with DMax. lngMaxLength = Len(DMax(strFieldname, strTableName))

But, because DMax can return a Null, you should declare the receiving variable as a Variant. When lngMaxLength is Null, you will want to change it to zero. The Nz() function will not work properly in this case. A method that works is:

```
If IsNull(lngMaxLength) Then lngMaxLength = 0
```

A way to quickly determine the longest value in a field with a query:

```
SELECT ID, Len(Applications)
FROM Scanned
ORDER BY 2 DESC;
```

Return One of Two Values

• IIf() function has three parameters: condition being tested, value if test returns Yes, value if test returns No.

=IIf([Confirmed] = "Yes", "Order Confirmed", "Order Not Confirmed")

Carriage Control Functions

You can include ASCII non-printing carriage control characters in strings and use the strings in text fields. The characters are non-printable ASCII codes which are commonly provided with the Chr function.

Char Function	VB Constant	Meaning
Chr(8)	vbBack	backspace character
Chr(9)	vbTab	horizontal tab character
Chr(11)		vertical tab character
Chr(12)		frm feed (to top of next page)
Chr(10)	vbLf	line feed
Chr(13)	vbCr	carriage return (does <u>not</u> advance to next line)
Chr(13) + Chr(10)	vbCrLf	carriage return–line feed combination

Apostrophes and Quotation Marks in Strings

[Note that page 101 on String Manipulation needs correction in its discussion of apostrophes: single quotes = apostrophe = Chr(39)]

There are some keyboard characters that are troublesome in text expressions, especially apostrophes, single quotation marks, and double quotation marks. In some contexts an apostrophe is identical to a single quotation mark; this discussion will refer to apostrophes (including single quotation marks) and quotation marks, the latter being the double version.

An apostrophe is just another keyboard character when you type it into a form control or a datasheet cell. But try and include it in a text string, especially in a concatenated query (SQL statement), and you will have grief. Ditto for the quotation mark. The query string cannot contain either character, neither can parameterized queries.

This is complicated by the use of quotation marks to bracket text: "this is a text string"

There are several techniques:

- You can embed paired apostrophes within paired double quotes and vice versa (Access and SQL Server expect string arguments to be bracketed by single or double quotation marks in matching pairs.
- You can use 2 quotation marks instead of an apostrophe.
- You can double-up any apostrophe and quotation mark.
- You can use special characters: quotation mark is Chr(34), apostrophe is Chr(39).

Example:

Original String	Encoded String
-----------------	----------------

Original String	Encoded String
O'Brien	var = "O'Brien"
	var = 'O''Brien'
	var = "O""Brien"
	var = "O" & Chr(39) & "Brien"
I said "boo"	var = 'I said "boo"'
	var = "I said " & Chr(34) & "boo" & Chr(34)

The Replace function is useful for replacing pesky characters by non-pesky ones. In the following examples the original string is a variable whose value is: O'Brien.

```
Replace(strText, "'", "'"")
Replace(strText, Chr(39), Chr(39) & Chr(39))
```

When you build a SQL statement in code, you are already dealing with the quotation marks that bracket the string:

```
strSQL = "SELECT . . . FROM . . . "
```

When the contents of the SQL statement may also include apostrophes and/or quotation marks, those characters have to be handled properly in order to avoid a syntax error. For example, if your SQL statement is like:

```
INSERT INTO tableA (fieldA1) VALUES "string here"
and you build this statement dynamically, you will have to edit the value string:
strSQL = "INSERT INTO tableA (fieldA1) VALUES " & Replace(string here, "'",
"''")"
```

This is complicated if your query is creating a new field containing a text string. For example, the goal is to create a string with the format;

```
COMMENT ON TABLE <tablename> IS "comment text";
```

You could create such a string:

```
SELECT [Table], "COMMENT ON TABLE " & [Table] & " IS " & sss AS NewField . . . where sss is the goal text string, which may contain one or more apostrophes and quotation marks or may be null. In this case you have to contend with the quotation marks that delimit the individual text strings.
```

The answer is:

```
SELECT [Table], "COMMENT ON TABLE " & [Table] & " IS " & IIf([Comment] is Null,
'" ";', '"' & Replace([Comment], Chr(39), Chr(39) & Chr(39)) & ' ";') AS
SQLtext
FROM ABC;
```

Type Conversion Functions

When text is put into a string to be displayed, non-string type data must be converted to a string data type.

Function CStr converts an expression of different data types to a string.
 If the expression is Boolean, the resulting string is True or False.
 If the expression is Date, the resulting string is in the short date format.

If the expression is Null, there is a run-time error.

If the expression is Empty, the resulting string has zero length ("").

If the expression is Error, the resulting string is the word Error followed by the error number.

CStr on a Word constant like wdFieldTOC which has a type of WdFieldType returns an integer. Using Chr returns a non-letter character, useless. The difficulty with constants is the text name is just a convenience, the real value is an integer.

Switch Function

This function can be useful in queries when you have a case situation. The Switch function argument list consists of pairs of expressions and values.

Switch evaluates all of the expressions, from left to right. Then it returns either (1) the value associated with the first expression to evaluate to True or (2) Null if none of the expressions is true. If the parts aren't properly paired, a run-time error occurs.

Syntax:

```
Switch(expr-1, value-1[, expr-2, value-2 ... [, expr-n, value-n]])
```

You can add an "elseif" pair at the end of the argument list with the expression True and some value. This works because the expression True evaluates as True. For example:

```
Switch(fieldA = "Honda", "HO", fieldA = "Accord", "AC", True, "OTHER")
```

Example returns the name of a language that matches the name of a city.:

This is equivalent to Select Case:

```
Select Case CityName
Case "London"
    "English"
Case "Rome"
    "Italian"
Case "Paris"
    "French"
End Select
```

An example of the function in a query:

```
UPDATE CandidateFiles AS CF INNER JOIN MigrateWorksheet AS MW ON CF.ID = MW.ID
SET CF.LibraryName = MW.LibraryName, CF.Library = Switch(MW.LibraryName =
"Public", "U", MW.LibraryName = "Private", "R", MW.LibraryName = "Contacts",
"C")
WHERE CF.ID IN (SELECT ID FROM MigrateDiscrepancyE);
```

TypeName Function

The TypeName function returns a string of the name of the data type of the subject variable. Syntax: TypeName(*varName*)

Constants

Yes/No fields:

Constant	Value	Description
vbTrue	-1	True (yes)
vbFalse	0	False (no)

CallByName Function

Used to get or set a property, or invoke a method at run time using a string name.

Syntax

CallByName(object, procname, calltype,[args()])

Named Argument	Description
object	Variant (Object). The name of the object on which the function will be executed.
procname	Variant (String). A string expression containing the name of a property or method of the object.
calltype	Constant. A constant of type vbCallType representing the type of procedure being called: vbGet (indicates Property Get), vbSet (indicates Property Let), vbMethod
args	Variant (Array).

Examples:

CallByName Text1, "MousePointer", vbLet,	set the MousePointer property of a text box
vbCrosshair	
Result = CallByName (Text1, "MousePointer", vbGet)	gets the value of the MousePointer property
CallByName Text1, "Move", vbMethod, 100,	invokes the Move method to move the text
100	box

Revision: 9/2/2012

Page 34 of 119

Formatting

- You can format fields in tables and queries and controls in forms and reports.
- Formats applied to table fields persist into queries, forms, and reports.
- Apply formats in reports if you don't want them to appear in anything else.
- Use the Format property.
- May also use the Format function and other functions.
- You can use the IIf function as a data Source, but it is more work than using the Format property with the CanGrow and CanShrink properties.
- There's lots more to say, but it will have to wait.

Format Function

This can be used to put a date-time stamp in a filename:

```
arch = libpath + "catalog" + Format(f.DateCreated, "yymmddhhmm") + ".xls"
```

Format Property

- The Format property uses symbols to create custom formats for datasheets, forms, and reports.
- For a control, you can set this property in the control's property sheet. For a field, you can set this property in table Design view (in the Field Properties section) or in Design view of the Query window (in the Field Properties property sheet). You can also set the value in a macro or Visual Basic code.
- If you set a field's Format property in table Design view, that format is used to display data in datasheets.
- Some symbols apply to any data type:

Symbol Meaning

display spaces as literal characters. (space)

"ABC" display anything inside quotation marks as literal characters.

force left alignment instead of right alignment.

- fill available space with the next character.
- display the next character as a literal character. You can also display literal characters by placing quotation marks around them.
- display the formatted data in the color specified between the brackets. Available colors: [color] Black, Blue, Green, Cyan, Red, Magenta, Yellow, White.

Format Property for Text and Memo Data Types

The Format property uses symbols to create custom formats.

Symbol Description

- text character (or space) is required @
- text character is not required &
- force all characters to lowercase
- force all characters to uppercase

Revision: 9/2/2012

Page 35 of 119

Custom formats can have one or two sections. Sections are separated by ";." Each section contains the format specification for different data:

Description Section

first format for fields with text

format for fields with zero-length strings and Null values second

Examples:

<u>Property Value</u>	<u>Data</u>	<u>Display</u>
@@@-@@-@@@	123456789	123-45-6789
@@@@@@	12-34-56	123456
>	test	TEST
<	Susan	susan
@;"Unknown"	[empty]	Unknown
@;"Unknown"	abc	a
!"(Cell 1) "@	[empty]	[empty]
!"(Cell 1) "@	abc	(Cell 1) abc

Format Property for Dates

There are predefined formats: "Short Date", "General Date", "Long Date", "Long Time",

```
object.Format = "Short Date"
```

There are custom formats. These use particular characters as symbols. Enclose separators and any literals in quotation marks.

```
object.Format = mmmm dd", "yyyy
```

Function Int

Function Int returns the integer portion of a number with no comma separators. Without it you can get a number with decimals and comma separating thousands. Their presence can change the number into a string. For instance, I wrote a query to add 4 fields and divide by 4, in order to average their values. Each field was defined as a Double, but the result of the arithmetic was a string.

Revision: 9/2/2012

Table Stuff

Relational Database

Access supports a database composed of tables that adhere to the relational model and that are normalized, i.e., in the third normal form. Consult Wikipedia for an explanation of database normalization. You will encounter frustrating problems creating forms, reports, and queries when your database is not normalized. Get your data model right!

Primary Key(s)

The role of the primary key (PK) is to identify the field(s) that uniquely identify a record. Access automatically prevents duplicate or null values from being entered in PK fields.

To set a PK in table design view:

- 1. Select the field(s). To select only one field, click its row selector. To select 2+ fields, hold down the [Ctrl] key and click the row selector for each field.
- 2. Click the PK icon in the Table Design Toolbar or select menu Edit, Primary Key.

Indexes

Indexes have pros and cons:

- they speed finding records by SELECT queries
- they slow append queries (because the index must be maintained too).

An index can be established for 1 to as many as 10 fields.

It can be helpful to create an index for key queries. An index is automatically created for the PK and for autonumber fields. An index can be defined as unique and to ignore nulls (which excludes records from the index whose indexed field contains Null values). An index has a sort order based on the sequence of the fields in the index; each field can be ASC or DSC.

Empty a Table

Do with a query:

DELETE * FROM [table name];

Import Table Specification

An *import/export specification* can be created to define the way in which an imported file can be transformed into an Access table. The specification can only be created and edited using the Import Text Wizard; access it with the [Advanced] button. The specification defines the input file format (for text files, as delimited or fixed width); the file origin (for text files, as Windows ANSI or DOS PC-8; the format of dates, times, and numbers; and, for each field, its name, data type, whether it is indexed, and whether it is skipped in the import. These specifications can be a parameter in the macro action TransferText (or DoCmd.TransferText). They can be imported to a different Access database along with menus and toolbars. They cannot be accessed via code! They are not in the object model.

Import Excel Worksheet

You can import an Excel worksheet into an existing table or into a new table. Importing accomplishes:

- The columns in the list are the fields in the database.
- The column labels in the list are the field names in the database.
- Each row in the list is a record in the database.

There are a few restrictions:

- The XLS file can have only one worksheet.
- The XLS file must have the same type of data in each field (column)
- The XLS file must have the same fields in every row.
- Only one heading row in the XLS file.
- If you import into an existing table, the XLS file column headings must match the table field
- If you import into an existing table, the XLS file column must have the same data type as table fields. Since XLS file columns don't really have data types, you mayhave to finagle getting a number into a text field (put a letter in the field in the first record, then delete it from the table).
- I suspect that the size of the table field must be large enough to hold the XLS cell. Not sure what happens if this is not true.

Link Table to DB2 Table

- You can access data in DB2 tables.
- Create linked Access table with File, Get External Data, Link. On "Import" window, select file type = ODBC(). A "Select Data Source" window opens. Select DSN Name = LOCDB6A. A "Connect to DB2 Database" window opens. Select Database alias = PD21R, enter your TSO id and password, [OK]. NOTE: this worked at PG&E.
- Access Table Properties, Description=

ODBC; DSN=CCSP; UID=ABCD; MODE=SHARE; PATCH1=131072; LOBMAXCOLUMNSIZE=1048575; LONGDA TACOMPAT=1; DBALIAS=PD21R;; TABLE=CCSP.CI BATCH CTRL

Link Table to SQL Server

- You can access data in SQL Server tables.
- Table properties, Description =

ODBC;DSN=BRTE;Description=ABC Exception Database ;UID=sa;APP=Microsoft® Access; WSID=WSG0182635; DATABASE=ABCE; Network=DBMSSOCN; TABLE=dbo.Statistics

Edit Data While Input

An Input Mask, a field property which can be set in the table definition, can be used to change lowercase letters to uppercase. For example, PG&E's LAN ID is a four-character code composed of letters and numbers. You can add an Input Mask with the following properties, and then assign it to certain fields:

Page 38 of 119

mask: >AAAA no placeholder sample data: ABC3 mask type: text/unbound.

Validation Rule can be a simple as: = "U" or "M"

Revision: 9/2/2012

Default Field Labels in Forms & Reports

The Caption field property is set in the table definition. When present, it overrides the actual field name as a field label in forms and reports.

Table Definition: Lookup Values with Value List

You can define a value lookup for a table field as a Value List. The appropriate properties are:

DisplayControl = Combo Box RowSourceType = Value List RowSource = "value1", "value2." . .

LimitToList = Yes/No

This causes a combo box to appear as datasheet view. If you add such a field to a form, its control can be defined as a text box, list box, or combo box. If the control is the same type as the table field's DisplayControl and if the field is added to the form after the Value List is created, then the same Value List is operative in the form. If you add the Value List after the field has been added to a form, delete the field from the form and then re-add it.

Table Definition: Lookup Values with Query

You can define a value lookup for a table field using a lookup table where lookup table has ID (code) and text description. The appropriate properties are:

DisplayControl = Combo Box

RowSourceType = Table/Query/SQL statement

RowSource = table/query name

LimitToList = Yes

BoundColumn = # of ID column in row source, usually 1

ListWidth combined column widths

You can define a value lookup for a table field so that the user is suggested to choose an existing value by using a combo box populated with a query like:

SELECT field FROM table GROUP BY field ORDER BY 1

Define Table Fields with Formatting

• You can control the formatting (presentation only, not stored) of fields in a table. For example, concatenate a literal to a text field:

```
Format = !"(Office 1) "@
```

The ! character forces the field to be left-aligned. The characters between the double-quotes are the literal. So any text entered is displayed as {Office 1} xxxxxxx. When you specify formatting at the table level, it carries forward to datasheets, queries, forms, and reports.

You can specify the format of date fields in a report with the control's Format property:

Format = !"(Office 1) "@

This has the advantage of concatenating the text only in that report.

Automatically Set Last Change Date in a Record

Code to do this can be put in an event procedure associated with a form control. You can use the form.Dirty event to set the value of the field. Or the form.BeforeUpdate event.

Revision: 9/2/2012 Page 39 of 119

Hyperlinks

Per Microsoft: If you create a table by importing data, Microsoft Access automatically converts any column that contains URLs (an address that specifies a protocol such as HTTP or FILE and a full filename or website address, for example: http://www.microsoft.com/) or UNCs (a name with the syntax \\server\share\path\filename) paths into a hyperlink field. Access will convert the column only if all the values start with a recognized protocol, such as "http:" or "\\." If any value starts with an unrecognized protocol, Access won't convert the column into a hyperlink field.

But what about an existing table?

In an Access table a valid hyperlink has the form displaytext#address#subaddress#screentip where the address part either includes the protocol (e.g., "file:") or is a UNC. When a field has data type Hyperlink, after you import data into it, the hyperlink has no address (href) and consequently has no effect. If you run a query like:

```
UPDATE CandidateFiles SET URL = URL + "#" + URL + "#" the hyperlink becomes operative. This will not work if the field's data type is Text.
```

If the value of a Text field has the form displaytext#address#subaddress#screentip, all the text appears as the field value and there is no operative hyperlink. When the field's data type is changed to Hyperlink, then the hyperlinks looks and operates correctly.

Hyperlinks in a Table Field: Open Any File

Say you want to let the user scroll through a list of files and be able to open them one by one. You might want to do this in conjunction with a program that reads all the files in a directory structure.

In this case get the filenames into a table. The table has an additional field, say URL, with a Hyperlink data type. The URL field can be populated programmatically by concatenating the path and filename, e.g.,

```
url = p & "\" & fno
Bracket the actual URL with #, e.g.,
    #c:\Data\Working Papers\Proposal.doc#
```

Then you can open the table in datasheet view and click on the URL field and—voila—the file opens. When the form is the data source for a form, then the corresponding form field is also a hyperlink.

The complete code to populate the table is:

DoCmd.SetWarnings True End Sub

Resetting Autonumbers to Zero

If a table is empty when the database is compacted, the autonumber fields are reset to 0. Should the table be used as the authority on some numbering sequence, you might preserve the highest-numbered record.

Fixing a Table with an Autonumber Key

When you mess up a table whose key is an autonumber data type, especially when the key has a corresponding foreign key in another table, what to do? Say you deleted a record whose key is present in the other table, and because it is an autonumber field you cannot add a new record with the old key value. This can be fixed:

Starting point: Table A is missing a needed record; its key is an autonumber field.

- 1. Design table A: change key data type from autonumber to long integer.
- 2. Create/edit table A record to recreate missing record with original key value.
- 3. Copy table A to table B.
- 4. Empty table A.
- 5. Design table A: change key to autonumber.
- 6. Append table B records to table A.
- 7. When you are sure all is well, delete table B.

Making Yes/No Type Fields Appear with Checkbox

Microsoft Access uses a check box control as the default control for the Yes/No data type in datasheets (both tables and queries), forms, and reports. The check box control is used when you define a table with the user interface (Design View). If you use DDL to add a field to a table (with ADD COLUMN), the check box control is NOT used.

Revision: 9/2/2012

Page 41 of 119

Form Stuff

Types of Forms

There are several types of forms based on their content, operation, and use:

Type of Form	How Defined/Opened	Use	Interaction with Code
window	Access form, opened manually or in code; several may be open at the same time, user can make manually each active in turn	to present data and processing controls	form and window events can run code; code in a separate module can interact with this form by opening it, changing its properties and data values, and/or closing it
dialog box	defined as an Access form, opened in code with DoCmd.OpenForm with the acDialog window mode parameter; form stays on top ¹	to present settings to control a future process so the user can change the settings	all processing stops until user closes the box
message box	opened in code with the MsgBox statement, parameters control contents; form stays on top	to present brief messages and/or to have user choose between two or three alternatives	all processing stops until user closes the box
input box	opened in code with the InputBox statement, parameters control contents; form stays on top	to capture text for future use	all processing stops until user closes the box
file dialog box	opened in code with the FileDialog object (in the Microsoft Office 11.0 Object Library); form stays on top	to select a file for opening or saving, or to select a path for future processing	all processing stops until user closes the box

Application Forms

A database application typically opens with a main form that I often call "Application Central." It has form controls that operate the various transactions. I prefer to use command buttons for these controls and label them as clearly as possible; that way the form can inform the user just what can be done.

¹ No other form can be made active while a dialog box is open.

- This main form should be opened automatically when the database opens. Do this with menu <u>Tools, Startup</u> which opens the "Startup" dialog box. Provide an application title and name the startup form.
- The startup form is the place to locate code that initializes global variables.

Form-Process Interaction

Typical Access applications employ forms to present data and provide mechanisms with which the user can run standalone processes. There are a number of situations where a process might interact with a form. Some of these are discussed in later sections:

- change values of form properties (see page 47)
- change values of form text controls
- use a popup form to control data on a main form (see page 77)
- use a form to accept selection criteria and pass it to another form or report (see page 78)

Other uses include:

- present progress information on a separate form (open a form, keep on processing, write to that form intermittently, then close it) (see page Error! Bookmark not defined.)
- present short messages with a message box (see page Error! Bookmark not defined.)
- capture limited text with an input box (see page Error! Bookmark not defined.)
- use a File Dialog box to capture path and filename (see page Error! Bookmark not defined.)

There are some form properties which affect how a process can interact with the form:

PopUp: A form that stays on top of other windows and the form's toolbar isn't available (so you cannot change the view). A pop-up form can be modal or modeless. The PopUp property can be set only in form Design view. The PopUp property by itself only affects how a process can close the form: the close code should name the form (and not assume it is the active one).

Modal: A form that requires the user to take some action before the focus can switch to another form. Dialog boxes are usually modal. When a form opens as modal, you must close it before you can move the focus to another Access object. When the form is modal, you can't switch to Datasheet view from Form view, although you can switch to Design view and then to Datasheet view. To disable menus and toolbars in addition to other forms, set both the Modal and PopUp properties to Yes.

When a form is opened in code, you use the DoCmd.OpenForm statement described on page Error! Bookmark not defined. It has a window mode parameter which affects how the program can interact with the form. When a form is opened with window mode = acDialog, the code stops execution until the user closes the form. When the form is opened with window mode = acWindowNormal (the default setting), then the code continues execution while the form is open. This is true for modal and modeless forms. If you use a modeless form that is not popup for progress information, be sure that the code closes that form by name, not just the active form (as the user can make another form active while the process is running).

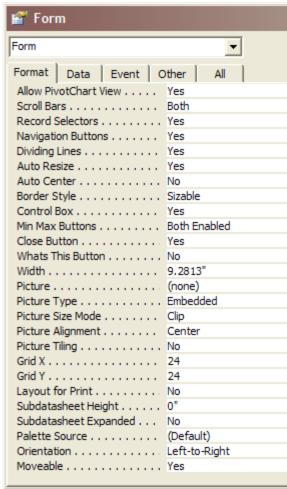
[I wrote a note in the margin: execution resumes when second form closes. – pause – But now I don't know where to put the note.]

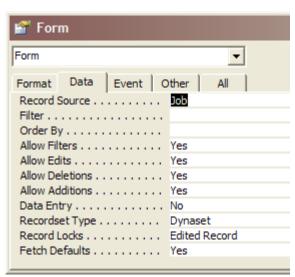
While a form is open and code is running, the code can change the contents of the form (such as text controls but also control properties), and the changes are effective immediately.

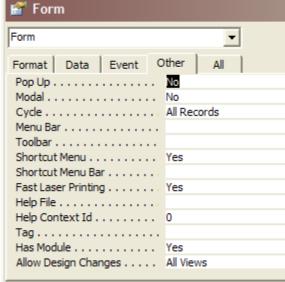
Useful Form Properties

The following images reflect all the form properties.









Useful properties:

Revision: 9/2/2012

Category	Property	Effect
Format	Caption	Appears in title bar.
	Record Selectors	Column at left of form with icons to indicate editing status.
	Navigation Buttons	Built-in record navigation buttons at bottom of form.
	Min Max Buttons	If either or both buttons in title bar are enabled (if disabled, they are greyed out).
	Close Button	If X button in title bar is enabled (if disabled, it is greyed out). If not you must have a custom control to close the form. A form can also be closed with [Ctrl+F4].
	Control Box	If there is a Control menu, which is accessed via an icon at the left side of the title bar. If there is no Control menu, the Control icon, Min Max buttons, and Close buttons do not appear in the title bar. The Control menu has items: Move, Size, Minimize, and Maximize.
Data	Record Source	Identifies the table or query whose data is presented in the form.
Other	Pop Up	If form remains on top until closed. A pop-up form stays on top of other open forms, even when another form is active?
	Modal	If form must be closed before you can move the focus to another object.
	Cycle	What happens when you press [Tab] and the focus is in the last control on a bound form. Can be All Records (moves the focus to the first control in the tab order in the next record), Current Record (moves the focus to the first control in the tab order in the same record), or Current Page (moves the focus back to the first control in the tab order on the page).
	Shortcut Menu	If form or any of its controls has a context menu (accessed with a right-click).
	Shortcut Menu Bar	Name of the context (shortcut) menu for the form and all of its controls. A shortcut menu can be any command bar whose Type property is set to Popup. Leave blank for the built-in menu. (Create a context menu with menu Tools, Customize then tab Toolbars.) Assign a custom menu to a control with its Shortcut Menu Bar property.
	Menu Bar	Name of the menu bar to appear when the form has the focus. Can be a custom menu bar or the built-in bar (leave blank).

Change Form Properties

After the form is created, you can manually change its properties. Use menu <u>Edit</u>, <u>Select Form</u>, then <u>View</u>, <u>Properties</u> to open the Properties window. Some properties can be changed in code while the form is open.

Form Control Scrollbar

A text box control can be defined with a vertical scrollbar using the ScrollBars = Vertical property. Be aware that the scrollbar will only appear if (1) the text box has the focus and (2) all margins = zero. If you want the scrollable text box to appear with the scrollbar visible when the form opens, then set the text box properties as follows:

```
TabStop = Yes
TabIndex = 0.
```

Refresh Contents of Form

Me.Refresh works only if the form has a RecordSource.

This seems to require both a Refresh and a Requery.

• In the form's class module, add this code after the code that changes the contents of the table(s) that populate the form:

Me.Refresh

• A control button can be used with an event procedure (applies only to current form):

```
DoCmd.RunCommand acCmdRefresh DoCmd.Requery
```

• When form B is used to edit records on form A so that they no longer qualify for form A, its AfterUpdate event procedure can refresh the contents of form A:

Forms![UnassignedExceptions].Requery

- When a record is deleted from a form, the contents of the form are automatically refreshed.
- The Recalc method of a form immediately updates all calculated controls on the form. (A calculated control is one whose data source is an expression.) The method could be used in an AfterUpdate event procedure.

Me.Recalc

• When form B adds a record displayed in a subform on Form A, and formB is opened as a popup modal window from formA:

```
RunCommand acCmdSave ' saves new record

DoCmd.Close ' closes formB, leaving formA with focus
RunCommand acCmdRefresh ' refreshes display on formA
```

Requery-refresh form with many list boxes and/or combo boxes.

```
Public Function fRequeryRowSources(frm As Form)
'Requery all combo and list boxes on the argument form
'and on any subforms it may contain, and any of their
'subforms, and so on.

Dim ctl As Control
With frm
For Each ctl In .Controls
Select Case ctl.ControlType
Case acListBox, acComboBox
ctl.Requery
```

```
Case acSubform
If Len(ctl.SourceObject) > 0 Then
fRequeryRowSources ctl.Form
End If
End Select
Next ctl
End With
End Function
```

Refer to Form Controls

See also Tab Control on page 63.

Refer to current open form:

Me

Refer to a named open form:

```
Forms!formname
Forms![form name]
Forms(strFormName)
```

Refer to control on current open form:

Me.fieldA

Refer to a control on a open form:

Forms!formname!controlname

Refer to a field in a subform (a form embedded by a subform control)

```
Forms!formname!ctlSubForm!fieldA -- faster
Forms!formname!ctlSubForm.Form!Controls.fieldA -- slower
Forms!formname!ctlSubForm.Form.controlA
Me.subformA!listboxcontrolB.Column(2) -- value of the third column
```

Refer to the original value of a field on current open form:

Me.fieldA.OldValue

• Refer to the current value of a field on current open form:

Me.fieldA.Value

Is Form Open

• Access 2000: The IsLoaded property of the AccessObject object. Example:

```
Dim objCP as Object
Set objCP = Application.CurrentProject
If objCP.AllForms!formname.IsLoaded Then . . .
or simply
If CurrentProject.AllForms("Main").IsLoaded = True Then . . .
```

Multiple Instances of Forms

When you open a form, you are creating an instance of that form. It is possible to open a form more than once, thereby creating multiple instances.

A form opened by a user from the Database window or DoCmd.OpenForm in code is called the *default form,* because the object instance is unique to the application environment. Only one instance of a specific form can be open at a given time if any of these invocation methods are used:

Opening the form from the Database window Opening the form with DoCmd.OpenForm Using CreateForm to build a new form Using a Form type variable to refer to the instance

- In order to have multiple instances a form must have a class module. And to keep the form alive until the user closes it, at least one variable that references it must be defined in the Declarations area of a form that will outlive it, say the Main form. Form instances perish when the variables referencing them go out of scope. See Object References page 4. See also http://support.microsoft.com/kb/198465.
- Create the form instance using a module-level variable. Adding the "Form_" prefix to the form class name is a VBA "feature" that tells the compiler you are trying to instance a form object¹.

```
Dim frmA As Form_frmIceCreamPopup
Set frmA = New Form_frmIceCreamPopup
frmA.Visible = True
' best if in the Declarations area
```

• You can use the "Form_" prefix to open a form and refer to a method or property of that form or one of its controls in one step. You refer to the form's class module as shown in the following example:

```
Form_Employees.Visible = True
Form Employees.Caption = "New Employees"
```

When you run this code, Microsoft Access opens the Employees form in Form view if it's not already open and sets the form's caption to "New Employees." The form isn't visible until you explicitly set its Visible property to True (–1). When the procedure that calls this code has finished executing, this instance of the form is destroyed, i.e., the form is closed. But another source says that because there is no attached object variable, it does not go out of scope when the procedure ends so you must close the form itself. Furthermore should there be more than one such instance, it is harder to address each one directly.

• If you will have several instances of the same form, you'll need a way to address each easily. You could create an array of form classes. Be aware that an array of objects, which holds the actual data, is resource intensive and should be used wisely.

```
Private mfrmMultiInst(3) As Form_frmMultiInst ' in form's Declaration section
. . .
Private Sub cmdOpen_Click()
' Purpose: Open multiple form instances
Dim iintLoop As Integer
For iintLoop = 0 To 3
Set mfrmMultiInst(iintLoop) = New Form_frmMultiInst
mfrmMultiInst(iintLoop).Caption = "Instance " & CStr(iintLoop)
mfrmMultiInst(iintLoop).Visible = True
Next iintLoop
End Sub
```

You can use a Collection object to group several related objects together into a container where they can be subjected to bulk operations. The only information the Collection object holds is references (pointers). Working with a Collection object is easy and involves the following three methods:

¹ The syntax for addressing form class objects with Form_formname was introduced in Access 95.

- Add method inserts an object reference (called a member) into the collection. It has
 parameters: Item, Key, and Before or After. Item provides the object reference. Key
 provides a unique string value to use when referring to the item, a "primary key";
 without a Key value, an element must be retrieved by its position in the collection.
- Item method returns an object reference from a specified location in the collection. The
 location is referenced by either a positional index number or a unique identifier called a
 key.
- Remove method removes a reference from the collection, either by positional index or key.

```
Public colForms As New Collection ' in a Declaration section colForms.Add After:=3, Item:=rstWidget, Key:="Widget1" colForms.Add Forms!frmWidget ' a specific form is added colForms.Add Item:=Forms!frmWidget!txtWidgetName ' a form control is added colForms.Add Form_frmWidget!subComponent.Form.txtSupplier ' a subform control colForms.Add Me.RecordSetClone ' a recordset is added Widget = colForms.Item("WidgetName") ' retrieve a named item colForms.Remove(7) ' remove item by position colForms.Remove("WidgetName") ' remove named item
```

• Another way to open an instance of the form is as follows:

```
Dim frm as Form
Set frm = Form Employees
```

Approach to Editing a Table With a Form

 The form can present a single record or multiple records at a time. Data is presented in text controls, most of which allow direct editing.

Style of Form	Records	Description	Record Navigation	Comments
columnar	one record at a time	you can layout the field controls any way you prefer	needed; the built-in navigation controls may be used	text boxes for long fields can have vertical scroll bar (only visible when control has the focus)
tabular	several records at a time	each record's field controls (text boxes) are arranged to suit on one or more lines, but the field labels are presented as column headings	can use built-in navigation controls and/or vertical scroll bar	column widths cannot be changed by user; all fields in fixed-height text boxes, no vertical scroll bar for text longer than box, however you can move cursor through all text; you can style text

Style of Form	Records	Description	Record Navigation	Comments
datasheet	several records at a time	grid, the same as if you had opened the table/query	uses built-in navigation controls	why bother with a form? for the form events?
embedded datasheet	several records at a time	like tabular	can use built-in navigation controls and/or vertical scroll bar	embedded in subform; otherwise like tabular
list box	several records at a time	grid	vertical scroll bar	record id must be included; column widths cannot be changed by user; text that extends beyond column width is invisible; memo fields cannot be included; you cannot edit data in the list box, however you can provide a double-click event which opens a second form for editing that single record (requires the source have a single field as unique key—only one column can be bound)
multi-page	one record at a time	like columnar	like columnar; vertical scroll bar provides access to pages	additional pages are created by adding a page break; [Page Down] and [Page Up] move between pages; not appealing
tabbed	one record at a time	like columnar	like columnar	pages created for tabs on tab control

Creating a multi-page form requires a number of particular steps:

When creating a multipage form, remember a few important steps:

- . Set the Default View property of the form to Single Form.
- . Set the Scrollbars property of the form to Neither or Horizontal Only.
- . Set the Auto Resize property of the form to No.
- Place the Page Break control exactly halfway down the form's Detail section if
 you want the form to have two pages. If you want more pages, divide the total
 height of the Detail section by the number of pages and place Page Break
 controls at the appropriate positions on the form.
- . Size the Form window to fit exactly one page of the form.

also set Cycle property is set to Current Page.

The limitation of the list box to present memo fields can be overcome if you use a query that selects only a fixed number of characters from the memo field, such as with Left(*memofield*, 40).

- You use navigation controls to access the other records in the table.
- The simplest approach is to present the records in a datasheet format and use the default navigation controls. This works for a simple table with no foreign keys and no child table. If you take this approach, be sure to provide directions for using standard Access navigation buttons, how to save a record, and how to undo changes.
- One or more requirements may necessitate using control buttons to add, delete, and change individual records. Possible requirements: different security for different users, fool-proofing. The actual data can be displayed in a pseudo datasheet format, all protected. Actions can be initiated by control buttons and double-clicking: An existing record can be selected for editing by double-clicking its key field; this opens a specialized form in Edit mode (form property Allow Edits = Yes, all others = No). A new record can be added by clicking an Add button; this opens a specialized form in Add mode (form property Allow Additions = Yes, all others = No). An existing record can be deleted by clicking a Delete button; this runs an event procedure class module that uses the DoCmd.DoMenuItem statement (built with a wizard) or the DoCmd.RunCommand acCmdDeleteRecord (more readable). See "Setting Field Value" and "Refresh Contents of Form" for details. Alternative to using a specialized form is to use a basic form tailored with VBA code, e.g, DoCmd.OpenForm "Portal", , , , acFormAdd
- Typically you use the DoCmd.RunCommand method which has one parameter, the acCommand constant. Frequently used values:

Constant	Meaning
acCmdDeleteRecord	delete current record
acCmdFind	presents Find dialog box
acCmdSaveRecord	save current record
acCmdUndo	undo changes to current record;
	BETTER: Me.Undo ** see page 77
acCmdRecordsGoToFirst	
acCmdRecordsGoToLast	
acCmdRecordsGoToNext	
acCmdRecordsGoToPrevious	

• An input mask can be used to guide the user. But the error message refers to the input mask itself, which is arcane. Best avoided.

- When the table being edited has a foreign key and you want both the id code and the text value from the lookup table to appear on the form, use a list box with a row source = query/table showing all values in the lookup table, and indicate which list box column is to be bound to the main table field (the id field).
- You may want to use two forms A and B. On formA use a list box to show all the table records; use a query to include foreign key text values. User selects an Add button to add a record, a Delete button to delete a record. To edit a record, user double-clicks its row. The Add button opens formB so that only one record is added, after which it requeries formA's list box and closes itself. The edit action opens formA so that only the selected record is presented; after the record is updated, the form requeries formA's list box and closes itself. The delete action confirms the delete, then builds a delete query and runs it. The sample code applies to the edit action; be sure to block additions and deletions.

```
DoCmd.OpenForm "formname", acNormal, , strWHERE
Forms!formname.AllowAdditions = False
Forms!formname.AllowDeletions = False
```

Be sure the variable strWHERE is created properly: Dim in the current Sub or Private/Public in the Declarations. If not, the argument is no-op.

Forms for Parent-Child Tables

A simple way to design a form to present data on parent and child tables is to use a form for the parent table fields with a subform for the child table fields. The parent form presents a single record with built-in record navigation. The subform contains a datasheet, which automatically has the built-in record navigation controls.

If you want the parent form to present a total of the amounts in the subform datasheet, this is done as follows:

- In the subform put a textbox in the footer with source = Sum(AmountCharged). Call this textbox txtSumCharged.
- In the parent form put a textbox with source = [subService].[Form]![txtSumCharged]. "subService" is the name of the control that contains the subform. Voila!

This approach is easily extended to a family of three tables, each in a 1:M parent-child relationship with each other:

```
A has a 1:M relationship with B
B has a 1:M relationship with C
```

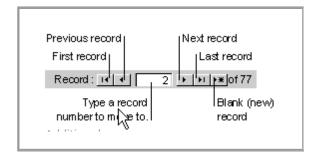
Use a parent form for A, a single-record subform for B, and a datasheet subform for C. Embed C's subform in B's subform, then embed B's subform in A's form. Let each have built-in record navigation.

Navigation on Single Record Form

Access has built-in navigation controls that are consistent to use BUT require training. They are located at the bottom of the datasheet or form. The controls can be organized into three groups:

- record navigation
- start new record
- record *x* of *y* where y = total record count, x = current record number

Revision: 9/2/2012 Page 52 of 119



You can replace these with custom controls.

To disable the built-in navigation controls, set form properties:

```
NavigationButtons = False
```

Sample code for [New]:

```
DoCmd.GoToRecord , , acFirst
DoCmd.GoToRecord , , acNewRec
```

Sample code for [Next]:

```
DoCmd.GoToRecord , , acNext
```

It may be that you have to check to be sure you are not at the end when this code is run.

Sample code for [Previous]:

```
DoCmd.GoToRecord , , acPrevious
```

It may be that you have to check to be sure you are not at the end when this code is run.

Sample code for [First]:

```
DoCmd.GoToRecord , , acFirst
```

It may be that you have to check to be sure you are not at the end when this code is run.

Sample code for [Last]:

```
DoCmd.GoToRecord , , acLast
```

It may be that you have to check to be sure you are not at the end when this code is run.

Sample code for total record count:

```
SELECT Count(*) FROM . . .
```

Return current record number:

```
Me.CurrentRecord
```

but only works on the recordset (table or query) that the form is based on. It may not reflect a later sort. [needs work]

You likely do not want the navigation controls enabled all the time. For instance, if there is only one record then only the New control should be enabled. You especially want to avoid running the Next code when you are at the end, because doing so will cause a run-time error.

Recognizing when you are at the beginning or end of the form's record source is possible with the Recordset property of the Form object. It returns an ADO or DAO Recordset object which has the necessary properties—BOF (beginning of file) and EOF (end of file)—which can be tested.

```
If Me.Recordset.EOF Then . . . Else . . . End If If Me.Recordset.BOF Then . . . Else . . . End If
```

The following table summarizes the various conditions you can expect to encounter and when to enable or disable each control. In each case the New control can be enabled.

Current Position	First	Previous	Next	Last
empty recordset	disabled	disabled	disabled	disabled
1st record of 1	disabled	disabled	disabled	disabled
1st record of 2	disabled	disabled	enabled	enabled
2nd record of 5	enabled	enabled	enabled	enabled
5th record of 5	enabled	enabled	disabled	disabled

In the following discussion be careful how you refer to the form object. If code is in the form module, "Me" works. Otherwise pass the form name to a standard module procedure which will use "Forms(strForm)".

The controls can be enabled and disabled by various events:

Event	Action	Sample Code
form.OFalsepen	if record count < 2, then disable all controls	<pre>If DCount("*", "table or query name") < 2 Then form.cmdNext.Enabledd = False form.cmdLast.Enabledd = False form.cmdFirst.Enabledd = False form.cmdPrevious.Enabledd = False ElseIf on first record Then form.cmdFirst.Enabledd = False form.cmdPrev.Enabledd = False End If</pre>
form.cmdFirst	go to first record disable First and Previous, Enabled Next and Last	DoCmd.GoToRecord , , acFirst form.cmdFirst.Enabled = False form.cmdPrevious.Enabled = False form.cmdLast.Enabled = True form.cmdNext.Enabled = True
form.cmdPrevious	go to previous record Enabled Next and Last if at beginning of recordset, then disable First and Previous, else Enabled them NOTE: Use of BOF does not work in standard module. There use form.CurrentRecord = 1.	DoCmd.GoToRecord , , acPrevious form.cmdNext.Enabled = True form.cmdLast.Enabled = True If form.Recordset.BOF Then form.cmdFirst.Enabled = False form.cmdPrevious.Enabled = False Else form.cmdFirst.Enabled = True form.cmdPrevious.Enabled = True form.cmdPrevious.Enabled = True End If

Event	Action	Sample Code
form.cmdNext	go to next record Enabled First and Previous if at end of recordset, then disable Next and Last, else Enabled them NOTE: Use of EOF does not work in a standard module. There use form.CurrentRecord = DCount(*, form.RecordSource)	DoCmd.GoToRecord , , acNext form.cmdFirst.Enabled = True form.cmdPrevious.Enabled = True If form.Recordset.EOF Then form.cmdNext.Enabled = False form.cmdLast.Enabled = False Else form.cmdNext.Enabled = True form.cmdLast.Enabled = True form.cmdLast.Enabled = True
form.cmdLast	go to last record disable Next and Last, Enabled First and Previous	DoCmd.GoToRecord , , acLast form.cmdNext.Enabled = False form.cmdLast.Enabled = False form.cmdFirst.Enabled = True form.cmdPrevious.Enabled = True

Place Cursor When Form Opens

This is done by using the method SetFocus for the desired field. It's Enabled property must
 = True/Yes.

Forms! Employees! EmployeeID. SetFocus

• This can be done in a form's Open or Current¹ event procedure:

=Forms!Employees!EmployeeID.SetFocus

• It can also be done in a macro? Not directly. A macro can run a function (via the RunCode action) that executes the method.

Save and Undo

Save: by default Access saves changes to the current record when you navigate to a different record. When you use custom navigation you may want to have an explicit save that retains the current record on the form or subform:

DoCmd.RunCommand acCmdSaveRecord

Undo: Changes to the current record may be discarded instead of saved. Access calls this "undo." The code is:

```
Me.Undo ' clear all changes to the current record Me!txtDetails.Undo ' clear changes to named control
```

- There are two kinds of changes in a form: (1) When current data is changed, the form's Dirty property is set to True. (2) When a new record is initiated, the form's NewRecord property is set to True.
- Changes in a subform are not known to the main form. They must be handled separately. When the subform holds a datasheet, the Dirty property does not correctly reflect changes in current data. The NewRecord property is correct. The basic problem is that when an [Undo]

¹ A form's OnCurrent event happens when the form opens, is refreshed, requeried, or the record changes.

button is on the main form, clicking it causes the focus to leave the subform and any changes there are saved.

For a form:

```
Me.Undo
Forms!("formname").Undo
DoCmd.RunCommand acCmdUndo
```

For a control:

Me!controlname.Undo

• For a subform (only works when done within the subform). Actually I am not sure about this code because if it is in the subform module, Me refers to the subform, so why the need to address the subform control? Why not just Me.Undo?

Me! subformcontrolname. Form. Undo

Programmatically undoing data entered by a user will cause a run-time error if there is nothing be be undone. Each condition (Dirty and NewRecord) should be tested. If there is a subform, changes to existing data are saved when a control on the parent form is selected, thus the only "change" that can be "undone" by the parent form is the new record.

```
If Me.Dirty = True Then
    Me.Undo
ElseIf Me.NewRecord = True Then
    DoCmd.RunCommand acCmdRecordsGoToPrevious
End If
```

Protect Form Field From Update

Set field's properties:

```
Enabled = Yes (can have focus, appears normally)
Locked = Yes (no editing)
TabStop = No (tab skips over field)
```

Miscellaneous Form Control Properties

- To designate a control button to be activated on [Enter]: set that control's Default property = Yes.
- To designate a control button to be activated on [Esc]: set that control's Cancel property = Yes.
- To put cursor on a given control when form event occurs:

```
Forms! [form name] . SetFocus
```

You can hide and/or unhide a form control in code:

```
Me![control name].Visible = No ' hides the control
Me![control name].Visible = Yes ' unhides the control
```

You can layer controls (place one beneath another) with menu <u>Format, Send to Back</u>. This can only be done in Design View. You can in code hide the controls on top while unhiding the controls on the bottom using the Visible property.

Command Buttons

Typical buttons are [OK] and [Cancel]. If you want a button to be the default and activated by the [Enter] key (in addition to clicking it), then set property Default = Yes. If you want a button to cancel the current operation and close the window and you want the [Esc] key to activate it (in

addition to clicking it), then set property Cancel = Yes. Remember these buttons still need an OnClick event handler.

Option Buttons

- Option buttons may be part of an option group. Each is its own control. Apparently the relationship is established by physical location: the buttons are located within the rectangular frame of the group, there is no property that refers to the relationship.
- For the option buttons in an option group to be selectable, the form must have property AllowEdits = Yes.
- How to test if a particular option button is selected: compare the option group's Value property to the option button's OptionValue property.

```
If [Date Option Group].Value = [Date Range].OptionValue Then . . .
```

• Refer to the option group:

Me!optRecordOrder

Refer to an individual option button:

Me!opt1ID

You can set the focus on the option group but not an option button:

Me!optRecordOrder.SetFocus

You can act on the selected option button with a SELECT CASE structure:

```
Select Case optRecordOrder.Value
Case 1    ' Latest History
    lstJobs.RowSource = "AllJobs"
Case 2    ' Employer-Title
    lstJobs.RowSource = "AllJobsEmployerTitle"
Case 3    ' Title-Employer
    lst Jobs.RowSource = "AllJobsTitleEmployer"
Case 4    ' ID
    lstJobs.RowSource = "AllJobsID"
Case Else    ' this would be an error
    lstJobs.RowSource = "AllJobs"
End Select
```

Memo Field in Text Box

The standard way to present a memo field (up to 65,000+ characters of text) is with a multi-line text box with a vertical scroll bar. The scroll bar only appears when the control has the focus. The size of the text box is fixed when viewed online. If the property Can Grow = Yes, then when the form is printed, the size of the text box increases to present all the text. You can zoom in on such a text box: when it has the focus press [Shift+F2] to open the "Zoom" window, which can be closed easily with [Esc].

Combo Box

- Drop-down box with text box at the top of a pre-defined value list. Can have vertical scroll bars. Value may be typed or selected from list. No multi-select allowed. When not open (dropped-down) the height of the box is one line. Down arrow is full-size in one line.
- The form must have property AllowEdits = True in order to select a row in the list.
- Relevant objects, methods, and properties:

<u>AutoExpand</u> property controls whether the typed value is automatically completed by a matching list entry. Best when list entries are text, not numbers.

<u>Column</u> property references a numbered column in a multi-column list. Has (column, row) format where (0, 0) is the first column in first row. Value is Null when there is no selection. Can be expressed as Column(x) to refer to the entire column.

<u>ColumnCount</u> property specifies number of columns in the box.

<u>ColumnWidths</u> property specifies width of each column separated by semicolon. You can hide a column by making its width = 0." Example: .3";1.5" and .3";0";1.5"

<u>DefaultValue</u> property specifies list entry to be used as default format. Value is like: "=Null"; the Null value has the advantage that it will not be accepted as a valid value when LimitToList is used. If no value is specified, "0" appears and, even when LimitToList is True, is accepted as a valid value.

<u>DropDown</u> method forces the named combo box to drop down. Typically used when something happens in another control.

Format property can customize way in which list data is displayed.

<u>IsNull</u> function determines if any row is selected: If IsNull(Forms!formname!controlname) Then . . . <u>ItemData</u> property returns the value in the bound column in the list; uses zero-based index (first entry is 0). I was unable to use this property as a default value. Example: ItemData(5) refers to the value in the numbered row.

<u>LimitToList</u> property controls whether entered value may be different from value list. "True" by default when first column is not bound. "True" does not keep a combo box from accepting a Null value; the way to disallow Null is to set the Required property of the field in the table to which the combo box is bound to True.

<u>ListCount</u> property returns the number of rows in the list.

<u>ListIndex</u> property returns relative position (index) of the selected entry in the list. First entry is 0. Read-only.

<u>ListRows</u> property specifies maximum number of rows to display in drop-down list.

<u>ListWidth</u> property specifies width of drop-down box. It is advisable to set this to the combined column widths as the default value is likely to be narrower.

OnNotInList property specifies name of procedure that will run automatically if entered value is not in the list.

<u>RowSourceType</u> property specifies values in the list as Table/Query, Value List, or Field List. For Value List, values are entered in property <u>RowSource</u> as "Open", "Closed" . . .

<u>Text</u> property sets or returns the text in the text box portion of the control. You can set the Text property to the text you want to display in the control. You can also use the Text property to read the text currently in the control. Note: to set or return a control's Text property, the control must have the focus, or an error occurs.

<u>Value</u> property returns the text in the text box portion of the control after the bound record has been updated. Hopefully, it also returns the text in an unbound column after the focus leaves it.

- To access selected list entry: Use Value property.
- To determine if entry selected (this is most useful when the combo box's <u>DefaultValue</u> property is set to Null):

If IsNull(Forms!formname!controlname) Then . .

• To set value of a second control (controlB) based on the value selected in the combo box (controlA); requires the second control be physically located after the combo box:

Me!controlB = Me!controlA.Column(2, n)

List Box

- List box is a fixed-height box containing a list of data in rows, no drop-down. Can have vertical and horizontal scroll bars. Multi-select allowed. If height is only one text line, down arrow is tiny.
- The form must have property AllowEdits = True in order to select a row in the list.
- Relevant objects, methods, and properties:

BoundColumn Links the records in the list box to the record source of the form.

<u>ColumnHeads</u> When True the first line in the list box appears as column headings.

<u>ColumnWidths</u> property specifies the width of each column in the list box separated by semicolons. In America, the unit of measure is inches. A <u>column can be hidden</u> by setting its width to zero; while it is not visible in the list box, it is still available to Access (and code).

Example with three columns: 1"; 2"; 0.4". Example with a hidden column: 0"; 1"; 1".

<u>Column</u> property references a numbered column in a multi-column list. Has (column, row) format where (0, 0) is the first column in first row. Value is Null when there is no selection. Can be expressed as Column(x) to refer to the entire column.

<u>ItemData</u> property returns the value in the bound column in the list; uses index. Example: ItemData(5) refers to the value in the numbered row.

<u>ItemsSelected</u> property returns a reference to the collection of the same name. The collection contains variants, each an integer index referring to a selected row. (0) is the first member in the collection. Read only.

<u>ItemsSelected.Count</u> property returns the number of indexes in the collection.

<u>ItemsSelected.Item(n)</u> property refers to the numbered row in the collection. It is optional as you can achieve the same end by ItemsSelected(n). Read-only.

<u>ListCount</u> property returns the number of rows in the list.

<u>ListIndex</u> property returns relative position (index) of the selected entry in the list. First entry is 0. Read-only.

<u>MultiSelect</u> property controls whether more than one entry can be selected and how (there are two methods).

<u>Row Source</u> property contains either the name of a query or table or the SQL SELECT statement that returns the fields for the list box. When a list box is created by the wizard, the latter value is used. Best to replace it with an actual query/table name.

<u>Selected</u> property indicates if the corresponding list item is selected. Example: Me!ctl.Selected(4) = True. Read/write.

<u>Value</u> property returns the value in the bound column for the selected row. Is "-1" if no column is bound.

To pre-select the first data row:

```
Me!lbJobs.Selected(0) = True ' when there are no column headings
Me!lbJobs.Selected(1) = True ' when there are column headings
```

To access selected entry of single selection list:

```
Dim n As Integer
Dim ctl As Control
Set ctl = Me!lbAssignee ' NOTE: the use of Me! is optional
If ctl.ItemSelected.Count = 1 Then
    ctl.ItemData(1) . . .
```

To access each selected entry of multi-select list:

```
Dim varItem As Variant
For Each varItem in ctl.ItemsSelected
```

. . .

Next

To unselect a row in a single-select list box:

```
If ctl.ItemsSelected.Count = 1 Then
    Dim n As Variant
    n = ctl.ItemsSelected(0)
    Me!ctl.Selected(n) = False
End If
```

To unselect all rows in a list box based on a query:

listboxname.Requery

To unselect a row in a multi-select list box: (have not yet proven this in code)

```
Dim varItem As Variant
Dim n As Variant
If ctl.ItemsSelected.Count => 1 Then
    For Each varItem in ctl.ItemsSelected
        n = ctl.ItemsSelected(varItem)
        ctl.Selected(n) = False
    Next
End If
```

- You can populate a list box with a query and set one row-value as DefaultValue. The query can append "all" to its values with a UNION.
- When you use a list box as the way of displaying table/query records for some action by the user (easier than using a datasheet), some form properties are critical:

DefaultView: Single Form

AllowPivottableView: No AllowPivotchartView: No

AllowEdits: Yes the next three enable a click on list box row to select the row

AllowDeletions: Yes AllowAdditions: Yes

DataSource: (none) > unbound

When list box holds a query result with a numeric field, a null value appears blank. When you test the value in code, you can test for a numeric value. Test for the blank value as follows:

```
If (cnt > 0 \text{ And Not cnt} = "") Then . . .
```

When no item is selected in a list box, the value of the bound column is Null. To tell if an item is selected:

```
If IsNull(Me.ProductID) Then . . .
```

- To reorder the contents of a list box provide a separate form control for the user to select one of a group of order choices and apply the selected order by changing the list box's Row Source. This is easiest if you have one query for the default order and additional queries based on it with different orders (e.g., SELECT * FROM AllJobs ORDER BY Employer, Title). One way to present record order choices is in an option group. Code sets the Row Source to the query corresponding to the selected order. See page 57 for an example.
- You can synchronize the contents of a list box to the form's primary record (its record source). This is helpful when you use a list box to present related data from a second table, either a child table or one with a foreign key relationship. This takes three components: (1)

the list box's Row Source is a SELECT statement with a WHERE clause that links a field on the form to a field in the list box, (2) list box's property Bound Column identifies the related form field, and (3) the form's Current event requeries the list box.

Example: The form has Record Source = Recruiter. The list box has Bound Column = ID (the field in the Recruiter table that has a foreign key relationship with the RecruiterID field in the Job table. The list box has Row Source = SELECT ID, Employer, Title, Location, Status, Submitted, Interviewed FROM JobsForRecruiter WHERE

(((JobsForRecruiter.RecruiterID)=Forms!RecruiterWithJobs!ID)) ORDER BY 1; The form's Current event has code: Me.lstJobs.Requery.

 You can present a memo field in a list box only by using a query that selects a fixed number of characters. Example: SELECT Name, Left(Notes, 30) FROM Comment.

Use List Box to Restrict Values to Lookup Table

List box properties:

```
RowSourceType = Table/Query
RowSource = lookup table name
MultiSelect = None
ColumnCount = 2 (code and description)
ColumnWidths = 0.25";1"
BoundColumn = 1
```

List box will appear always, so both code and text will always be displayed.

Populate List Box with Query Result

List box properties:

```
RowSourceType = Table/Query
RowSource = query name
BoundColumn = some column, preferable one that is unique
Enabled = Yes
Locked = No
ColumnCount = number of columns in query
ColumnWidths = 1"; 1"; . . .
MultiSelect = depends
```

Text box can present count of rows selected in list box. List box can have OnClick event that can be used to set value of text box:

```
Private Sub ListBoxUnassigned_Click()
Dim tempCount As Long, lb As Control, lbItem As Variant
tempCount = 0
Set lb = Me!ListBoxUnassigned
If lb.ItemsSelected.Count > 0 Then
    For Each lbItem In lb.ItemsSelected
        tempCount = tempCount + lb.Column(4, lbItem)
    Next lbItem
    Me!txtCount = tempCount
End If
End Sub
```

Populate List Box Containing Names of Queries

- Form has list box with property RowSourceType = function name. Leave property RowSource blank. Property ColumnCount must be correct.
- VB function populates list box.

```
Function ListBoxQueries(fld As Control, id As Variant, row As Variant, col As
Variant, code As Variant) As Variant
' this function is the data source for a list box
Dim strUserName As String
Dim n As Long ' for record count

Dim r As Integer ' subscript for q array

Dim qdfItem As QueryDef ' QueryDef object variable
Dim ReturnVal As Variant
ReturnVal = Null
Select Case code
    Case acLBInitialize
                                        ' Initialize, build array.
       strUserName = GetUserName()
        n = DBEngine.Workspaces(0).Databases(0).QueryDefs.Count
        Static q(70, 3) As String
        For Each qdfItem In DBEngine.Workspaces(0).Databases(0).QueryDefs
            If Not Left(qdfItem.Name, 1) = "~" Then
            q(r, 0) = qdfItem.Name
            q(r, 1) = QueryDefType(qdfItem.Type)
            q(r, 2) = "Updatable for user " & strUserName & ": " &
qdfItem.Updatable
            r = r + 1
            End If
        Next qdfItem
        ReturnVal = r
    Case acLBOpen
                                         ' Open.
       ReturnVal = Timer
                                         ' Generate unique ID for control
(seconds since midnight).
    Case acLBGetRowCount
                                         ' Get number of rows.
       ReturnVal = n
    Case acLBGetColumnCount
                                        ' Get number of columns.
        ReturnVal = 3
    Case acLBGetColumnWidth
                                         ' Column width.
       ReturnVal = "2 in;1 in;1 in"
    Case acLBGetValue
                                         ' Get data.
        ReturnVal = q(row, col)
                                         ' End.
    Case acLBEnd
       Erase q
End Select
ListBoxQueries = ReturnVal
End Function
```

Scroll Down List Box to Show Pre-Selected Row

Scroll a listbox to a specific row. Emulates the VB ListBox TopIndex property. You can alter the code to easily have the selected row display as the first or last row as well. The example code is placed behind a Command Button. From Stephen Lebans www.lebans.com

Private Sub cmdListIndex Click()

```
On Error GoTo Err cmdListIndex Click
' Always make NumRows an odd number
' if you want selected Row to be in the
' middle of the ListBox.
' NumRows is the number of completely visible rows in the ListBox Const
NumRows = 7
' Row we want displayed in middle of ListBox.
Dim intDesiredRow As Integer
' Arbitrarily select the 24th row.
intDesiredRow = 24
' ListBox must have the Focus
Me.List2.SetFocus
' Force ListBox to start from the top
Me.List2.ListIndex = 1
' Force the Scroll offset we desire
Me.List2.ListIndex = intDesiredRow + (NumRows / 2)
' Now select the row without further scrolling
Me.List2.ListIndex = intDesiredRow
Exit cmdListIndex Click:
Exit Sub
Err cmdListIndex Click:
MsgBox Err.Description
Resume Exit cmdListIndex Click
End Sub
```

Tab Control

A tab control is a neat way to present a lot of data in a small amount of space—by layering it and using the tabs themselves to access each layer. There is an Access tab control and there are ActiveX controls.

The following discussion uses the terms:
tab control: refers to the overall control
tab: refers to the rectangle that is commonly positioned at the top
layer: the body of the control that is specific to an individual tab; others call this "page"
tab bar: referns to the bar at the top of the tab control in which the tabs are located
tab text: refers to the text that appears in the tab; the control property is Caption
data control: refers to any control which appears on a layer

Access Tab Control

BackgroundColor can be the default gray or "transparent" which lets the color of the form section be visible. Unfortunately, the latter does not apply to the area of the tab bar where there are no tabs. I was unable to place a rectangle on top of the unused tab bar in hopes of giving that area a different color; even moving the rectangle to the front did not make it visible in the tab bar.

- FixedWidth applies to the width of the tabs. The default width is just enough to hold the tab text; there is some effort to balance the width of the tabs so a tab with short text is similar in width to the tab with long text. You can set a width, like 1.25", to widen the tabs. There is no way to automatically expand the width to fill the width of the overall control.
- You can change the default tab height with the property Tab Fixed Height. It can be visually better to increase it.
- Tab text length. In design view you can pad the actual text with spaces before but not after (you can type a space but it disappears). You could increase the padding in code in the form's OnCurrent event:

```
Me!Page1.Caption = " Job "
```

- Number of tabs is initially 2. Use the context menu to add tabs (<u>Insert Page</u>), remove tabs (<u>Delete Page</u>), and change the text (caption) on a tab (<u>Rename</u>). Alternatively, use menu <u>Insert, Tab Control Page</u> to add a tab. Use the Properties window to set each tab Name (may be best left to the default) and Caption.
- Data tab order: You set the data control tab order within each layer at a time with the context menu <u>Tab Order</u>. There does not seem to be a way to tab between layers, therefor it is advisable to include an access key in the tab text: enter the text like "F&lowers" to produce the result "Flowers" where [Alt+L] makes that layer active.
- You can use the keyboard to move between tabs: [Ctrl+Tab] moves to the next tab page on the right. [Ctrl+Shift+Tab] moves to the next tab page on the left.
- In design mode you place data controls on a tab layer in different ways. If dragging a field from the Field List window, you can drag it directly to the tab layer. If adding a control with the Toolbox window, you can drag it directly to the tab layer. Once on a tab layer you cannot drag a control off to the form itself; instead you must cut (with [Ctrl+X]) the control from the form and then paste it on a tab layer (with [Ctrl+V]). Likewise you cannot drag a control from the form to a tab layer; instead you must cut (with [Ctrl+X]) the control from the form and then paste it on a tab layer (with [Ctrl+V]).
- Useful Tab Control properties: Visible, Style. Useful events: OnClick, OnChange (occurs when you move from page to page, or tab to tab, cannot be canceled).
- Relevant VBA object model: the TabControl object contains two objects: Pages collection and Properties collection. The Pages collection contains the Page object which also contains a Properties collection. The Page object correlates to what I think of as a layer.

A tab control's default property is Value, which returns an integer that identifies the current page: 0 for the first page, 1 for the second page, and so on. Because the Value property is the default property for a tab control, you do not have to refer to it explicitly. By reading the Value property at run time, you can determine which page is currently on top.

You reference the tab control itself as any form control:

You can reference a Page object in three ways:

Me.Pages!pagename
Me.Pages("pagename")
Me.Pages(index)

Because each page is a member of the form's Controls collection, you can reference a page solely by its Name property without referring to the tab control's name or its Pages collection.

```
Me!TabControlName.Pages(0).Caption = "First Page"

Me!Page1.Caption = "First Page" ' Page1 is the tab's name
```

The Page object is also a form control. Controls on a Page object belong to that Page object's Controls collection <u>and</u> to the Controls collection of the form itself. For this reason, each control on a tab control page must have a name that is unique with respect to all other controls on the same form. You can refer to controls on a tab control page by using the same syntax for controls on a form without a tab control.

```
Me!Page1!txtDescription.Value = "Anon"
Me!txtDescription.Value = "Anon"
```

Each Page object has a PageIndex property that indicates its position in the Pages collection. The Value property of the tab control is equal to the PageIndex property of the current (active) page. You can use these properties to determine which page is currently selected after the user has switched from one page to another, or to change the order in which the pages appear in the control. The PageIndex is automatically set to start at 1 and increment by 1 when you use the form designer to create the tab control and the tabs. I cannot see why you would change the values. If a user or code changes a page's PageIndex property, the reference to the page's index and the page's position in the page order change. In this case, if you want to maintain an absolute reference to a page, refer to the page's Name property.

Useful Page properties:	Useful Page methods:	Useful Page events:
Caption	Requery	OnClick
Enabled	SetFocus	OnDblClick
Name		(fires when the user clicks a
Visible (read/write)		blank area or record selector
PageIndex		on the form)

You may want to put a record count in a tab caption, such as "Comments (3)". This is easily done. In the Current event of the main form set the tab captions; this will work correctly when the form is opened and when navigating between records on the main form. If you need the caption to update automatically as data is added or deleted on a subform on the tab, use the AfterInsert event and OnDelete event of the subform to change the tab caption. See "Counting Records in a Subform" on page 71 for details.

```
Private Sub Form_Current() ' in main form

'set the caption of the tabs

Me!Page3.Caption = "History (" &

Me!HistorySubform.Form.RecordsetClone.RecordCount & ")"

Me!Page4.Caption = "Comments (" &

Me!CommentsSubform.Form.RecordsetClone.RecordCount & ")"

End Sub

Private Sub Form_AfterInsert() ' in subform

'set the caption of the tab

Me.Parent!Page3.Caption = "History (" & Me.RecordsetClone.RecordCount & ")"
```

End Sub

```
Private Sub Form_AfterDelConfirm(Status As Integer) ' in subform 'set the caption of the tab

Me.Parent!Page3.Caption = "History (" & Me.RecordsetClone.RecordCount & ")"

DoCmd.GoToRecord , , acFirst

End Sub
```

- Editing data on tabs may be tricky. If the main form is not bound, a bound subform will save the record as soon as a different tab is on top. "Subforms with master/child links in Access 2003 are rather buggy. Subforms on tab controls just add to the complexity."
- You cannot keep the user from clicking on a tab and giving that page the focus. You can have a tab control Change event that conditionally changes back to the previous page:

- One alternative to using the built-in tabs is to set the Style property = None (2) and use in their place labels styled to suit and having a Click event that sets the focus on the corresponding tab page. The current tab label could be chiselled and sunken when not current (SpecialEffect property). Because you cannot disable a label, the Click event will have to test the SpecialEffect property to see if it is current.
- One application which I found on Experts-Exchange: "Many folders went in a box, and many boxes belonged to a project. And every box has a label that has to do with the folders in the box and the project the box is in. In that case, I had four tabs: Projects, Boxes, Folders, Labels. This super-switchboard acted as the only form the user actually interacted with (other than some pop-ups and dialogs, etc.). Once you chose a project on the first tab, you just kept working to the right, tab by tab, until it was time to print the labels."

Microsoft Forms 2.0 Tab Strip

I found this unworkable. The biggest problem I encountered was the inability to place data controls on a tab layer and then see them in Form View. I could also not set the text for any but the first tab. It is likely that I need to know more about the control before being able to use it effectively.

Microsoft TabStrip Control, version 6.0

This looks good. The formatting of the tabs and tab bar are more sophisticated than for Access's tab control. However, using an ActiveX control requires programming.

There is a help file for this control and others in MSComctl.ocx: cmctl198.chm; it is used in conjunction with cmctl198.chi. They are normally installed when MSDN is installed; it may be found in C:\Program Files\Microsoft Visual Studio\MSDN\2001APR\1033. I found the files at $\frac{\text{http://homepages.uel.ac.uk/u0005080/vb/COMMON/HELP/}}{\text{http://homepages.uel.ac.uk/u0005080/vb/COMMON/HELP/}}$

However it does not work at this time. The answer seems to be in comctl1.cnt and comctl1.hlp.

The TabStrip object has a Tabs collection consisting of the individual tabs. Each tab has properties:

index: The position of the tab within the collection (and in the tab bar).

caption: The text string appearing on the tab.

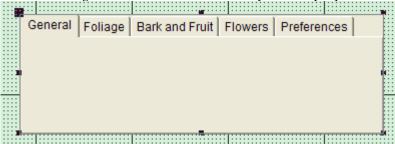
key: A unique string that identifies the Tab. Can be used in code. Optional.

image: The index of an image in an associated ImageList control. Optional.

tooltiptext: to be determined.

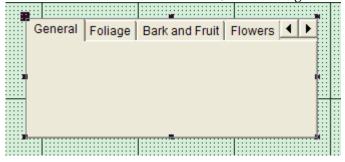
Use the control's context menu <u>TabStrip Object, Properties</u> to open the (custom) dialog box of the same name. This dialog box presents some of the properties that are visible in the standard controls Properties dialog box, but in a more useful manner. Unfortunately you cannot resize the control while the custom dialog box is open.

The formatting of the tab bar is controlled by several properties.

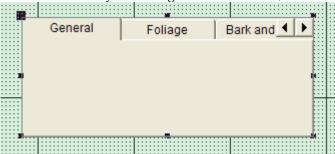


In the preceding example, MultiRow = no, Style = 0 - tabTabs, TabWidthStyle = 1 - tabNonJustified, Placement = 0 - tabPlacementTop, TabStyle = 0 - tabTabStandard, TabFixedWidth = 0, TabFixedHieght – 0, TabMinWidth = 567.

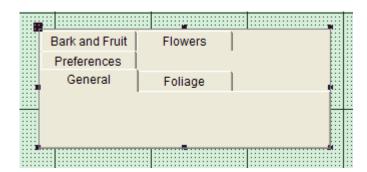
If the width of the control is reduced, the tab bar gets scroll buttons:



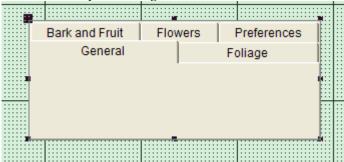
If the TabWidthStyle is changed to 2 – tabFixed, the bar looks like this:



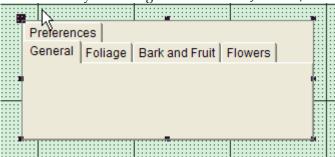
If MultiRow is set on, the bar looks like this:



If TabWidth Style is changed to 0 – tabJustified, the bar looks like:



If TabWidthStyle is changed to 1 – tabNonJustified, the bar looks like:



- Tabs can be added, removed, and re-texted with the Tabs tab of the custom properties dialog box.
- There is no way to change the background color of the control.
- The TabStrip control is not a container. Any contents must be held in a container object. If you use a control array for the container, you can associate each item in the array with a specific Tab object.
- This control should be placed in an unbound form. The contents of the control are held in a container object, the easiest being a subform control. You can place one subform control in the tab strip control; it should hold the form corresponding to the data in the first tab's layer. The tab strip control utilizes an OnClick event handler to replace the contents of the subform control with a different subform when a different tab is selected.

For example: Create a bound form for each tab with no record selectors, navigation buttons, and dividing lines. On the control place a pre-defined bound subform that represents the contents of the first tab: With the Database window in the foreground, drag such a form to a TabStrip control layer. Select and then delete the subform's label. Adjust the size of the tab

strip control to hold the subform completely. Write a TabStrip onClick event handler to swap the contents of each layer based on which tab is clicked, like:

```
' the name of the tabstrip control is TabStripPlant
' the name of the subform control is subContainer
' the names of the forms are General, Foliage
Private Sub TabStripPlant Click()
On Error GoTo errhandler
'Dim rs As DAO.Recordset, mybookmark As String, I As Integer
Select Case Me!TabStripPlant.SelectedItem.Index
  Case 1 'Show General form
     subContainer.SourceObject = "General"
  Case 2 ' Show Foliage form
     subContainer.SourceObject = "Foliage"
End Select
Exit Sub
errhandler:
MsgBox "There is an error"
Customers.SourceObject = "General"
End Sub
```

Note that the Dim line seems extraneous and I commented it out successfully; if you use it you must reference the Microsoft DAO 3.6 Object Library.

• Microsoft offers an alternate method for changing the content of TabStrip control's container: If you use a control array for the container, you can associate each item in the array with a specific Tab object, as in the following example:

```
Option Explicit
Private mintCurFrame As Integer' Current Frame visible

Private Sub Tabstrip1_Click()
   If Tabstrip1.SelectedItem.Index = mintCurFrame _
        Then Exit Sub ' No need to change frame.
   ' Otherwise, hide old frame, show new.
   Frame1(Tabstrip1.SelectedItem.Index).Visible = True
   Frame1(mintCurFrame).Visible = False
   ' Set mintCurFrame to new value.
   mintCurFrame = Tabstrip1.SelectedItem.Index
End Sub
```

Using access keys in tab text also requires programming. Set the KeyDown property of the form to the following event procedure:

```
Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)

If (Shift and acAltMask) > 0 Then ' ALT key was pressed

Select Case KeyCode

Case vbKeyA ' ALT+A was pressed

oleTab.Tabs(1).Selected = True

Case vbKeyB ' ALT+B was pressed

oleTab.Tabs(2).Selected = True

Case vbKeyC ' ALT+C was pressed

oleTab.Tabs(3).Selected = True

End Select

End If
```

End Sub

Setting Field Value

- Initial value can use the table definition: the field's property Default Value can be used, but
 it is limited to a value or an expression using the built-in functions (not custom application
 functions).
- Initial value can be done on the form: The value can be set with an event procedure either immediately after the form is opened (using the form's On Open event) or when a record is added/updated (using the field's Before Update event). In both cases, the field is included on the form and it may be protected.

When form A opens form B which contains fields that must be initialized, the event

- procedure used to open the form can be a macro OpenFormB. That procedure has two actions: (1) OpenForm and (2) RunMacro InitializeFormB. The second step has one SetValue action per field. For example, a date field: [Forms]![formB]![Date Assigned] = Date(). A field on form B that is set to a value from form A: [Forms]![formB]![Exception ID] = [Forms]![formA]![Exception ID].

 I USED MACROS WHEN I STARTED, BUT NEVER AGAIN. In this example, use VBA code to open the form (DoCmd.OpenForm) and initialize the fields (Forms("formB")![Date Assigned] = Date(); Forms("formB")![Exception ID] = Me![Exception ID]). Note that you can initialize the field in the same Sub only if the form is not opened as a dialog box. In that case use an OnOpen event in formB to initialize the field (Me![Date Assigned] = Date(). You could alternatively define formB so the field in question has property Default Value =Date(). When you want to initialize a field to a value in another form, be sure to test to see if the form is open first.
- When form A opens form B in order to edit one record from a list on A, use the where condition to specify which data is to be presented on form B:

```
DoComd.OpenForm "formB" acFormEdit, , "ID = " & Me![PortalID]
```

- The value of a form field can be set to the value of an expression or function, either built-in or custom.
- Programmatically set a Date-Time:
 - a) The field is protected on the form: with properties Enabled = Yes, Locked = Yes, TabStop = No.
 - b) Each form that changes the object with the date-time (1) displays the field in a text box named "Revision" and (2) has a BeforeUpdate event that runs procedure "Set Revision DateTime."
 - c) The procedure has one step: Me![Revision]. Value = Now(). If date only is desired, use function Date().
 - d) In the form event AfterUpdate include statement Me.Refresh.
- A text field can be set to the sum of a table/query field. This may be done when the table/query results are displayed in a subform.
 - 1) Its property ControlSource:

```
=DSum("fieldname", "tablename")
```

2) The DSum function works to initialize the field, however if the table on which it is based changes, the text field will not reflect the current value until it is requeried:

```
Dim ctlSum As Control
Set ctlSum = Forms!MassAssignments!txtCountAssgn
```

ctlSum.Requery

- 3) If in the subform, records can be added, deleted, and changed, then the requery can be done by subform events AfterUpdate, AfterInsert, and AfterDelConfirm. It can also be done as the main form's control event OnExit. It is not clear if all event procedures are needed.
- A text field can be set to the count of rows selected in a list box:

```
Private Sub ListBoxUnassigned_Click()
Dim tempCount As Long, lb As Control, lbItem As Variant
tempCount = 0
Set lb = Me!ListBoxUnassigned
If lb.ItemsSelected.Count > 0 Then
    For Each lbItem In lb.ItemsSelected
        tempCount = tempCount + lb.Column(4, lbItem)
    Next lbItem
    Me!txtCount = tempCount
End If
End Sub
```

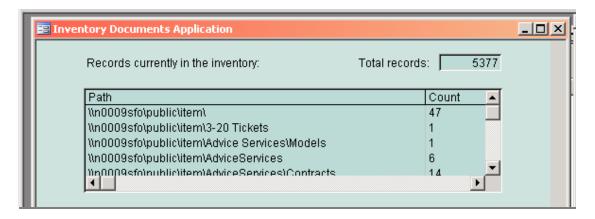
A form used to add a record can, in code, set field value:

```
[Parameter Count] = CountParms(.., ..)
DoCmd.Save
```

Put Record Count on Form

When you want to present a record count on a form, use a Text Box control with the source =DCount("*", "table or query name"). The control value will be refreshed by form module code Me.Refresh (along with any list boxes that may be present).

I had a problem in the form below. The list box displays a query of the main table, Total records is meant to be the records in the table, but control source =Count("*") only worked when the form was first opened. It was not refreshed when records were deleted or added to the table. I finally tried a control source = DSum("Count", "CountFilesByPath") where the domain is the query displayed in the list box.



There are other approaches, but they did not count correctly: (1) using TableDef RecordCount property, (2) using value of count query. The second approach caused an error.

Counting Records in a Subform

When you want to count records displayed on a subform, the basic count is obtained with RecordsetClone.RecordCount, but you have to be careful how you qualify it, i.e., which object

you reference and when. Remember that "Me" refers to the current object; for the form event Me is the form, for a subform event, Me is the subform.

Note: when setting the value of a calculated control with Control Source, using =Screen.ActiveForm.RecordsetClone.RecordCount worked, using Me did not.

(1) With a calculated control on the main form:

=Me!subformname.Form.RecordsetClone.RecordCount

(2) With a calculated control on the subform:

=Me.RecordsetClone.RecordCount

(3) In an event of the main form:

Me! subformname. Form. RecordsetClone. RecordCount

(4) In an event of the subform:

Me.RecordsetClone.RecordCount

(5) In the subform and when it is linked to the main form and contains the subset of all the records that relate to the current record in the main form, use

Me.RecordsetClone.RecordCount

Edit Data

- You may want to edit data entered by a user. This can be done with form field edits.
- Relevant properties:

Format

ValidationRule (caution: active even during Cancel)

ValidationText

- Access will not edit a field unless the value changes (by the user) or ? (not by setting the Dirty property).
- OnExit event procedure performs the edit. When there is a relational edit, the OnExit event procedure for each field should perform the edit.

Find Individual Record on Form

- Use command button created with the wizard and specify Category = Record Navigation, Action = Find Record.
- The command button has Caption property = &Find Record. The & character enables the shortcut key [Alt+F] for the control.
- The command button opens the standard Microsoft Access "Find in field" window, which operates on the field in which the cursor is located. Some user training may be necessary.

Block Record Additions/Deletions on Form

Include form properties:

NavigationButtons = yes RecordSelectors = yes AllowAdditions = no AllowDeletions = no

Print a Form

You can print a form with menu <u>File, Print</u>. The Print dialog box has a Print Range control that controls which data are printed; the choices are All, a page range, and Selected Record(s). The

best use for such a printout is as a record of the form; normally the colors and type are printed as they appear on your monitor. A report is the best way to print the data.

What data is printed?

- If the form has a subform with a default view of Datasheet, many records appear on the form. Regardless of your choice of Print Range, all the records are printed.
- If the form presents one record at a time, you can print only the current record by selecting Print Range = Selected Record(s).
- If the form presents one record at a time with child record subform(s), you can print only the current record by selecting Print Range = Selected Record(s). All the child records will print.

Enhance a form's printability:

- Use menu <u>File, Print Preview</u> to see how printed form will look. A form that is wider than the printed page will wrap horizontally to a second page.
- If the form is wider than a printed page, you can either reduce the page's side margins (with Page Setup) and/or set the page orientation to lanscape (with printer Properties).
- A page header with the form's title and the current date-time make printing a form self-documenting. The current date-time is accomplished with a text box whose data ControlSource = =Now(); the text box can have properties: Border Style = Transparent, Back Style = Transparent, Special Effect = Flat. The page footer can be shrunk to nothing.
- Add a [Print] button to the form that removes the background colors and/or picture before printing.
- Copy the form as a report, then edit the report to make it more "printable." To copy the form: In the Database window, select the form, select <u>Save As</u> from the context menu, in the Save As box select type Report, then [OK]. Put a [Print] button on the form with an OnClick event handler that opens the report.

Initiate New Query/Report

Macro step: Action = RunCommand, Command = NewObjectQuery or NewObjectReport.

Update a Table From a Form Based on a Query

This is commonly done and works well in most situation.

- If there are M:M relationships, set form's RecordsetType property = Dynaset (Inconsistent Updates).
- If the query has a JOIN, an Add may try to add records to both tables—and fail.

Edit Table with Two Forms

You may want to use two forms A and B. On formA use a list box to show all the table records; use a query to include foreign key text values. User selects an Add button to add a record, a Delete button to delete a record. To edit a record, user double-clicks its row. The Add button opens formB so that only one record is added, after which it requeries formA's list box and closes itself. The edit action opens formA so that only the selected record is presented; after the record is updated, the form requeries formA's list box and closes itself. The delete action confirms the delete, then builds a delete query and runs it.

Revision: 9/2/2012

 FormA had one list box and command buttons. (Actually these samples came from different forms.)

```
Private Sub cmdAdd Click()
DoCmd.OpenForm "Subject", acNormal, , , acFormAdd, , "Add"
End Sub
Private Sub cmdDelete Click()
msqTitle = "Edit Subjects"
Dim ctl As Control
Set ctl = Me!SubjectList
If Not ctl.ItemsSelected.Count = 1 Then     ' if a row is not selected, error
   MsgBox "No subject selected.", , msgTitle
   Exit Sub
End If
SubjectID
Dim selRow As Long
                              ' relative position of selected row (row
selRow = ctl.ListIndex
' If DocCount is not zero, cannot delete
If cnt > 0 Then
   MsgBox "You cannot delete subjects assigned to documents.", , msgTitle
   Exit Sub
End If
Dim response As Integer
response = MsgBox("Do you really want to delete the selected subject?",
vbYesNo, msgTitle)
If response = vbNo Then Exit Sub
strSQL = "DELETE ID FROM Subject WHERE ID = " & sid
DoCmd.SetWarnings False
DoCmd.RunSQL strSQL
DoCmd.SetWarnings True
ctl.Requery
End Sub
Private Sub PortalList DblClick(Cancel As Integer)
Dim ctl As Control
Set ctl = Me!PortalList
If Not ctl.ItemsSelected.Count = 1 Then Exit Sub
Dim lbItem As Variant
Set ctl = Nothing
Dim strWHERE As String
strWHERE = "ID = '" & sid & "'"
DoCmd.OpenForm "Portal", acNormal, , strWHERE, acFormEdit, , "Subject"
Forms!Portal!PortalName.SetFocus
Forms!Portal!ID.Enabled = Yes
Forms!Portal!ID.Locked = Yes
End Sub
```

 FormB has data fields and command buttons. It checks for an OpenArgs in order to decide to close after a save.

```
Private Sub cmdSave Click()
```

```
DoCmd.RunCommand acCmdSaveRecord

Dim how As String

If IsNull(Me.OpenArgs) Then Exit Sub

how = Me.OpenArgs

If Len(how) > 0 Then

If how = "Add" Then

Dim ctl As Control

Set ctl = Forms!Subjects!SubjectList

ctl.Requery ' refresh list box to include new subject

DoCmd.Close ' close this form

End If

End If

End Sub
```

Subforms

- You can embed one form within another. Generally you use a subform to present data in a different table than is presented in the main form. The subform data may be for a child table, a parent table, or a completely unrelated table. For convenience the data source of the subform is referred to here as the secondary table. More than one subform can be embedded within the main form.
- A subform is embedded within a main form with a subform control. The subform can be a form or datasheet. If the subform is to be a form, create it first. Then in the main form use the Subform tool to create an area to hold the subform and identify the form to be contained. If the subform is to be a datasheet, start by adding a subform control to the main form, then set its SourceObject property to a table name, query name, or form name; the linked field does not appear and the subform has property CanGrow = Yes.
- Link the subform to the main form with subform properties LinkMasterFields and LinkChildFields. These specify the field(s) that must be the same on the main form and on the subform. See below for details.
- If the subform is for data entry, set the subform's DataEntry property to Yes. This will open the subform at a new blank record and prevent users from scrolling back thru the existing records.
- When the secondary table has a field that holds a code with a separate lookup table (a foreign key) and you want to show the text value but not the code, create a query to hold the parent table primary key, child table key, and lookup table text. Then create the subform based on the query. Be aware that you will not be able to use the standard navigation controls, so you must include your own custom controls. Form properties should include:

DefaultView = Datasheet AlowFormView = No RecordSelectors = No NavigationButtons = No

- To tab out of a subform: {Ctrl+Tab] will tab you out manually, or you could set an event in the LostFocus event of the last field of the subform to manually move the focus to a control on the main form.
- When the data is displayed as a datasheet the standard navigation controls and record selector icon are presented.

- When the secondary table is linked to the main table with a foreign key (not a primary key) do not use the standard navigation controls in the subform. Include the primary key as a text control in the main form (which can be populated manually or by special code you write). You can add a secondary table record at the same time as a main table record. But if you try to type or paste text into a control on the subform before the main table record is saved, you will get a warning message "You can't assign a value to this object." But after you click [OK] the value appears in the control.
- When you use custom navigation controls: To add a record, you must use a separate form; a popup modal form works well; the key field(s) that are linked to the main form can be initialized with form property DefaultValue:

```
DefaultValue = Forms! [main form name]! [linked field name]
```

The Add form uses code to act on [Add] and [Cancel] buttons. In the following example, code checks to be sure a value was selected from the combo box.

 Subforms with master-child links are buggy in Access 2003. Subforms on tab controls just add to the complexity.

Editing Data in a Child SubForm with Standard Navigation Controls

- Fo a new record in the main form, the only standard subform navigation control that is enabled is Last. The record selector is a tiny arrow. Normally you enter the data in the main form first, when you move the focus to a control in the subform (probably by positioning the mouse in the control) the main form record is saved automatically and the enabled navigation keys for the child subform become First and Last. You then enter text in the subform.
- After a new main record is saved, when the first child record is saved, the enabled navigation keys for the child subform become First, Next, Last, and New.
- Should you enter data in the child subform without entering the main table data and then try to start a new child record, you'll get an error message "Index or primary key cannot contain a Null value." This happens because Access is trying to save the first child record but, because it does not have the primary key for the parent record, it has a Null value in part of the child record's key. Your only action at this point is to undo the edits to first child record and enter text in the main form.
- When adding or changing a child record, your edits are saved automatically when you click a control outside of the subform.

You can delete a record from a datasheet by selecting the row, then pressing the [Delete] key.

Link Subform to Main Form

You can use the LinkChildFields property with the LinkMasterFields property to specify how Access links records in a form or report to records in a subform, subreport, or embedded object such as a chart. When these properties are set, Access automatically updates the related record in the subform when you position on a different record in the main form (this does not work when you pre-select a record in the main form). Both properties apply to the subform control in the main form. Corresponding fields in each form must have compatible data types and field sizes.

LinkChildFields is the name of one or more linking fields in the subform, subreport, or embedded object. It is analogous to a foreign key. If you want to use a calculated value as the link for a subform, subreport, or embedded object, define a calculated field in the child object's underlying query and set the LinkChildFields property to that field. When there are two or more fields, separate each with a semi-colon, e.g., "ID;Title".

LinkMasterFields is the name of one or more linking fields (bound or unbound) in the parent form or report. It is analogous to a primary key. When there are two or more fields, separate each with a semi-colon, e.g., "ID;Title".

It is common to link a subform to a single record presented in the main form. It is also possible to link a subform to the selected row in a list box.

How To Tell Which Form Subform Is In

If the subform has the focus, the property ActiveForm refers to the main form.

Control Data on Main Form with Popup Form

- This approach is generalizable: the popup form can be used by more than one main form.
- The popup form selects data that is then presented on the main form.
- Popup form has two hidden text fields: main form name and control name (to hold the selected data) on main form. The main form sets the values of these fields before opening the popup form.
- Popup form (1) prompts user for selection criteria, (2) selects data and stores it in a temporary table, (3) sets RowSource property of the control whose name is in the hidden text field = temporary table, (4) closes itself, and (5) sets focus on the main form (by using the hidden text field value).

```
Dim strSQL As String
strSQL = "SELECT . . . INTO temptable FROM . . . WHERE . . ."
. . .
dbs.Execute strSQL2
Dim ctl As Control
Set ctl = Forms(txtFormName).Controls(txtControlName)
ctl.RowSourceType = "Table/Query"
ctl.RowSource = "temptable"
ctl.Requery
dbs.Close
Dim strFormName As String
```

```
strFormName = Me!txtFormName.Value
DoCmd.Close
Forms(strFormName).SetFocus
```

Resequencing Records in a Single Record Form

The default record sequence is that of the record source. It is possible to resequence the records in an open form. You can do this by selecting a field and using menu <u>Records, Sort, Sort</u> <u>Ascending/Descending</u>. This can only be done for one field at a time.

Note that memo fields are sorted only on the first 255 characters.

If you want to sort on multiple fields in Form view, or sort on non-adjacent columns in Datasheet View, or specify a column order other than the default left-to-right order in Datasheet view, use the Advanced Filter/Sort dialog box accessed with menu Records, Filter, Advanced Filter/Sort. While the dialog box is open, use menu Filter, Apply Filter/Sort to apply the sort, then close the filter/sort box. The special sort can be removed with menu Records, Remove Filter/Sort.

In VBA there seems to be at least two ways of doing this:

a. You can change the underlying record source. Be sure the new query contains all the fields in the form. Any changes to the record source are not saved when you close the form, so that when you re-open the form it uses its original record source.

```
Private Sub cmdResequence_Click()
Me.RecordSource = "JobsByEmployer"
Me.Requery
End Sub
```

b. You can use the OrderBy property of the Form object. The down side to this approach is that the form's OrderBy property is changed permanently, not just while the form is open. Consequently, when you re-open the form, the record sequence will be the same as when it was

```
OrderBy = "ContactName DESC"
OrderBy = "Job, Contact Name"
The property must be applied separately:
```

last open—after the OrderBy property was changed.

frmMaine.OrderByOn = True

It may be necessary to have an index on the table to facilitate the different sequence.

Resequencing Records in a List Box or Combo Box

The default record sequence is that of the list box's row source. It is possible to resequence the records using the underlying row source. It would be great if the user could just click a column heading to change the order, but column headings cannot have events. There are several other approaches:

- (1) Replace the list box with a sub-form datasheet. The datasheet allows the user to resize the columns and easily the sort or filter the data. The disadvantage is that the only way to highlight an entire row is to click on the Record Selector (not obvious).
- (2) Use the VB list box control. This is probably available as an ActiveX control: Microsoft ListView Control version 6.0.

Revision: 9/2/2012 Page 78 of 119

- (3) Use a separate control: Set the listbox headers to No, create wide thin command buttons across the top of the listbox columns to look like headers, create OnClick code for each button to change the listbox recordsource to correspond with the desired sequence.
- (4) Use a separate control: Keep the list box column headings and use a group of radio buttons, one button for each sequence. The default button is the one that corresponds to the default list box sequence. When the user clicks a button, the group's OnClick event changes the record sequence.



Use the Option Group Wizard to assign a value to be returned when a radio button is clicked:



```
Sub optGrpSequence_OnClick()
Dim rst as Recordset
Set rst = Me.Recordset
Dim curRec as Variant
Set curRec = rst.Bookmark
Select Case optGrpSequence.Value
Case 1 ' ID
    Me.RecordSource = "RecruitersByID"
Case 2 ' Name-Agency
    Me.RecordSource = "RecruitersByNameAgency"
Case 3 ' Agency-Name
    Me.RecordSource = "RecruitersByAgencyName"
Me.Requery
Me.Requery
Me.Bookmark = curRec
End Sub
```

With Separate Control

You could have one query to reflect each possible record sequence; be sure the new queries contains all the fields in the list box and in the same order. The query for the default sequence is used for the list box's row source. Use a separate control to allow the user to select a different sequence. That control's OnClick event can change the list box's row source to the corresponding query and then requery the list box.

```
Sub cmdResequence_OnClick()
Me.lbGadgets.Rowsource = "GadgetsByDate"
Me.lbGadgets.Requery
End Sub
```

Selection Criteria Form, version 1

- One or more of several fields can be used to provide selection criteria.
- Each criterion has a checkbox and a list box of possible values. The sample code is based on a list box that is not multi-select.
- Form has no data source. It allows record editing but no additions or deletions.
- Checkbox has properties: DefaultValue = False. Checkbox Click event procedure enables
 the corresponding list box when the checkbox is enabled and vice versa
 (listboxcontrol.Enabled = checkboxcontrol).

```
lbDateAssigned.Enabled = CheckDateAssigned
Set ct1 = Me!lbDateAssigned
Call Deselect(ct1)
. . .
Private Sub Deselect(lbctl As ListBox)
Dim n As Variant
With lbctl
If .ItemsSelected.Count = 1 Then
    n = .ItemsSelected(0)
    .Selected(n) = False
End If
End With
End Sub
```

- List box has properties: Enabled = No; Locked = No; BoundColumn = 0 (unbound); RowSourceType = Table/Query; RowSource = SQL select statement (like "SELECT DISTINCT fieldname FROM queryname").
- Select command button: Click event builds SELECT query and (1) saves it as QueryDef or (2) saves results in new temporary table (done as "SELECT INTO").

The remaining work is to make the query results visible. My first attempt at this is based on (1) embedding the selection criteria form as a subform, (2) saving the selected records as a temporary table, (3) displaying the results in a list box (originally with no data source), and

(4) using a command button to attach the results to the list box (by setting its RowSource property = table name and using the Requery method). NOTE any temporary table must be deleted before closing the form; if displayed in a list box, it must be first removed from the list box (by clearing the RowSource property).

Selection Criteria Form, version 2

- This form is used only to accept and edit selection criteria and to execute the query. The query stores the results in a temporary table TempSelection. The results are displayed in a list box in a separate form, what I call the action form.
- The select form is designed to be used with several different actions forms.
- Approach:
- 1. Action form opens. Its list box has no data source.
- Select form opens.
 as modal and popup, with no control box or close box
 opened by action form's OnOpen event, which hides the action form name and list box
 control name in invisible text boxes on the select form
- 3. Select form edits selection criteria. When no errors, it builds the SQL statement as a "SELECT INTO" and runs it. Then it changes the list box's data source to the temporary table, and closes itself.
- 4. The action form reappears from behind the closed select form with the selected data in place.
- 5. The user interacts with the action form. If the user cancels the action, the Cancel button's OnClick event deletes the temporary table. If the user completes the action, the completion process ends with deleting the temporary table and closing the form.
- Code in action form to open the select form:

```
Private Sub Form_Open(Cancel As Integer)
DoCmd.OpenForm "SelectionCriteriaPopup"
Forms!SelectionCriteriaPopup.txtFormName = "RemoveAssignments"
Forms!SelectionCriteriaPopup.txtControlName = "ListTempSelection"
End Sub
```

Code in select form after select query run:

```
Dim ctl As Control
Dim s As String
Set ctl = Forms(txtFormName).Controls(txtControlName)
ctl.RowSourceType = "Table/Query"
ctl.RowSource = "TempSelection"
ctl.Requery
dbs.Close
DoCmd.Close
End Sub
Private Sub cmdCancel Click()
Dim strFormName As String
strFormName = Me!txtFormName.Value
DoCmd.Close acForm, strFormName, acSaveNo
DoCmd.Close
End Sub
    Code in action form after action completed:
response = MsgBox("Assignments removed.", vbOKOnly, "BRT Exceptions")
```

Call cmdCancel Click

Code in action form when cancelled:

```
Private Sub cmdCancel Click()
Dim ctl As ListBox
Set ctl = ListTempSelection
With ctl
   .RowSourceType = ""
    .RowSource = ""
    .ColumnHeads = False
    .BoundColumn = 0
    .Requery
End With
Dim strSQL As String
strSQL = "DROP TABLE TempSelection"
Dim dbs As Database
Set dbs = CurrentDb
dbs.Execute strSQL
DoCmd.Close
Exit Sub
```

Using OpenArgs

- Forms can be passed a string argument when they are opened by code. See the FileOpen method on page **Error! Bookmark not defined.** for its details.
- Essentially, formA opens formB:

```
DoCmd.OpenForm "Subject", acNormal, , , acFormAdd, , "Add"
```

• FormB tests for the argument and acts on it. In this case formB is used to add a single record, then close itself. Normally, it would stay open after the record save.

Positioning an Open Form on a Different Record

When a form opens it is normally positioned on the first record in the underlying record source. What if you want to (1) open the form so that it is positioned on a particular record or (2) reposition an open form on a particular record? Either can be accomplished by (1) positioning a clone of the underlying recordset on the particular record with Find or Seek and (2) setting the form's Bookmark to the recordset's Bookmark.

The key player in this process is the recordset which corresponds to the form's record source. The recordset, either DAO (the native Access library) or ADO, has methods Find and Seek which can be used to locate the particular record. It also has a Bookmark property which uniquely identifies the current record.

Choosing Find or Seek:

- You can only use Seek with recordsets of type table: when the record source is a table or a single-table query and the search field is indexed.
- You can use Find for recordsets of type dynaset: when the record source is a multi-table query.
- Seek can be easier to code, but is limited to indexed table recordsets.

About form recordsets:

- The *form*.Recordset property returns a recordset object representing the record source for the specified form, report, list box control, or combo box control. Read/write. Caution: changing which record is current in the recordset returned by the form's Recordset property also resets the current record of the form.
- The *form*.RecordsetClone property returns a recordset that is a <u>copy</u> of the form's underlying table/query. Read-only. When a new recordset object is opened, its first record is the current record. If you apply a filter to the form, the recordset object reflects the filter. The recordset object is available only as long as the form is open. You use the RecordsetClone property to navigate or operate on a form's records independent of the form itself.
- You can synchronize the form's current record with the recordset's current record: By setting the form's Bookmark to the cloned recordset's bookmark, the current record in the recordset is made current in the form, i.e., displayed in the form.

```
Dim rst As Recordset
Set rst = Me.RecordsetClone
rst.Index = "ID"
rst.Seek "=", curID
If rst.NoMatch = False Then
   Me.Bookmark = rst.Bookmark
End If
rst.Close
Dim rst As Recordset
Set rst = Me.RecordsetClone
rst.FindFirst "ID" = curID
                                 ' best performance if field is indexed
If rst.NoMatch = False Then
   Me.Bookmark = rst.Bookmark
End If
rst.Close
```

In the two preceding examples, you can use operators other than "=": >, <, <>, LIKE, etc. You can also do substring comparisons.

So, to reposition an open form:

- 1. Establish a cloned recordset.
- 2. Find or Seek the desired record in the recordset.

- 3. Set the form's bookmark to the recordset's bookmark.
- 4. Close the recordset.

To open a form so that it is positioned on a particular record:

- 1. Pass the key of the particular record to the form in OpenArgs (of DoCmd.OpenForm).
- 2. The form's OnOpen event checks for the presence of OpenArgs.
- 3. If it finds it, it uses the Recordset property to open a recordset, then Find or Seek to position on the desired record, then closes the recordset.
- 4. If it does not find it, it does nothing.

There is another technique—using DoCmd.OpenForm with the Where parameter—which can work even if the form is already open, but is known to have odd behaviors.

Note that using the Filter property is not suitable as it will not provide access to the entire table/query once the form is open.

Events

Event category	Events	Occur when
Window events	Close, Load, Open, Resize, Unload	A user or code opens, resizes, or closes a form or report.
Focus events	Activate, Deactivate, Enter, Exit, GotFocus, LostFocus	Objects receive or lose the focus, or become active or inactive.
Data events	AfterDelConfirm, AfterInsert, AfterUpdate, BeforeDelConfirm, BeforeInsert, BeforeUpdate, Change, Current, Delete, NotInList, Updated, Dirty	A user or code enters, deletes, or changes data in a form or control, or moves the focus from one record to another.
Mouse events	Click, DblClick, MouseDown, MouseMove, MouseUp	A user performs a mouse action, such as clicking or double-clicking.
Keyboard events	KeyDown, KeyPress, KeyUp	A user types on the keyboard, or keys are sent using the SendKeys action or the SendKeys statement.
Print events	Format, NoData, Page, Print, Retreat	A report is being printed, or is being formatted for printing.
Filter events	ApplyFilter, Filter	A user creates, applies, or removes a filter for a form.
Error and Timing events	Error, Timer	Access or the Jet database engine encounters an error, or a specified time interval passes.

You can cancel the following events:

ApplyFilterDeleteMouseDown (only the rightBeforeDelConfirmDirtymouse button, and only with

BeforeInsert	Exit	a macro)
BeforeUpdate	Filter	NoData
DblClick	Format	Open
	KeyPress (only with a macro)	Print
		Unload

You cancel an event by specifying a macro containing the CancelEvent action as the corresponding event property setting or, with the exception of the MouseDown and KeyPress events, by setting an event procedure's Cancel argument to True.

Form Events

The following definitions are from the help file.

Error event occurs when a run-time error is produced in Microsoft Access when a form or report has the focus. This includes Microsoft Jet database engine errors, but not run-time errors in Visual Basic or errors from ADO.

Open: The Open event occurs when a form is opened, but before the first record is displayed. For reports, the event occurs before a report is previewed or printed.

Load: The Load event occurs when a form is opened and its records are displayed. (In the case of a form with slow-to-complete queries as the source of data controls, this event occurs before the data appears in the controls.)

Activate: The Activate event occurs when the form or report receives the focus and becomes the active window. (I could not engineer a situation when this event occurred)

Current: The Current event occurs when the focus moves to a record, making it the current record, or when the form is refreshed or requeried. (In my test, this happened before form appeared.)

Dirty occurs when the contents of a form or the text portion of a combo box changes. It also occurs when you move from one page to another page in a tab control. Canceling the Dirty event will cause the changes to the current record to be rolled back. It is equivalent to pressing the ESC key. If you need to test this, use your own variable. The Dirty event does not fire for new record. **Close** occurs when form is closed and removed from the screen.

Unload occurs when the form is closed but before it is removed from the screen.

BeforeUpdate event occurs after you have entered the new or changed data in the current record and before the record is saved.

Delete: Occurs when the user performs some action, such as pressing the DEL key, to delete a record, but before the record is actually deleted.

Timer: The Timer event occurs at regular time intervals as specified by the form's TimerInterval property (stated in milliseconds).

LostFocus event occurs when form loses the focus, applicable when focus switches from form to subform and vice versa. Applies to form only when it has no visible or enabled controls. **Deactivate** event occurs when you switch between two open forms. It cannot be cancelled.

Followed by Activate event in second form. Also fires between Unload and Close.

Opening a form

Open . . . Load . . . Resize . . . Activate . . . Current The form does not become visible until after activate.

"If you're trying to decide whether to use the Open or Load event for your macro or event procedure, one significant difference is that the Open event can be canceled, but the Load event

can't. For example, if you're dynamically building a record source for a form in an event procedure for the form's Open event, you can cancel opening the form if there are no records to display."

The OnLoad event procedure can carry out actions for the controls or records on a form after the form opens. For example, you can specify default settings for controls.

When you first open a form with data

```
Open . . . Load . . . Resize . . . Activate . . . Current . . . Enter (control) . . . GotFocus (control)
```

Closing a form

Unload . . . Deactivate . . . Close

Closing a form with X button

Exit (control) . . . LostFocus (control) . . . Unload (form) . . . Deactivate (form) . . . Close (form) If you've changed data in a control, the BeforeUpdate and AfterUpdate events for both the control and the form occur before the Exit event for the control.

Creating a new record

```
Current (form) . . . Enter (control) . . . GotFocus (control) . . . BeforeInsert (form) . . . BeforeUpdate . . . AfterInsert (form) . . . AfterUpdate . . .
```

Deleting a record

Delete . . . BeforeDelConfirm . . . AfterDelConfirm

Move to another record

BeforeUpdate . . . AfterUpdate . . . Exit (control2) . . . LostFocus (control2) . . . RecordExit (form) . . . Current (form)

Enter text in text box in new record.

```
KeyDown = . . . KeyPress . . . BeforeInsert . . . Dirty . . . KeyUp
```

Some events can be cancelled. One way is with the DoCmd.CancelEvent method. (The following events can cancel themselves: BeforeInsert, BeforeUpdate, DblClick, Dirty, Exit, Open, Unload.) Another way to cancel an event procedure is as follows:

If you cancel an update, the value of the OldValue property replaces the existing value in the control.

- Use the form's Before Update event to perform -level validation, including checking for nulls. Use a control's After Update event to programmatically fill in a separate control.
- What I am looking for is which event it is that happens after the data contents of controls appear. I am creating a form with three list boxes, each with a source of a record count of a query. The queries take time to run, so the form opens and the boxes are blank for awhile until numbers appear. I want to know which event occurs when the numbers appear.

I could not find an event that fit my needs. But the following code in the OnOpen event worked perfectly. The hourglass appeared until the form opened with all the data visible.

```
DoCmd.SetWarnings True
Dim x
If (Me.DiscrepancyA.Value = 0 And Me.DiscrepancyC.Value = 0 And
Me.DiscrepancyE.Value = 0) Then
    x = "no"
Else
    x = "yes"
End If
DoCmd.SetWarnings False
```

Form Control Events

AfterUpdate	after the changed data has been saved
BeforeUpdate	before the changed data has been saved; useful for validating
	data
Dirty	only for combo box and tab control, when the content changes
Change	after contents of a text box or the text portion of a combo box
	changes; it also occurs when you move from one page to
	another page in a tab control
Click	
DoubleClick	can be cancelled
Enter	before control receives the focus from another control
Exit	before control loses the focus to another control
GotFocus	
KeyDown	
KeyPress	
KeyUp	
MouseDown	
MouseMove	
MouseUp	

Form Opens Blank

This happens when the form has a RecordSource and the data controls have no values. For instance if the form has text boxes with source of DCount() and the counts are all zero, the form opens blank—with nothing in it, only the border. One solution is to remove the RecordSource. But . . . doing so means Me.Refresh does not work. However, if all the text controls are calculated (i.e., with sources equal to functions like DCount), then Me.Recalc works. It may be necessary to add a command button to run the Recalc.

Slow Opening Form

When a form has controls whose source is a query, the queries are run one after another¹, and the form is not opened until all the queries are done (I think). When a form takes a long time to open because source queries take a long time to run, the form has no values in its text box controls until the queries complete. This can be misleading. There are three ways to handle this:

1) Speed up the queries as much as possible.

-

¹ This is because Access is single threaded.

- 2) Move one or more queries to an OnEnter event of the combo box so they are not run when the form opens.
- 3) Display an hourglass until the form appears with data on it.

There are four techniques for option 1:

- Make sure the sort and parameter fields in the queries are indexed fields in the underlying tables.
- Limit the number of fields returned.
- Set the "Auto Expand" property to No.
- Use saved queries rather than SQL statements for the control's record source.

There is one technique for option 2:

Leave the Row Source for the combo box blank in its property sheet. Then create an "OnEnter" event procedure for the combo box. In that procedure set the combo box's RowSource to the desired query or table by specifying:

```
Me!ComboBoxName.RowSource = "queryortablename"
```

This will make the form load faster by not running the combo box query until the user requests the data.

There is one technique for option 3:

 Find the query that takes the longest to open. Add an OnLoad event to the form like the following code. In the example InvalidTitles is the text box sourced with the slowest query.

```
Private Sub Form_Load()
' this code is to hold the hourglass while the text boxes are filling with slow-running
' queries.
DoCmd.Hourglass True
Dim x
If (Me!InvalidTitles.Value = 0) Then
        x = "no"
Else
        x = "yes"
End If
DoCmd.Hourglass False
End Sub
```

Form Suspends Itself

A form can be opened with window mode = Hidden. The Stop statement suspends execution, but unlike End, it doesn't close any files or clear variables; it can be placed anywhere in procedures. The UserForm has a Hide method which hides it without unloading it. A form has a Visible property (boolean) which can be queried or set to affect the form. form.SetFocus moves the focus to a specified form. All in all it looks possible, needs testing to confirm.

Controlling One Form From Another

Open form before initializing contents:

```
DoCmd.OpenForm "formb"
Forms!formB!controlname = value expression
```

The second line happens after last form open event (onCurrent). So any programmatic use of that data must be done by an event procedure for a control.

Report Stuff

Field Control Source

The control source for a report text box control can be (1) a field in the report record source or (2) an expression (in which case the control is called a calculated control). Examples of expressions:

=[English Name] & ": " & [Comment]

=[English Name] & Chr(10) & Chr(13) & [Comment]

In these examples [English Name] and [Comment] may refer to either control names or data fields. Be careful to avoid confusion by naming the control differently than the field.

When using an expression do not refer to the field itself. Typically text controls have the same name as the field data source. If you change the data source to an expression which refers to the actual field, you will need to change the name of the control to avoid an endless (recursive) loop.

Grouping Records

You can group records by one or more fields. Grouping allows you to incorporate special report features: group header and/or footer that are presented only once for each group. A group header is defined in its own report section as is a report footer. Groups are defined by:

- Group On: the value or range of values that starts the the group.
- Group Interval: interval or number of characters to group on; the interval depends on the field's datatype and the Group On property, default is 1.
- Keep Together: controls when a group is kept together on the same page; choices are No, Whole Group, and First Detail (prints the group header on a page only if it can also print the first detail record).
- Additional space before and after the section.
- Colored background in the section.
- Decorative controls in the section like a horizontal line or an image.

A group can be used to present data like:

group1text	group1atext	
		detailrecordtext
		detailrecordtext
		detailrecordtext
group1text	group1btext	
		detailrecordtext
		detailrecordtext
		detailrecordtext

In this example the group fields are presented in the group header section and the detail record fields are presented in the detail section.

A similar style can be achieved without grouping:

group1text	group1Atext	detailrecordtext
		detailrecordtext
	group1Btext	detailrecordtext
group2text	group2Atext	detailrecordtext

This style is achieved with the property HideDuplicates = Yes, all fields are in the detail section with no defined group.

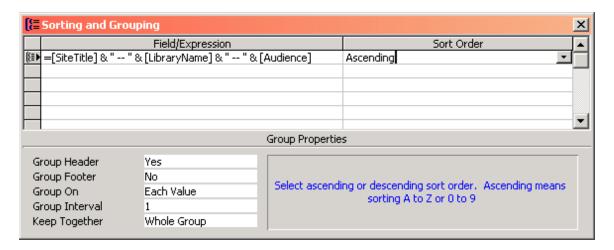
You can combine grouping these two approaches: put all fields in the detail section and define a group header for its Keep Together property and its spacing. Note a header always resets HideDuplicates—so if there are two fields in the group, as in the second example, you will get:

group1text	group1Atext	detailrecordtext
		detailrecordtext
group1text	group1Btext	detailrecordtext
group2text	group2Atext	detailrecordtext

If you want to achieve the second example with spacing between groups, create a group on a single compound key (combining the two group fields into one).

Compound Group Heading

If you have data with a compound key (multiple fields) which you want to report with one group header that is populated with the compound key, then in the "Sorting and Grouping" window you can enter an expression equal to the concatenated fields:



Page Break on Group

Use empty (i.e., no controls) group footer section with property Force New Page = After Section.

Repeat Group Headers on Subsequent Pages

When the contents of a group span more than one page, you may want to repeat the group header on each page on which the group details are presented. In group header section set property RepeatSection = Yes.

I encountered the situation where a report worked just fine on one computer, but not on a second: when it began to open the status message said "Formatting page, press Ctrl+Break to stop" until I stopped it. The report had no problem until the group header RepeatSection was set to Yes, then it would not open, . Looking for an answer with Google found possible factors: keeping with next, memo fields, section Can Grow/Can Shrink. I found the group header section had Keep Together = True. Turning it off did not solve the problem. I found the group header

section had CanGrow = True, turned it off. The Detail section had KeepTogether = True, turned it off. That worked!

Summed Data in Group Header

The example here is a report based on a query with a field [Count of Titles]. I want to sum this field for each report group and include the sum in the group header formatted as:

headertext (xx)

The solution involves the Sum function.

=Sum([fieldname]) has a scope that corresponds to the report section in which it is located. When in the Report Header/Footer, the scope is the entire report. When in a Group Header/Footer, the scope is for that group. This hold true for other arithmetic functions like Count and Average.

Memo Fields

- You may find that memo fields are truncated in a report text control. Be sure:
 - 1. field is not truncated in the record source (table or query)
 - 2. field is not used for grouping
 - 3. field has no format in the table definition
 - 4. field has no format property in the report definition
 - 5. field in report definition has properties Can Grow = Yes, Can Shrink = Yes.

Put Record Count on Report

• In report header/footer only, use a text box with a data source = "=Count(*)".

Adjusting Size of Controls to Data

- It is possible to (1) allow a control to expand to as many lines as are needed to present its data and (2) allow a control to shrink when there is no data. Use the CanGrow and CanShrink properties respectively. In order for these to work, there must be no other control on the line, nor may the subject control overlap any adjacent control. If you are going crazy trying to get a control to shrink when the data is absent, check the Line Height of the control and its neighbors; change as necessary. If you use either CanGrow or CanShrink, you will probably want to limit the height of the control to one line of text.
- You can display a variable with its label such that nothing displays if the variable is empty:

```
Format: !"(Cell 1) "@
CanGrow: Yes
CanShrink: Yes
```

Multi-Column Report

Is defined with menu <u>File</u>, <u>Page Setup</u> and the columns tab. You can specify the number of columns, margins, and data flow.

Presenting Data From Multiple Sources

A report can have but one data source. That controls the source of the data presented in the various controls. However, data from additional sources can be presented in subreports. There is a subreport control that is used to include data from other sources.

- You can embed a query's datasheet view in a report: create a subreport control and set its data source = query name. The column widths in the report are the same as in the query. If you want to change the report's appearance, you must change the query's datasheet view.
- A subreport can be defined as you are inserting the subreport control or you can define the subreport as its own report, then copy it into the master report. If the latter, you edit the subreport in its own report object.

Subreports

- You can embed one report within another. The two reports may be related or not. A main report can contain 1, 2, or more related subreports.
- Linking a subreport to a main report involves several things:
 - (a) Relationships must be defined for the underlying tables.
 - (b) The main report is based on a table with a primary key.
 - (c) Subreport property LinkMasterFields (exists only for subreport control, not the original report object) = list of fields in main report's table (master table) that correspond with fields in subreport's table (child table).
 - (d) Subreport property LinkChildFields = list of fields in subreport's table that correspond to LinkMasterFields.
- Subreports appear on printed reports.
- Subforms do not appear when report is printed.

Child SubReport

- When you have a one:many parent-child table relationship, you can easily present both tables in one report. The report is linked to the parent table (as record source). The children data is presented in a subreport. Use the control toolbox to create a subreport on the main form. Use the wizard tool because it makes the process easy. The wizard prompts you for the child table, the fields, the linking field(s), and the name of the subreport. It creates a separate subreport and embeds it in the main report. The subreport has the CanGrow attribute = yes.
- The default view of the child subreport is datasheet.
- To edit the format of the subreport, open it in design mode.

Filter Report Data

You can restrict the data in a report with two report properties:

Filter = wherecondition

FilterOn = Yes

Example:

Filter: DataValue(DateTime) = Date() selects records with current date

Using Date Selection Criteria for a Report

In this example, a report has a table as a data source. The table's records contain a date-time stamp. A form allows the user to specify for which dates the records are to be reported. There are three choices: today, a date range, and all dates. The form edits the choice, returns an error message when appropriate, and opens the report when there is no error.

Revision: 9/2/2012 Page 93 of 119

- When the user chooses today or a date range, a WHERE phrase is applied to the report to qualify the records. All code is contained in the form's module.
- Form controls include an option group with three option buttons, a Start Date text box, an End Date text box, a Select command button, and a Cancel command button. The form's AllowEdits property = Yes.
- The text boxes are protected except when the date range option button is selected. The values in the text boxes are cleared when a different option button is selected. The form's OnOpen event procedure protects the text boxes. The option group's OnClick event procedure clears the text boxes' values and protects or unprotects the Start Date box depending on which option button is selected. The End Date box is only unprotected when the Start Date value passes its edits; this is done by the Start Date box's OnExit event procedure.

```
Private Sub Form_Open (Cancel As Integer)
[End Date].Locked = True
[Start Date].Locked = True
[End Date]. Enabled = False
[Start Date].Enabled = False
Private Sub Date Option Group Click()
If [Date Option Group]. Value = [Date Range]. Option Value Then
    [Start Date].Locked = False
    [Start Date].Enabled = True
Else
    [Start Date]. Value = ""
    [End Date].Value = ""
    [Start Date].Locked = True
    [End Date].Locked = True
    [Start Date].Enabled = False
    [End Date].Enabled = False
End If
End Sub
```

Most of the code involves editing the date range. The Start Date must be >= the date the application is first used and < today. The End Date must be >= the Start Date and <= today. The Start Date's OnExit event procedure tests the value for a valid date (the text box allows any keyboard character) and one that is in the valid range; if there is an error, it clears the Start Date and disables the Select button, otherwise it unprotects the End Date and initializes it with the Start Date. The End Date's OnExit event procedure tests the value for a valid date and one that is in the valid range; if there is an error, it clears the End Date and disables the Select button, otherwise it enables the Select button.

```
Private Sub Start_Date_Exit (Cancel As Integer)
Dim DateError As Boolean
If Not IsDate([Start Date].Value) Then
        MsgBox "Start date must be a valid date", vbOKOnly, AppTitle
        DateError = True
        GoTo SetControls
End If
If CDate([Start Date].Value) > #6/25/2003# and CDate([Start Date].Value) < Date
Then
        DateError = False
Else</pre>
```

```
DateError = True
    MsgBox "Start date must be <= 6/25/2003 and < today", vbOKOnly, AppTitle
End If
SetControls:
If DateError= True Then
    [Start Date].Value = ""
    [End Date].Locked = True
    [End Date]. Enabled = False
    cmdSelect.Enabled = False
    [Start Date].SetFocus
Else
    [End Date].Value = [Start Date].Value
    [End Date].Locked = False
    [End Date]. Enabled = True
    cmdSelect.Enabled = True
    [End Date].SetFocus
End If
End Sub
Private Sub End-Date Exit (Cancel As Integer)
Dim DateError As Boolean
If Not IsDate([End Date].Value) Then
    MsgBox "End date must be a valid date", vbOKOnly, AppTitle
    DateError = True
    GoTo SetControls
End If
If CDate([End Date].Value) >= CDate([Start Date].Value) And CDate([End
Date }. Value <= Date Then
   DateError = False
Else
    DateError = True
   MsgBox "End date must be >= start date and <= today", vbOKOnly, AppTitle
SetControls:
If DateError= True Then
    [End Date]. Value = ""
    cmdSelect.Enabled = False
    [End Date].SetFocus
Else
   cmdSelect.Enabled = True
End If
End Sub
```

The Select command button's OnClick event procedure builds a WHERE clause based on which option is selected. It closes the form, then opens the report with the WHERE clause when today or a date range were selected, otherwise it opens the report without a WHERE clause. The form is closed before the report is opened so that the report's window is on top.

```
where = StatDt & " = Date()"
Case 2  ' date range
    where = StatDt & " >= #" & CDate([Start Date].Value) & "# And " & StatDt
& " <= #" & CDate([End Date].Value) & "#"
    Case 3  ' all dates
        where = "n"
End Select
DoCmd.Close acForm, "Report Statistics"
If where = "n" Then
        DoCmd.OpenReport "Statistics", acViewPreview
Else
        DoCmd.OpenReport "Statistics", acViewPreview, where
End If
End Sub</pre>
```

• The form's Cancel button's Onclick event procedure closes the form. The Cancel button has property Canel = Yes; this causes the [Esc] key to close the form.

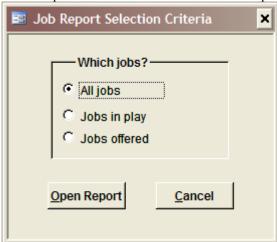
```
Private Sub cmdCancel_Click()
DoCmd.Close
End Sub
```

Use Form to Select Data For Report

- Use a form to display a list box of summarized actual key data. The user selects one or more rows from which query SQL is constructed in code and passed to the report as a WHERE clause. The list box query uses the DISTINCT predicate.
- If report is opened with DoCmd.OpenReport then add a where phrase. In the first example the ID field is a long integer. If it were a different data type, it would require different handling. In the second example the Location field is a string.

```
DoCmd.OpenReport "JobDetail", acPreview, , "ID = " & Me.ID DoCmd.OpenReport "JobDetail", acPreview, , "Location = '" & Me.Location & "'"
```

• Use a form to let the user choose one of several selection criteria, then build a WHERE clause to correspond to the selected criterion and open the report with the WHERE clause.



Private Sub cmdOpenReport_Click()
On Error GoTo Err_cmdOpenReport_Click
Dim stDocName As String
stDocName = "Jobs"
Select Case optWhichJob.Value

```
Case 1
   strWhere = ""
   strWhere = "Status <> 'Dead'"
    strWhere = "Status = 'offered' or Status = 'accepted'"
Case Else
   strWhere = ""
End Select
DoCmd.OpenReport stDocName, acViewPreview, , strWhere
DoCmd.Close acForm, "ReportSelectionCriteria"
Exit cmdOpenReport Click:
Exit Sub
rivate Sub cmdOpenReport Click()
On Error GoTo Err cmdOpenReport Click
Dim stDocName As String
stDocName = "Jobs"
Select Case optWhichJob.Value
Case 1
    strWhere = ""
Case 2
   strWhere = "Status <> 'Dead'"
Case 3
    strWhere = "Status = 'offered' or Status = 'accepted'"
Case Else
   strWhere = ""
End Select
DoCmd.OpenReport stDocName, acViewPreview, , strWhere
DoCmd.Close acForm, "ReportSelectionCriteria"
Exit cmdOpenReport Click:
Exit Sub
Err cmdOpenReport Click:
If Err.Number <> 2501 Then ' this traps the report being cancelled for no
records
   MsgBox Err.Description
End If
Resume Exit_cmdOpenReport_Click
End Sub
```

You may want to block the report from opening if it has no data. Do this with (1) a NoData event in the report to present a message "no data" and then cancel the report with "Cancel = True". (2) In the form code trap the report cancel error:

```
If Err.Number <> 2501 Then
     MsgBox Err.Description
End If
Resume Exit_cmdButton_Click
```

Using Date Selection Criteria for a Report with a Parameterized Query

 When you want to report data only for a certain month and year, you can use a parameterized query.

- Approach:
- 1. Create query for base report data that selects all data regardless of date value.
- Create a second query that prompts user to enter first year, then month to be used as selection criteria. This query includes created year and month fields based on the key date.

```
SELECT [Request ID], [Complete Date]
FROM sjd3
WHERE (((Year([Complete Date]))=[Enter 4-digit year for report data]) AND
((Month([Complete Date]))=[Enter month number for report data]))
ORDER BY 1;
```

- Create report with data source = second query. There are two ways to put the year and month in the report heading: (1) include the values in the query and then source them in the heading or (2) use date functions sourced on the [Complete Date] field.
- The disadvantages of parameterized queries is that there is no editing of the user's input values(s). So if the report is empty, there is no explanation. This disadvantage can be minimized by tightly focusing the selection criteria.
- When a complete date is used for a query parameter, it must be entered as "1/23/2004" or "1-23-2004" or "01-23-2004" or "01/23/2004" to work properly.

Report Events

When you first open a form, the following events occur in this order: Open \rightarrow Load \rightarrow Resize \rightarrow Activate \rightarrow Current

When you close a form, the following events occur in this order: Unload \rightarrow Deactivate \rightarrow Close

There are other events: Error, NoData, Page.

Error: Occurs when a run-time error is produced in Microsoft Access when a form or report has the focus.

Page: Occurs after Microsoft Access formats a page of a report for printing, but before the page is printed. You can use this event to draw a border around the page, or add other graphic elements to the page.

NoData: Occurs after Microsoft Access formats a report for printing that has no data (the report is bound to an empty recordset), but before the report is printed. You can use this event to cancel printing of a blank report. This form open event can be cancelled: Setting the Cancel argument to True (-1) prevents the report from opening. You can also use the CancelEvent method of the DoCmd object to cancel printing the report.

Macros

Generally macros are to be avoided in deference to VBA procedures. However there is one exception: the Autoexec macro. This macro will run automatically when the MDB file is opened. The macro can run a VBA procedure.

Queries

You can always edit the data in a query when the query is based on either only one table or on two tables that bear a one-to-one relationship to each other.

You can never edit the data in a query when:

- The query is a crosstab query
- The query is a SQL-specific query—subqueries, pass-through, union, and data-definition queries.
- The field you are trying to edit is a calculated field.
- The query is based on three or more tables and there is a many-to-one-to-many relationship.
- The query contains a GROUP BY clause.
- The query includes more than one table or query, and the tables or queries aren't joined by a join line in Design view.

Parameter Queries

- When a query starts, it can display a text box for you to enter an ad hoc value.
- Can be used as basis for forms and reports.
- To create, in Design View, enter prompt text in Criteria cell within square brackets, e.g., "[Which month?]" or "Between [Type the beginning date:] And [Type the ending date:]".
- Examples:

```
SELECT . . . FROM . . . WHERE Year = [Which year?]
SELECT ID, Name, [Which year?] AS Year FROM . . .
```

 Be sure to trap [Cancel] in code or it will trigger the run-time error 2001. You'll need On Error GoTo FormError which will close the form if Err.Number = 2001.

Revision: 9/2/2012

SQL

Access 2000 SQL conforms closely, but not completely, to the most recent ANSI standard for SQL, SQL-92.

The original SQL standard was completed as a USA ANSI (American National Standards Institute) standard in 1986, and adopted as an ISO (International Standards Organization) standard in 1987. Expansions and revisions were released in 1989 and 1992. Because of ANSI and ISO copyright restrictions, copies of the SQL standards specifications are available only for purchase.

The ANSI SQL-92 standard specification can be found at http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt

I generally refer to IBM's DB2 SQL Reference. It is available as PDF files on IBM's website.

- IBM DB2 Universal Database SQL Reference Volume 1 Version 8.2 SC09-4844-01 is available at ftp://ftp.software.ibm.com/ps/products/db2/info/vr82/pdf/en_US/db2s1e81.pdf
- Version 9, DB2 for Linux, Unix, and Windows, is SC10-4250-00 and is available at ftp://ftp.software.ibm.com/ps/products/db2/info/vr9/pdf/letter/en_US/db2s2e90.pdf

SFLFCT

General format of SELECT query is:

```
SELECT ... FROM ... WHERE ... GROUP BY ... HAVING ... ORDER BY ...
```

The clauses of the select statement are processed in the following sequence:

- 1. FROM phrase
- 2. WHERE phrase
- 3. GROUP BY phrase
- 4. HAVING phrase
- 5. SELECT phrase

Thus you cannot filter data using a field created with AS in the SELECT phrase:

```
SELECT x, y, z AS mm FROM g WHERE mm = "w" is invalid. This can be effected with a subselect: SELECT x, y, mm FROM (SELECT x, y, z AS mm FROM g) WHERE mm = 'w"
```

NOTE: In Access 2003 a subselect is valid in a FROM phrase.

The SELECT phrase is composed of a list of one or more field names separated by commas:

```
SELECT Hostname
SELECT Hostname, Facility
```

In the SELECT phrase you can rename a field in the result set by the use of the AS clause:

```
SELECT Hostname, Facility AS Fc
SELECT Facility, Count(*) AS CountFacility
```

This is strictly not an alias name; in DB2 an alias is an independent object defined with DDL.

The following example illustrates the use of the Format function in a SELECT phrase and the AS label with an embedded space.

```
SELECT Format(CandidateFiles.MigrateDate,"Short Date") AS [Migrate Date],
LibraryName, Count(*) AS [Count]
FROM CandidateFiles
WHERE (((IsNull([MigrateDate]))=False))
GROUP BY Format(CandidateFiles.MigrateDate,"Short Date"),
CandidateFiles.LibraryName
ORDER BY 1, 2;
```

When a SELECT sources a table with a field with type Hyperlink, the query datasheet also has the Hyperlink. Unless . . . the query includes a UNION. If this is undesirable, you can get around it by using two queries:

```
SELECT ID, URL FROM MigrateWorksheet WHERE ID IN (SELECT ID FROM ErrorQueryWithUnion)
```

ORDER BY field | field position ASC | DESC [, field | field position ASC | DESC]

Examples:

```
ORDER BY 1, 2
ORDER BY field2, field5 DESC, field6
```

SELECT INTO

This query is a make-table query, i.e., it inserts records into a previously non-existent table. When you create the table, the fields in the new table inherit the data type and field size of each field in the query's underlying tables, but no other field or table properties are transferred (such as primary key). Syntax:

```
SELECT field1[, field2[, ...]] INTO newtable [IN external database] FROM source
```

The following example copies the named fields in every record: SELECT Column1, Column2, Column3, INTO Table2
FROM Table1

The following example copies all the fields in every record: SELECT *
INTO Customers_copy
FROM Customers

I cannot find a way to create a field that did not exist in the source table.

JOIN

The JOIN phrase is used to combine tables. It works best when used with tables in a parent-child relationship. When used with other relationships, the result set may be a fiction. It can contain all the possible combinations of the two (or more) tables—which is probably not what you want.

- When using Design View to create queries with JOINs with foreign keys, show the tables in reverse order of their significance. For instance if the the primary table is A and the lookup table is B, order them as B then A (B to the left of A). This gets the JOINs written correctly. In any case, use SQL View to verify which join(s) were created; change by overtyping as necessary.
- JOINs are sometimes used to qualify the selection. When the table relationship is not parent-child, this may be better accomplished with a WHERE phrase and a subquery.
- There are 3 different JOINS.
 - INNER JOIN returns data when the join condition is true (2 tables match on 1+ common fields).
 - LEFT JOIN returns data for all records in the left table even if there are no matching records in the right table.
 - RIGHT JOIN returns data for all records in the right table even if there are no matching records in the left table.
- When using a join for a foreign key-lookup table, use the join (left/right) that handles the case where there is no matching record.
- Nested joins are done differently than in DB2. In first example, join two tables where they will always match on a common field (as in parent-child or lookup table when foreign key is required). In second example, join one table to two tables. When second table is joined on foreign key and key is required, use inner join. When second table is joined on foreign key and key is optional, use left join. ?? Third example is of main table with two foreign keys, one required, the other optional; FROM clause right joins tableC to inner join of tableA and В.

```
SELECT * FROM tableA INNER JOIN tableB ON tableA.fieldA = tableB.fieldA
SELECT * FROM (tableA INNER JOIN (tableB INNER JOIN tableC ON tableB.fieldB =
tableC.fieldB) ON tableA.fieldA = tableB.fieldA)
SELECT * FROM (tableC RIGHT JOIN (tableA INNER JOIN tableB ON tableA.fieldA =
tableB.fieldA) ON tableC.fieldB = tableA.fieldB)
```

Access will do a sequence of two left joins, but not an inner join followed by a left join.

Example of JOIN on more than one field:

```
SELECT fields
FROM table1 INNER JOIN table2
ON table1.field1 = table2.field1 AND
ON table1.field2 = table2.field2) OR
ON table1.field3 = table2.field3)];
```

UNION

A UNION query combines two or more result tables or queries. Syntax:

```
| TABLE tablename | UNION | TABLE tablename | UNION . . . | ORDER BY
| queryname | UNION ALL | queryname | UNION ALL . . .
| SELECT . . .
                          | SELECT . . .
```

The UNION phrase causes duplicates to be excluded. The UNION ALL phrase causes duplicates to be retained.

Revision: 9/2/2012 Page 102 of 119

All queries must return the same number of fields, but they may have different data types and sizes.

Aliases may be used only in the first SELECT statement; they are ignored in all others.

The ORDER BY phrase can use column names or integers. If the former, they must reflect the field names used in the first SELECT statement.

SQL like "queryname1 UNION queryname2" fails the syntax check. "SELECT . . . FROM queryname1 UNION SELECT FROM queryname2" works.

UPDATE

General syntax of UPDATE query is:

Be aware that the second syntax is not compatible with other DBMS like SQL Server. Nor is their syntax compatible with Access.

[There is no FROM phrase as in other versions of SQL.)

Example:

```
UPDATE Orders
SET OrderAmount = OrderAmount * 1.1, Freight = Freight * 1.03
WHERE ShipCountry = 'UK';
```

The following example does a character replacement in a table, saves the field before it is changed, and saves a code to reflect the nature of the change. The error code is meant to hold several values, each indicating a separate update.

```
UPDATE Test
SET OriginalFilename = Filename, Error = Error & "&", Filename =
REPLACE(Filename, "&", "")
WHERE InStr(Filename, "&") > -0;
```

Apparently you cannot use a UNION with an UPDATE.

You can use an UPDATE with a JOIN but only when you are accessing tables (and not queries). If the SQL is in an (Update) query then it must have property Unique Records = Yes., otherwise the query won't work—it is "not updatable".

(This works in Access and MySQL 5 but not in SQL Server)

```
UPDATE TableOne
INNER JOIN TableTwo ON TableOne.commonID = TableTwo.commonID
SET TableOne.field1 = TableTwo.fieldX

UPDATE TableOne, TableTwo
SET TableOne.field1 = TableTwo.fieldX
WHERE TableOne.commonID = TableTwo.commonID
```

```
UPDATE CandidateFiles AS CF
INNER JOIN MigrateWorksheet AS MW
On CF.ID = MW.ID
SET CF.LibraryName = MW.LibraryName, CF.Library = IIF(MW.LibraryName = "Public", "U", "R")
WHERE CF.ID IN (SELECT ID FROM MigrateDiscrepancyE);
```

When you JOIN two tables, if the source field of a SET statement is an autonumber field, then the corresponding field in the destination table must be a long integer (and not an autonumber).

The following query updates records in one table that match records in a second table. In addition it appends a record to the first table when no record matches the second table row.

```
UPDATE Table1
RIGHT JOIN Table2
ON
   Table1.Ix = Table2.Ix
SET
   Table1.Ix = Table2.Ix,
   Table1.Descr = Table2.Descr
```

I learned the hard way that I cannot have an UPDATE query with two fields and where field 2 is calculated from the value of field 1. Here is what I meant to say:

An expression may contain references to columns of the target table of the UPDATE statement. For each row that is updated, the value of such a column in an expression is the value of the column in the row **before the row is updated**.

Here's an example of nested joins in an update query. This query was written to handle the situation where data from one table (N) was being merged into a second table (S). The merge was effected by copying data (only when it was of higher quality) from the first table into matching records in the second table. Almost 1000 records needed to be updated. The merge was complicated by the quality of data in each table which resulted in over 10 use cases for handling the merge, many of which reflected specific data values. The approach was to create a temporary table to control the update, this temporary table was created with a SELECT INTO query with GROUP BY:

```
SELECT INTO ScannedDiffModel
N.Model AS NM, S.Model AS SM, Count(*) AS RecordCount, "" AS UseCase
FROM Scanned AS N INNER JOIN Server AS S ON N.Hostname=S.Hostname
WHERE N.Model<>S.Model And N.Model<>""
GROUP BY N.Model, S.Model;
```

Then the use case identifier was entered manually in the new table. This table then was used to control the UPDATE. For use case C:

```
UPDATE
(ScannedDiffModel AS D INNER JOIN
        (Server AS S INNER JOIN Scanned AS N
        ON S.Hostname = N.Hostname)
ON S.Model = D.SM
```

```
AND N.Model = D.NM)
SET S.Model = N.Model
WHERE D.UseCase = "C";
```

INSERT

Used to append records to an existing table.

General format of INSERT query is:

The use of * to refer to fields works only when fields have the same name in source and destination

There are two simple forms:

```
INSERT INTO . . . VALUES . . . INSERT INTO . . . SELECT . . . FROM . . .
```

The second form can also contain WHERE, GROUP BY, and HAVING clauses, as well as table joins, aliases, and subselects.

INSERT INTO table1
SELECT *
FROM table2
WHERE fieldx NOT IN (SELECT fieldx from table1)

This form works fine except when the inserted records could duplicate existing records on the primary key. This can happen when there is an autonumber field in both tables.

DELETE

The DELETE query deletes one or more records from a table. The general syntax of the DELETE query is:

```
DELETE FROM table WHERE criteria

Examples:
DELETE * FROM Employees WHERE Title = "Trainee"
```

TRANSFORM

DELETE * FROM Employees

The TRANSFORM statement creates a crosstab query, aka pivot table (in Excel); the statement is non-standard and unique to Access. The result set in datasheet view looks like a two-

dimensional table with a x-axis, y-axis, and some data in the intersecting cells. The result set is easily published by pasting it into a Word document.

```
TRANSFORM aggfunction
selectstatement
PIVOT pivotfield [IN (value1[, value2[, ...]])]
```

The TRANSFORM statement has these parts:

Part	Description
aggfunction	An SQL aggregate function that operates on the selected data. The results populate the intersecting cells.
selectstatement	A SELECT statement. This selects the y-axis values (row headings). This statement can include a WHERE clause and/or a GROUP BY clause.
pivotfield	The field or expression you want to use to create column headings in the query's result set; the x-axis.
value1, value2	Fixed values used to create column headings. This will restrict pivotfield to create headings only for the specified values. Columns are created for all values even if there is no data.

SQL aggregate functions: Avg, Count, First, Last, Min, Max, Sum, StDev (standard deviation), and Var (variance).

Example:

TRANSFORM First(Definition)
SELECT Entity, Attribute FROM AttributesInDuplicateEntities
GROUP BY Entity, Attribute
PIVOT ModelName

	CrosstabAttributeDefinitionsByEntityModel				
Entity	Attribute	ITEM CSD STAGING	ITEM MONEY MARKET FUND STAGING	ITEM MUTUAL FUND STAGING	ITEM SERVICE SDA STAGING
CURRENT STATUS SERVICE REASON	issue status service change time				The time expressed in hours, minutes and seconds when a change in issue status and service was recorded.
CURRENT STATUS SERVICE REASON	operational issue service code				A symbol identifies one and only one operational service.
DEBT	AD_VALOREM_TAX_CD				Ad Valorem Tax Status: B = supported b limited/unlimited ad valorem tax pledge L = Supported by limited ad valorem tax pledge U = supported by unlimited ad valorem tax pledge Not applicable will be blank in this field.
DEBT	bank qualified status code				A code which identifies the Bank Qualified status of an

	CrosstabAttributeDefinitionsByEntityModel				
Entity	Attribute	ITEM CSD STAGING	ITEM MONEY MARKET FUND STAGING	ITEM MUTUAL FUND STAGING	ITEM SERVICE SDA STAGING
					Issue.
DEBT	bearer suffix identifier				
DEBT	bond form code				A code which identifies the forms (registered, bearer, both) in which a bond may exist.
DEBT	call type code				A code which identifies the times in the year when the call may be exercised.
DEBT	conversion status code				
DEBT	convertible indicator	An indicator which identifies whether a Debt Issue is or is not convertible. Y = Debt Issue is convertible N = Debt Issue is not convertible on the convertible N = Debt Issue is not convertible			An indicator which identifies whether a Debt Issue is or is not convertible. Y = Debt Issue is convertible N = Debt Issue is not convertible
DEBT	convertible to cash amount				The dollar amount into which a Debt Issue may be converted.

Correlation Names

Access can use correlation names for tables/queries in FROM phrase. One source says a table correlation name is aka alias. The syntax can be:

FROM tablea AS a

FROM tablea a

Example:

```
SELECT S.ID, S.URL, S.Title AS [Susan Title], S.Subject AS [Susan Subject], S.[Sub-Subject] AS [Susan Sub-Subject], S.DocumentType AS [Susan DocumentType],
L.Title AS [Larry Title], L.Subject AS [Larry Subject], L.[Sub-Subject] AS [Larry Sub-Subject], L.DocumentType AS [Larry DocumentType]
FROM FilesToMigrateSusan AS S INNER JOIN FilesToMigrateLarry AS L
ON S.ID = L.ID
ORDER BY 1;
```

Example:

```
UPDATE CandidateFiles AS CF
INNER JOIN Library ON UCase(CF.Site) = Library.SiteID AND UCase(CF.Library) =
Library.LibraryID
SET CF.LibraryName = Library.LibraryName
WHERE (IsNull(CF.LibraryName) OR IsEmpty(CF.LibraryName)) AND UCase(CF.Migrate)
= "Y";
```

Using Comparisons in SQL

Beware comparing dates. If the field is a Short Date, i.e., created by Format(field, "Short Date"), then the field is actually text and comparisons will yield erroneous results. If you want to compare two dates, be sure they are either date-time data types or DateValue(date-time field). The DateValue function returns the Short Date as a date, not text.

Predicates

SELECT * FROM Products

There are several types of predicates.

comparison, use comparison operators like > =

quantified: ANY, SOME, ALL, DISTINCT, DISTINCTROW, TOP n [PERCENT]

Use the ANY or SOME predicate, which are synonymous, to retrieve records in the main query that satisfy the comparison with any records retrieved in the subquery. The following example returns all products whose unit price is greater than that of any product sold at a discount of 25 percent or more:

```
WHERE UnitPrice > ANY
SELECT UnitPrice FROM OrderDetails
WHERE Discount >= .25)
SELECT DISTINCT fielda, fieldb FROM tablex
```

between: BETWEEN, NOT BETWEEN

Examples:

EMPLOYEE.SALARY BETWEEN 20000 AND 40000 SALARY NOT BETWEEN 20000 + HV1 AND 40000

exists: EXISTS

The EXISTS predicate tests for the existence of certain rows.

The fullselect may specify any number of columns, and

- The result is true only if the number of rows specified by the fullselect is
- not zero
- The result is false only if the number of rows specified is zero
- The result cannot be unknown.

Example:

```
EXISTS (SELECT * FROM TEMPL WHERE SALARY < 10000)
```

The EXISTS predicate specifies a correlated subquery (defined in section Subselect-Subquery) and then compares a value against the existence of one or more rows in that subquery, i.e., whether the subquery returns any records (1+ records is True, 0 records is False). "WHERE EXISTS subquery" is satisfied if subquery returns True; "WHERE NOT EXISTS subquery" is satisfied if subquery returns False.

Functionally it is similar to IN and NOT IN, but may be more efficient. In general, if the selective predicate is in the subquery, then use IN. If the selective predicate is in the parent query, then use EXISTS [not sure what this means].

SELECT fieldlist FROM datasource WHERE [NOT] EXISTS (subquery)

Which customers have placed an order?

SELECT CustomerID, CompanyName FROM Customers AS A WHERE EXISTS

```
(SELECT * FROM Orders WHERE A.CustomerID = Orders.CustomerID) 1
```

Which customers have not placed an order?

```
SELECT CustomerID, CompanyName

FROM Customers AS A

WHERE NOT EXISTS

(SELECT * FROM Orders WHERE A.CustomerID = Orders.CustomerID)
```

in: IN, NOT IN

subject of predicate can be an expression, fullselect, or set of expressions, for example: expression IN expression

is equivalent to a basic predicate of the form: expression = expression expression IN (fullselect)

is equivalent to a quantified predicate of the form: expression = ANY (fullselect) expression NOT IN (fullselect)

is equivalent to a quantified predicate of the form: expression <> ALL (fullselect) expression IN (expressiona, expressionb, ..., expressionk)

is equivalent to: expression = ANY (fullselect)

where fullselect in the values-clause form is: VALUES (expressiona), (expressionb), ..., (expressionk)

(expressiona, expressionb,..., expressionk) IN (fullselect)

is equivalent to a quantified predicate of the form: (expressiona, expressionb,..., expressionk) = ANY (fullselect)

Example:

```
SELECT ID, Filename, Migrate
FROM CandidateFileExtensions
WHERE Extension NOT IN (SELECT Extension FROM ValidFileTypes)
AND UCase(Migrate) = "Y";
```

like: LIKE, NOT LIKE

The LIKE predicate searches for strings that have a certain pattern. The pattern contains text and special characters:

- The underscore character (_) represents any single character.
- The percent sign (%) represents a string of zero or more characters.
- Any other character represents itself.

Example:

```
SELECT PROJNAME FROM PROJECT WHERE PROJECT.PROJNAME LIKE '%SYSTEMS%'
```

null: NULL, NOT NULL

This predicate tests for null values.

Examples:

PHONENO IS NULL SALARY IS NOT NULL

 $^{^{1}}$ It is the WHERE x = y phrase that establishes the subquery as correlated.

SELECT Predicates

• The SELECT statement can include predicates:

```
SELECT (ALL | DISTINCT | DISTINCTROW | TOP n [PERCENT]) FROM table . . .
```

ALL is assumed if none specified.

```
SELECT ALL * FROM table . . . 
SELECT * FROM table . . . ' same as SELECT ALL * FROM . . .
```

DISTINCT omits records containing duplicate data in the selected fields. When more than
one field is specified, it is their combination that is evaluated for unique values.

```
SELECT DISTINCT field FROM table
```

 DISTINCTROW omits data based on entire duplicate records. Useful with inner joins, but only when fields from some (but not all) of the tables are selected.

```
SELECT DISTINCTROW field FROM table
```

• TOP returns a given number of record that fall at the top/bottom of a range specified by an ORDER BY phrase.

```
SELECT TOP 25 fieldA, fieldB FROM table ORDER BY Zip DESC SELECT TOP n PERCENT fieldA, fieldB FROM table ORDER BY ID ASC
```

Create one row with a value of your own choosing.

```
SELECT TOP 1 "all" AS fieldname FROM table
```

• Populate a list box with values from a table and a special value like "all." To get "all" to sort first, put one blank in front (" ALL").

```
SELECT TOP 1 " ALL" AS [Tier] FROM JobTitle UNION
SELECT DISTINCT [Tier] from JobTitle
```

Subquery-Subselect

- Access SQL is a subset of SQL-92. Some capabilities are missing. Especially in the following list, it may be that Access disallows a subselect in a FROM phrase.
- A subquery is a SELECT statement used as an *expression* (nested) within another query. It is always bracketed by parentheses.
- There are two kinds of subqueries. A *plain subquery* is not dependent on the outer query; also known as an uncorrelated subquery. A *correlated subquery* is dependent on the outer query in that it contains a reference to one or more columns in the outer query. Within a subquery a reference to a column of a table identified at a higher level (e.g., an outer query) is called a *correlated reference*.
- Subqueries can be nested, establishing an hierarchy. For example, query 1 uses query 2 which uses query 3. The terminology here is that queries 2 and 3 are inner queries, query 1 is an outer query for the other two, and query 2 is an outer query for query 3.
- Cannot have an ORDER BY phrase.
- Can refer to tables specified in a previous outer query within the hierarchy with a correlated reference;

```
SELECT * FROM tableA A1 WHERE fieldB IN (SELECT fieldB FROM tableB WHERE fieldC = A1.fieldC)
```

• Can be used in the field list (the AS clause works with DB2 but not perhaps Access):

```
SELECT fieldA, fieldB, (SELECT fieldC FROM . . .) AS xyz FROM . . .
```

Can be used in the FROM phrase:

```
SELECT ZZ.fieldA, ZZ.fieldB FROM (SELECT field A, field B, fieldC FROM . . .) AS ZZ . . .
```

Can be used in a JOIN phrase:

```
SELECT fieldA, fieldB FROM tableA A1 JOIN (SELECT fieldB FROM . . .) AS YY ON . . .
```

• Can be used in a WHERE phrase with membership predicates (may not work):

```
SELECT * FROM tableA WHERE fieldB [NOT] IN (SELECT fieldB FROM . . .)

SELECT * FROM tableA WHERE (fieldA, fieldB) IN (SELECT field A, fieldB FROM . .
.)

SELECT * FROM tableA WHERE fieldB = ANY/SOME/ALL (SELECT fieldB FROM . . .)

SELECT * FROM tableA WHERE (fieldB, filedC) = ANY/SOME (SELECT fieldB, fieldC FROM . . .)
```

Note: the "= ANY" predicate is the equivalent of the "IN" predicate when both are of the form "(expressiona, expressionb, . . . , expressionk) IN (subquery)." Similarly, "NOT IN" is the equivalent of " \Leftrightarrow ALL".

Can be used in a WHERE phrase with comparison predicates:

```
SELECT * FROM tableA WHERE fieldB > (SELECT Avg(fieldB) FROM tableB . . .)
Other comparison operators: =, <, <>, >=, <=.
```

Can be used in a WHERE phrase with existence predicate:

```
SELECT * FROM tableA WHERE EXISTS (SELECT fieldB FROM . . .)
```

Can be used in a HAVING phrase:

```
SELECT * FROM tableA GROUP BY fieldA HAVING Max(fieldC) < (SELECT Avg(fieldC
FROM . . .)</pre>
```

When you join one table to another for purposes of qualifying the first table's records, the subselect can be used in both the FROM phrase and the JOIN phrase with WHERE phrase to limit the number of records in the two temporary tables before they are joined. The following sample also uses the GROUP BY phrase instead of a SELECT DISTINCT.

```
SELECT . . . FROM (SELECT . . . FROM tableA WHERE . . . ) AS T1 JOIN (SELECT . . . FROM tableB WHERE . . . GROUP BY . . .) AS T2 ON . . .
```

- The correlated subquery will be run once for each candidate row selected by the outer query.
- It is good practice to qualify all column references used in subqueries.

```
SELECT * FROM tableA WHERE ID IN (SELECT A FROM tableB WHERE tableB.x = tableA.x)
```

Miscellaneous Query Techniques

Enclose field names in square brackets when using functions in SQL.

```
SELECT ID, Len([Notes]) AS NotesLength FROM . . .
```

Use custom VBA functions for complex processing. If you want to set a table field equal to a
calculated value whose calculation involves using complex string manipulation, conditional
processing, and/or APIs, then create a custom function in a VBA module and invoke it in
your SQL statement.

```
SELECT ID, CalcIP([Hostname]) AS CalcIP FROM . . .
```

Where CalcIP() is a VBA function:

```
Function CalcIP(h as String)
. . .
End Function
```

- When you use VBA functions in a query, especially in a WHERE clause, you may get a runtime error ("invalid procedure call"). This happens when one or more records come to grief with the functions. For example, using an InStr() function without excluding the instances when the value is zero (meaning the substring could not be found) will cause an error.
- When you want counts of groups of records where there is more than one field whose values can be combined as a group, do as follows:

```
SELECT A, B, C, D, Count(*) AS RecordCount FROM tablea GROUP BY A, B, C, D;
```

This will	yield a	recordset	like t	he fol	lowing:

Α	В	C C	D	RecordCount
	Intel	Microsoft	Windows	2
	Linux			1
	Linux	IBM	AIX	4
	Linux	Microsoft	Windows	1
	Linux	Red Hat	Enterprise Linux	2
	Unix			5
	Unix	IBM	AIX	318
	Unix	Red Hat	Enterprise Linux	2
	Unix	Sun Microsystems	Solaris	140
Microsoft	Windows			20
Microsoft	Windows	IBM	AIX	3
Microsoft	Windows	Microsoft	Windows NT	102

When you want to count records by groups of data and one of the groups is, not the domain of values of a field but whether a certain value is present or not, use the following form of query:

```
SELECT Facility, Billed, Count(*) As RecordCount
FROM (SELECT Facility, IIf(InStr([InventoryMethod], "billed") = 0, "no", "yes")
AS Billed FROM Server)
GROUP BY Facility, Billed;
```

Facility	Billed	RecordCount
	no	319
	yes	803
BDC	no	2
RCC	no	209
RCC	yes	252

Facility	Billed	RecordCount
SDC	no	48
SDC	yes	40

 Select the record for each record key having the latest date. Only works if the date field is not defined as Short Date

```
SELECT JobID, Max(Date) AS LatestDate
FROM History
GROUP BY JobID
ORDER BY 1;
```

Select records having particular date.

```
SELECT BIRTHDAY_TABLE.PersonName, BIRTHDAY_TABLE.BirthDate
FROM BIRTHDAY_TABLE WHERE (Month(BIRTHDAY_TABLE.BirthDate)) < Month(Date())
OR ((Month(BIRTHDAY_TABLE.BirthDate)) = Month(Date()) AND
Day(BIRTHDAY_TABLE.BirthDate) < Day(Date())) ORDER BY
Month(BIRTHDAY_TABLE.BirthDate), Day(BIRTHDAY_TABLE.BirthDate);"</pre>
```

Select records with time-date stamp = today

WHERE DateValue([tableA].[fieldA]) = Date()

• Find combinations of fields that have the same value. In this example the table has many more fields than these. Note that the WHERE phrase excludes some records from consideration. The HAVING phrase selects only instances where there are duplicates (more than one occurrence in the group).

```
SELECT SiteTitle, LibraryName, Title, Count(Title) AS CntTitle
FROM MW
WHERE DoNotMigrate = False
GROUP BY SiteTitle, LibraryName, Title
HAVING Count(Title) > 1
ORDER BY 1;
```

Select records having a particular date.

```
SELECT ID, StartDate
FROM Claim
WHERE
StartDate=#09/06/2009#
ORDER BY 1;
```

• List the records associated with duplicate key values (i.e., that are duplicates on one or more fields). To extend the previous example, select the individual records that match the combination of fields selected in that query.

```
SELECT *
FROM MW INNER JOIN (query above) AS D
ON MW.SiteTitle = D.SiteTitle AND MW.LibraryName = D.LibraryName AND MW.Title =
D.Title :
```

List the records that are duplicates on one field.

```
SELECT SiteTitle, LibraryName, Title, ID FROM MW
WHERE Title IN
(SELECT Title FROM MW
GROUP BY Title
HAVING Count(Title) > 1)
ORDER BY 3;
```

Create new field with conditional value

```
SELECT RC.[RS CD], RC.EFFDT, RC.[RC SEQ], RC.[ERR IF NO VAL SW], RC.[BF CD], BF.[CHAR TYPE CD], BF.[CHAR SRC FLG], BF.[ERR IF NO VAL SW], IIf([RC].[ERR IF NO VAL SW]="Y" Or [BF].[ERR IF NO VAL SW]="Y","Y","") AS Required
FROM RC INNER JOIN BF ON RC.[BF CD] = BF.[BF CD]
WHERE (((BF.[CHAR TYPE CD])<>"N/A") AND ((BF.[CHAR SRC FLG])<>"CH"))
ORDER BY RC.[RS CD], RC.[BF CD];
```

Get all unique combinations of month and year from a date in a table

```
SELECT DISTINCT Year([Complete Date]) & "-" & Month([Complete Date]) AS YM FROM [MT Log Table]
ORDER BY 1;
```

Count records returned by query

```
SELECT Count(*) AS CountA1
FROM [A1 Detail List];
```

Count records returned by query and new column created to hold query name

```
SELECT
"A1 Detail List" AS [Query Name],
Count(*) AS Count
FROM [A1 Detail List];
```

Count groups of records. For example, for each record in tableA, count the number of records in tableB with a foreign key of the tableA record. The basic approach is to combine the GROUP BY clause and the Count function, each focused on a different field. If you need to add another table, do this as a separate query that is based on the count query. In the second example, tableB is DocumentSubject. NOTE: If you use Count(*) on an inner join, it will return 1 when there are no matching records; the solution is to say

```
Count([tablename].[fieldname])
```

```
SELECT Count(*) AS [Bill Factor Count], [Bill Factors by Rate Category].[Rate
Category]
FROM [Bill Factors by Rate Category]
GROUP BY [Bill Factors by Rate Category].[Rate Category];

SELECT SubjectsWithPath.ID, Count(DocumentSubject.SubjectID) AS DocCount
FROM SubjectsWithPath LEFT JOIN DocumentSubject ON SubjectsWithPath.ID =
DocumentSubject.SubjectID
GROUP BY SubjectsWithPath.ID;
```

Add numbers in records

```
SELECT PortalName, Sum([Count]) AS CountChanges
FROM JoinCountsAfterExportGrouped
GROUP BY PortalName;
```

CASE construct for creating new output field based on multiple conditions

```
Switch([Alg Type].[ALG TYPE CD]="CV",[SQ Rule].[SQ PARAM 01],[Alg Type].[ALG TYPE CD]="BC" Or "CD" Or "MD",[SQ Rule].[SQ PARAM 03],[Alg Type].[ALG TYPE CD]="MA",[SQ Rule].[SQ PARAM 08]) AS Parameter,
```

simpler version:

```
Switch([Alg Type Parm].SEQNO=1,[SQ Rule].[SQ PARAM 01],[Alg Type
Parm].SEQNO=3,[SQ Rule].[SQ PARAM 03],[Alg Type Parm].SEQNO=8,[SQ Rule].[SQ
PARAM 08]) AS Parameter,
```

Create a table while putting data into it

```
SELECT * INTO newtable FROM (subquery)
SELECT field[, field2. . . ] INTO newtable FROM table
SELECT field[, field2. . . ] INTO newtable FROM queryname
```

Create named table by the insertion of the results of a named query

```
SELECT * INTO Counts FROM [A1 Detail Count]
```

Create named table by inserting some rows from a named query

```
SELECT * INTO TempSelection
FROM AssignedExceptions
WHERE [Exception ID] = 6409;
```

Create named table by inserting <u>aggregate</u> rows from a named query

```
SELECT [Batch Job Name] AS Job, [Run Date-Time] AS [Date], [Message Category] AS [Msg Cat], [Message Number] AS [Msg Nbr], [Assigned To], Count([Exception ID]) AS Count INTO Test FROM AssignedExceptions GROUP BY [Run Date-Time], [Batch Job Name], [Message Category], [Message Number], [Assigned To] ORDER BY [Run Date-Time], [Batch Job Name], [Message Category], [Message Number], [Assigned To];
```

Empty named table

DELETE * FROM tablename

Delete named table

DROP TABLE tablename

 Append query result to a named table; clearly, the field names in the Counts table are the same as in the query

Add records to named table and set value of one field based on values of other fields.

```
INSERT INTO StatusByPhase ( Department, JobTitle, Phase )
SELECT [Department], [JobTitle],
Switch(IsDate([DateKNCompleted]),5,(IsDate([DateJobProfileDrafted]) And
IsDate([DateInventoryDrafted])),4,IsDate([DateLastTranscriptSent]),3,IsDate([DateJobOutlineDrafted]),2,IsNull([DateJobOutlineDrafted]),1) AS Phase
FROM JobTitle;
```

Convert a table—using only three columns—into a two-dimensional matrix. The x-axis is CHAR TYPE CD, the y-axis is RS CD. CHAR SCR FLG is in the intersections. Additional columns are ignored.

```
TRANSFORM First([Matrix List].[CHAR SRC FLG]) AS [The Value]

SELECT [Matrix List].[RS CD]

FROM [Matrix List]

GROUP BY [Matrix List].[RS CD]

PIVOT [Matrix List].[CHAR TYPE CD];

Set date field

UPDATE Message
```

```
WHERE . . . ;
```

• An update query can update more than one field at a time:

```
SET fieldA = "y", fieldB = "x", . . .
```

- Update query with inner join must have property Unique Records = Yes. If not, the query won't work—it is "not updatable".
- Delete record from table

```
DELETE [table.*] FROM table WHERE criteria

DELETE * FROM Employees WHERE Title = 'Trainee'

DELETE ID FROM Subject WHERE ID = 7

strSQL = "DELETE ID FROM Subject WHERE ID = " & sid (DoCmd.RunSQL strSQL)
```

Import Excel file with long text fields

Access 97 will not create Memo text fields when importing an Excel file. Instead, the fields are imported as Text fields and truncated to 255 characters. (Apparently not so in Access 2000.) The solution:

- 1. In Excel, save the file as a CSV (comma delimited) file. Columns with commas in the text are enclosed in double quotes.
- 2. In Access, import the file as Text, with a comma delimiter and Text Qualifier = double quote.
- 3. After import, edit the table description to correct the field lengths.
- Illustrates:
- 1. use of IIf to programmatically set value of a field.
- 2. creation of a new field (usng IIf)
- 3. name change of existing field (AS xxx)

The IIf code is an IF ... THEN ... ELSE ... construct.

```
SELECT
[RCs with BF w/o NA].[RS CD],
[RCs with BF w/o NA].[BF CD],
[RCs with BF w/o NA].[CHAR TYPE CD],
[RCs with BF w/o NA].[CHAR SRC FLG],
IIf([CHAR TYPE CD]="AGENDUSE",[CHAR SRC FLG],"") AS AGENDUSE,
IIf([CHAR TYPE CD]="BILLER", [CHAR SRC FLG], "") AS BILLER,
IIf([CHAR TYPE CD]="CARE",[CHAR SRC FLG],"") AS CARE,
IIf([CHAR TYPE CD]="CUSTCRG",[CHAR SRC FLG],"") AS CUSTCRG,
IIf([CHAR TYPE CD]="DACREDIT", [CHAR SRC FLG], "") AS DACREDIT,
IIf([CHAR TYPE CD]="DCL", [CHAR SRC FLG], "") AS DCL,
IIf([CHAR TYPE CD]="ESP-CHRG",[CHAR SRC FLG],"") AS [ESP-CHRG],
IIf([CHAR TYPE CD]="ESP-DISC",[CHAR SRC FLG],"") AS [ESP-DISC],
IIf([CHAR TYPE CD]="ESP-KWH", [CHAR SRC FLG],"") AS [ESP-KWH],
IIf([CHAR TYPE CD]="MDMACRED",[CHAR SRC FLG],"") AS MDMACRED,
IIf([CHAR TYPE CD]="MTRCHRGE", [CHAR SRC FLG],"") AS MTRCHRGE,
IIf([CHAR TYPE CD]="SENIOR",[CHAR SRC FLG],"") AS SENIOR,
IIf([CHAR TYPE CD]="SERVPHAS",[CHAR SRC FLG],"") AS SERVPHAS,
IIf([CHAR TYPE CD]="TOT-CODE", [CHAR SRC FLG],"") AS [TOT-CODE], I
If([CHAR TYPE CD]="VOLTLVL",[CHAR SRC FLG],"") AS VOLTLVL,
IIf([APPL IN CONT SW]="N"," ",[APPL IN CONT SW]) AS [CONT RIDER],
IIf([VAL IN CONT SW]="N"," ",[VAL IN CONT SW]) AS [CONT VALUE],
IIf([EXEMPT IN CONT SW]="N"," ",[EXEMPT IN CONT SW]) AS [TAX EXEMPT]
FROM [RCs with BF w/o NA]
```

```
WHERE ((([RCs with BF w/o NA].[RS CD])="A10P" Or ([RCs with BF w/o NA].[RS CD])="E1" Or ([RCs with BF w/o NA].[RS CD])="E19S" Or ([RCs with BF w/o NA].[RS CD])="G1" Or ([RCs with BF w/o NA].[RS CD])="GNR1 "));
```

Character Replacement in SELECT

Can be done by a single query. Use a WHERE phrase to limit the character replacement to the records in error. If there is more than one character to replace, use a UNION.

```
SELECT ID, Filename, REPLACE(Filename, "%", " ") AS NewFilename, "%" As Error
FROM Test
WHERE InStr(Filename, "%") > -0
UNION
SELECT ID, Filename, REPLACE(Filename, "#", " ") AS NewFilename, "#" As Error
FROM Test
WHERE InStr(Filename, "#") > -0
SELECT ID, Filename, REPLACE(Filename, "~", " ") AS NewFilename, "~" As Error
WHERE InStr(Filename, "\sim") > -0
SELECT ID, Filename, REPLACE (Filename, "&", "") AS NewFilename, "&" As Error
FROM Test
WHERE InStr(Filename, "&") > -0
SELECT ID, Filename, REPLACE(Filename, "*", " ") AS NewFilename, "*" As Error
FROM Test
WHERE InStr(Filename, "*") > -0
UNTON
SELECT ID, Filename, REPLACE(Filename, "[", "") AS NewFilename, "[" As Error
FROM Test
WHERE InStr(Filename, "[") > -0
UNION
SELECT ID, Filename, REPLACE(Filename, "]", " ") AS NewFilename, "]" As Error
FROM Test
WHERE InStr(Filename, "]") > -0
SELECT ID, Filename, REPLACE(Filename, ":", " ") AS NewFilename, ":" As Error
FROM Test
WHERE InStr(Filename, ":") > -0
UNTON
SELECT ID, Filename, REPLACE(Filename, "<", " ") AS NewFilename, "<" As Error
FROM Test
WHERE InStr(Filename, "<") > -0
UNION
SELECT ID, Filename, REPLACE(Filename, ">", " ") AS NewFilename, ">" As Error
FROM Test
WHERE InStr(Filename, ">") > -0
UNION
SELECT ID, Filename, REPLACE(Filename, "?", " ") AS NewFilename, "?" As Error
FROM Test
WHERE InStr(Filename, "?") > -0
UNION
SELECT ID, Filename, REPLACE (Filename, "+", " ") AS NewFilename, "+" As Error
FROM Test
```

```
WHERE InStr(Filename, "+") > -0;
```

Creating a Table

You can create a table by importing an XLS file: with DoCmd.TransferSpreadsheet. The text field sizes are 255 and there is no Unicode compression (see DDL below).

You can create a table from the results of a SELECT query: SELECT * INTO MigrateWorksheet FROM FilesToMigrate where FilesToMigrate is a SELECT...FROM CandidateFiles query.

The field types, sizes, and Unicode compression are copied intact from the source table, except for type Hyperlink (called Anchor on the Documenter report) which reverts to text. There is no index.

[put INSERT statements here]

You can use a data definition query to change the field types and sizes. But it cannot set type = Hyperlink. And it cannot set compression on after the table has been created.

You can use a data definition query to set an index.

Data Definition Language (DDL)

You can create a table with SQL:

```
CREATE [TEMPORARY] TABLE table (field1 type [(size)] [NOT NULL] [WITH COMPRESSION | WITH COMP] [index1] [, field2 type [(size)] [NOT NULL] [index2] [, ...]] [, CONSTRAINT multifieldindex [, ...]])
```

About compression: Microsoft Access 2000 and later uses the Unicode character-encoding scheme to represent the data in a Text, Memo, or Hyperlink field. Unicode represents each character as two bytes, so the data in a Text, Memo, or Hyperlink field requires more storage space than it did in Access 97 or earlier, where each character is represented as one byte.

To offset this effect of Unicode character representation and to ensure optimal performance, the default value of the Unicode Compression property for a Text, Memo, or Hyperlink field is Yes. When a field's Unicode Compression property is set to Yes, any character whose first byte is 0 is compressed when it is stored and uncompressed when it is retrieved. Because the first byte of a Latin character—a character of a Western European language such as English, Spanish, or German—is 0, Unicode character representation does not affect how much storage space is required for compressed data that consists entirely of Latin characters.

How you create a table governs the value of compression:

- When you create a table with the Access GUI, the Text, Memo, and Hyperlink fields have by default compression = yes.
- When you copy a table, or create a table with a SELECT INTO query, the new table has the compression settings of the source table.
- When you create a table with DoCmd.TransferSpreadsheet, these fields have compression = no. If the amount of data is large, it can be better to create the table with the GUI or DDL SQL before running the TransferSpreadsheet; this will conserve file space.

Note: A DDL query cannot change the compression on an existing table.

You can change an existing table with SQL:

```
ALTER TABLE table {ADD {COLUMN field type[(size)] [NOT NULL] [CONSTRAINT index] |
ALTER COLUMN field type[(size)] |
CONSTRAINT multifieldindex} |
DROP {COLUMN field I CONSTRAINT indexname} }

You can add an index to an existing table with SQL:
CREATE [ UNIQUE ] INDEX index
ON table (field [ASC|DESC][, field [ASC|DESC], ...])
[WITH { PRIMARY | DISALLOW NULL | IGNORE NULL }]
```

Data Types

Hyperlink: A data type for an Access database field that stores hyperlink addresses. An address can have up to four parts and is written using the following format:

 $display text \verb|# address \verb|# subaddress \verb|#. This data type cannot be set in DDL SQL.$

Text: In a Microsoft Access database, this is a field data type. Text fields can contain up to 255 characters or the number of characters specified by the FieldSize property, whichever is less.

Memo: In a Microsoft Access database, this is a field data type. Memo fields can contain up to 65,535 characters.

Datetime: Stores date and time between years 100 and 9999.

Integer: Stores an long integer with a value between -2,147,483,648 and 2,147,483,647 in 8 bytes.

Decimal: Stores a decimal number in 17 bytes.

Money: Stores a number in 8 bytes.

Bit: Stores a binary (yes/no) value in 1 byte.

The following create table statement creates a test table with all of the DDL available to Microsoft Access data types:

```
CREATE TABLE TestAllTypes

(

MyText TEXT(50),

MyMemo MEMO,

MyByte BYTE,

MyInteger INTEGER,

MyLong LONG,

MyAutoNumber COUNTER,

MySingle SINGLE,

MyDouble DOUBLE,

MyCurrency CURRENCY,

MyReplicaID GUID,

MyPateTime DATETIME,

MyYesNo YESNO,

MyOleObject LONGBINARY,

MyBinary BINARY(50)
```

Note: You cannot create "AutoNumber Replication," "HyperLink," or "Lookup" type fields using a Microsoft Access DDL SQL statement. These field types are not native Jet field types and can be created and used only by the Microsoft Access user interface. The MyBinary field above is a special fixed-length binary field, which cannot be created via the Microsoft Access user interface but can be created using a SQL DDL statement.