Article: Excel - Using Arrays in VBA
Date: 01/12/2011
Posted by: HeelpBook Staff
Source: Link
Permalink: Link

Create an Array in VBA

You can declare an **array** to work with a set of values of the same data type.

An **array** is a single variable with many compartments to store values, while a typical variable has only one storage compartment in which it can store only one value. Refer to the **array** as a whole when you want to refer to all the values it holds, or you can refer to its individual elements.

For example, to store *daily expenses for each day of the year*, you can declare one array variable with **365 elements**, rather than declaring **365 variables**. Each element in an array contains one value. The following statement declares the array variable curExpense with **365 elements**.

By default, an **array** is indexed beginning with zero, so the *upper bound* of the array is **364** rather than **365**.

```
Dim curExpense(364) As Currency
```

To set the value of an individual element, you specify the element's index. The following example assigns an initial value of **20** to each element in the array.

```
Sub FillArray()
Dim curExpense(364) As Currency
Dim intI As Integer
For intI = 0 to 364
curExpense(intI) = 20
Next
End Sub
```

Changing the Lower Bound

You can use the **Option Base** statement at the top of a module to change the default index of the first element from 0 to 1. In the following example, the **Option Base** statement changes the index for the first element, and the Dim statement declares the array variable **curExpense** with **365 elements**.

```
Option Base 1
Dim curExpense(365) As Currency
```

You can also explicitly set the lower bound of an array by using a **To** clause, as shown in the following example.

```
Dim curExpense(1 To 365) As Currency
```

```
Dim strWeekday(7 To 13) As String
```

Storing Variant Values in Arrays

There are two ways to create arrays of **Variant** values. One way is to declare an array of Variant data type, as shown in the following example:

```
Dim varData(3) As Variant
varData(0) = "Claudia Bendel"
varData(1) = "4242 Maple Blvd"
varData(2) = 38
varData(3) = Format("06-09-1952", "General Date")
```

The other way is to assign the array returned by the **Array** function to a **Variant** variable, as shown in the following example.

```
Dim varData As Variant
varData = Array("Ron Bendel", "4242 Maple Blvd", 38, _
Format("06-09-1952", "General Date"))
```

You identify the elements in an array of Variant values by index, no matter which technique you use to create the array. For example, the following statement can be added to either of the preceding examples.

```
MsgBox "Data for " & varData(0) & " has been recorded."
```

Using Multidimensional Arrays

In **Visual Basic**, you can declare arrays with up to 60 dimensions. For example, the following statement declares a *2-dimensional*, 5-by-10 array.

```
Dim sngMulti(1 To 5, 1 To 10) As Single
```

If you think of the **array** as a matrix, the first argument represents the rows and the second argument represents the columns.

Use nested **For...Next** statements to process *multidimensional* arrays. The following procedure fills a two-dimensional array with Single values.

```
Sub FillArrayMulti()
Dim intI As Integer, intJ As Integer
Dim sngMulti(1 To 5, 1 To 10) As Single

' Fill array with values.
For intI = 1 To 5
For intJ = 1 To 10
sngMulti(intI, intJ) = intI * intJ
Debug.Print sngMulti(intI, intJ)
Next intJ
Next intI
End Sub
```