

Records Types and Classes in VB6

Malcolm Moore
© 09-Sep-2009

VB6 - Dimensioning a Record Set

A recordset is a grouping of associated variables, and usually in that grouping many of these variables are different types of data. The problem is that if you want to group these variables, then in some variable naming standards, the prefix is different – denoting a special type of data – for example a String or a Long Integer, or a Single Precision number.

One way around all this is to use the dot connect format and this makes the associated different types of data associated by a common name, and this assists in developing the software as the VB6 development program will prompt you, minimising typos in many ways. If we wanted to associate five variables as a group, the following example shows the development process.

```
Public Type Buf
    Symbol as String
    Price as Single
    Volume as Long
    EpochTime as Long
    MMDateTime as String * 20
End Type
```

We now have five variables all associated by the common prefix “**Buf.**” and all these variables can be seen as associated, because when typing in “**Buf.**” (without the double quotes) a drop-down list will appear providing the choice of the five variables. If needed, an indicator of the data type for each variable can be included by a lower case letter – or letters in front of the data type.

In practice, the string variable for the `Symbol` would be `Buf.Symbol`, the single precision variable for the Price would be `Buf.Price` etc. so the various data variables are associated, and it is quick and largely typo free.

When programming, I prefer to use Title Case (where the Variables are given a leading Upper Case letter and the rest is all lower case), because when typing in a program, sometimes the drop-down assist does not come into play – like when naming database table headings; and when you finish the line, the VB6 program associates the variables and brings the all lower case into Title Case. If it does not jump into title Case, then we immediately know that we have a data association problem, and it can be addressed and resolved immediately, not several days or weeks later!

Dimensioning a Data Array Record Set

With the recordset for one row of data dimensioned as in the above commonly dimensioned recordset, this is one line away from dimensioning a total array. By adding a common array dimensioning statement linking the recordset, after dimensioning that recordset, the recordset is included into an array. Here is an example:

```
Public Buffer (0 to 5000) as Buf
```

We could have put simply 5000 instead of 0 to 5000 as it meant exactly the same – noting that arrays start from 0, unless we specifically instruct otherwise – for example 3 to 5000. Note that this number range is a long integer, and if we wish to use a variable to specify the row position of the recordset, then this variable must be dimensioned as a long integer.

Now the development is slightly different as the data array takes its form. Consider that we are going to put a value eg “AAC” into the first (zeroth) row of the Symbol variable, then the syntax would be:

```
Buffer(0).Symbol = "AAC"
```

This is very powerful syntax and it is not until we have used it for some time that its power really becomes noticeable, as it prompts, minimises typos and makes it rather easy to remember, especially when we may have several arrays working together. Also, there is nothing stopping us from having more than one array, using one commonly dimensioned recordset reference.

Making a Database Table Record Class Array

A Class is a group of variables that are tied together by a common parent heading. This concept was never common in early languages but with Object Oriented programme language development, Classes have become rather common as they provide immediate hints on their structures and that greatly simplifies the process of engineering a program.

The tie between a parent variable title and its children is a ‘dot’ and most advanced programming languages understand the dot to be the prompt to provide a drop down menu of the children from that point and it is then a case of highlight and paste instead of type it in (occasionally with errors).

Here is the construction of a typical Class to do End Of Day (EOD) data in the Stock Market:

```
Private Type EndOfDay
    fOpen As Single
    fHigh As Single
    fLow As Single
    fClose As Single
    lVolume As Long
    lDate As Long
End Type

Public EOD As EndOfDay
```

The first part is to Privately define a 'Type' and give it an arbitrary name, then define the components of the type as you normally would have done, relating the dimension structure by the leading lower case letter, then end the Type definition.

While constructing a program in VB6 with the annotation on, strictly, the leading (lower case) letter to hint about the variable format is not necessary, as the VB6 annotation will pick up the field formatting and display that as you edit the programming.

Now here is the 'trick' (in the last programming line shown above)! Publicly dimension the Type (which in this case is EndOfDay) as EOD, and this statement links EOD to the defined variables with a dot and this dynamically links the defined variables as follows:

EOD. Invokes the Type List, so then it is a simple case of highlighting the appropriate variable and <Enter> or <Click> and the Variable is entered – without mistakes! For example, the Close Price would have previously been written as fClose = 4.34 for example, but with this in a Class the structure would be EOD.fClose = 4.34 and now you know that fClose relates directly to the EOD data and nothing else.

If for example, we wished to create more data variables say for the End Of Week, we could fairly easily use a very similar structure as follows:

```
Private Type EndOfDay
    fOpen As Single
    fHigh As Single
    fLow As Single
    fClose As Single
    lVolume As Long
    lDate As Long
End Type
Public EOW As EndOfDay
Public EOD As EndOfDay
```

So now we have two Classes: EOW (as an End of Week), and another: EOD (as an End Of Day) and both of these two Classes are separate and independent.

In this example, the closing price at the end of the week would be shown as EOW.fClose, which is very different from EOD.fClose but it has the same dimensions of (f leading) Single Precision. Note that the date in this case is a Long Integer; and it could have been set as a String.

With very little lateral thinking, the data types as described above could be seen as a recordset that ties an array of linked values to a events, so the Type could be made into a virtual data Table if the second stage of the dimension statement was made into an Array as follows:

```
Public Type EndOfDay
    fOpen As Single
    fHigh As Single
    fLow As Single
    fClose As Single
    lVolume As Long
    lDate As Long
End Type

Dim EOD(100) As EndOfDay
```

In this case the array is static, but it could be far more flexible if done as a dynamic array:

`Dim EOD() As EndOfDay` and this is one step away from being a database table that could the features of SQL like sorting, finding, matching and relating, that without using the Class features can be an unholy mess.

For example the close price for the first recordset (1) could be entered as follows:

`EOD(1).fClose = 2.45`

Object Oriented Programming thrives under this structure as all the attributes of an Object are (should be) tied to that Object, which is the parent prefix. So each object has its own set of attributes. This process can be taken a little further if there are many attributes, and-or several parts to an object, and in this case the class can extend over more than one level of definition, for example: