7. Delimiters

A delimiter is a character used to mark the boundaries of a particular value or to separate different parts of data. For instance, in this example:

```
8:30 am
```

A colon (:) is used to separate hours from minutes.

When we reference a file on a drive, we use the backslash \ to separate directory names from each other and from the file name:

```
C:\Data\Documents\MyFile.Doc
```

Access has several different delimiters and each means something different. Some delimiters, such as quotes and hash marks are balanced like parentheses around a value and tell Access what data type to assign to the value. Other delimiters, like bang and dot, tell Access that what comes next is a member or a property.

"String in Double Quotes"

Double quotes are used to delimit strings

```
"this is a string of text"
```

```
"Mary's lamb has a hyphen which is okay because double-quotes
are used to delimit this string "
```

If you are delimiting a string with double quotes and want to use a double quote IN the string, use 2 of them.

```
"The width is 4"""
```

Will be displayed as → The width is 4"

When you are concatenating field and literal values, use quotes around the literal values. for instance, there needs to be a comma and a space between City and State

```
=[City] & ", " & [State]
```

the brackets are not necessarily necessary -- they are often used when typing what to do so that you know it is a fieldname that is being referenced.

'String in Single Quotes'

Single quotes are also used to delimit strings in queries and control source expressions. If a value in the string contains a double-quote, like writing four inches as 4", you can delimit the string with single quotes.

```
'The width is 4"'
```

There are many cases when single quotes don't work on the outside, like in an argument for dLookup (unless you are constructing it in code, then it does). In VBA code, you must use double-quotes as the outer delimiter when referencing strings.

#Date#

Hash marks are used to indicate that the value is a date/time data type. This is perhaps the data type that causes the most trouble. Internally, it is stored as a number yet it is shown to you in various formats. It also stores multiple pieces of information: year, month, day, hour, minute, and second.

Access stores date/times in a numeric format where the integer portion of the number represents the date and the decimal portion of the number represents time:

```
1/1/100 = -657,434
1/2/100 = -657,433
12/30/1899 = 0
1/1/1950 = 18,264
1/1/2005 = 38,353
12/31/2007 = 39,447
1/1/9999 = 2,958,101
12/31/9999 = 2,958,465
```

Time is a fraction of the day

```
12 noon is 0.5

6pm is 0.75

12/31/07, 12 noon = 39,447.5
```

If you have a control with just a date and you want to make sure it converts to a whole number, use:

DateValue([control_or_fieldname])
or
cLng([control_or_fieldname])

Likewise, if you only want the time component, you can use:

TimeValue([control_or_fieldname])

Since dates are whole numbers and times are the fractions, you can also do arithmetic operations on them. That is why you can subtract one date from another and get the number of days between the two.

If you have a date stored in text format, use cDate (convert to Date) around it:

cDate([textDate_controlname_or_fieldname])

… or concatenate the date with delimiters:

"#" & [textDate_controlname_or_fieldname] & "#"

Because dates can also have a time component, it is handy to use DateAdd, which let you specify the time interval (year, month, day, hour, etc) to add or subtract to calculate a new date or use DateDiff to get a numeric difference between dates for a specific interval.

The DateDiff function can be used to specify what time increment you want returned when you subtract dates. Likewise, there is a DateAdd function to add specific time increments to a date

Dates are stored as floating point numbers. This makes them inaccurate for equality comparisons -- the best way to ensure you have only the Whole part of the number (the date), is to use the Integer portion of the number (the date) only -- this, in essence, is what DateValue does. In addition to showing the result in a date format, it strips off the decimals.

For more information on dates:

International Dates in Access - Allen Browne
http://allenbrowne.com/ser-36.html

Brackets []

Brackets are used around names that you make up, like fieldnames, form names, report names, etc. If you have been good and didn't use spaces or special characters, you usually don't have to type them. In code, you can skip them unless they are necessary. In a query, Access will adjust what you type – but pay attention! Sometimes Access erroneously thinks what you typed needs quotes not brackets (for instance, in an UpdateTo cell).

```
[MyFieldname]
```

Take control references and variables out of a string

When you reference a control or variable, it comes OUT of the string

```
dLookup("Fieldname","Tablename" _
,"someID=" & Me.ID_controlname)
```

If the control you are referencing is a string value, use quotes to delimit the value

```
dLookup("Fieldname","Tablename" _
,"SomeName='" & Me.Text_controlname & "'")
```

if the control is a date value, use # to delimit the value

```
dLookup("Fieldname","Tablename" _
,"SomeDate=#" & Me.Date_controlname & "#")
```

Line Continuation _

While a Line Continuation character is not really a delimiter, it does have a special meaning when you are writing code. When you type a space and an underscore character at the end of a line, you are telling Access that the statement is continued on the next line.

Bang !

Use an exclamation point !, also called *Bang* (much easier to say <smile) when referring to a member of a collection

`Forms!Formname`	a particular form in the Forms collection
`Reports!Reportname`	a particular report in the Reports collection
`Me!fieldname`	a field in a form or report RecordSource
`rs!fieldname`	a field in a RecordSet

you can also identify members this way:

```
Forms("Formname")
Reports("Reportname")
```

this method offers greater flexibility because you can also do this:

```
Forms(stringVariableName)
```

And, in code …

```
'declare (dimension) variables you will use
dim rs as dao.RecordSet, mStr as string

'open a recordset
Set rs = CurrentDb.OpenRecordset("MyTable")

'if not at the end, store a value in a variable
If Not rs.Eof then
   mStr = rs!SomeStringField 'you HAVE to use Bang to refer to field in Recordset
End If

'close recordset and release object variable
rs.Close
Set rs = Nothing
```

The Bang allows you to "Late Bind", so if referring to a Field from a RecordSource, you do not have to have the RecordSource set in order to compile, because VBA will *not* verify that the object/field you specify after the bang is present upon Compile, VBA just assumes its there. When you are executing your code, VBA will try to resolve the reference and, if VBA can not resolve it, and error is raised. This concept of "late binding" is very handy if, for instance, you use VBA code to manipulate or set the RecordSource of your form upon form load.

Dot .

Use a Dot (period) when referring to a Property or Method that belongs to an object.

`Me.NewRecord`	NewRecord property (true or false) of a form
`rs.AddNew`	AddNew record method of a Recordset
`rs.Update`	Update (save) method of a Recordset
`Me.controlname`	Name of control in controls collection of form or report

You can use Me.fieldname, Me!fieldname, or Me.controlname when you are behind a form or report. I've heard that Bang has a slight speed advantage but I usually use Dot for the IntelliSense advantage while coding.