NetMedia

Basic Express Application Note

Block Data Classes

Introduction

This application note describes how to use block data classes in a BasicX system.

Array initialization issues

One problem with conventional arrays is that there is no easy way to initialize them without incurring a performance penalty. Traditional Basic dialects use DATA statements for similar purposes, but DATA statements are awkward to use and are not VB compatible.

BasicX addresses this problem by providing system-defined block data classes, which are the equivalent of initialized arrays stored in EEPROM memory:

In this example, we'll use a 1-D array of bytes. The first step is to declare a block data object, which must be in module-level code:

```
Private Vector As New ByteVectorDataRW
```

The next step is to define the source of the data:

```
Call Vector.Source("ByteVector.txt")
```

The Source method specifies a text file located on the PC. Although it is legal to specify a full pathname, in this example we'll omit the path, which means the file is located in the same subdirectory as the BXP project file. Note that the Source method must be called before any other access to the block data object:

Once the object is defined, we can read and write to it as though it were an array. In this case, the array happens to reside in EEPROM memory, which is also where the program is stored. Syntax:

```
Dim B As Byte

' Write to element 1.
Vector(1) = B

' Read element 1.
B = Vector(1)
```

Warning -- a block data object is similar to a persistent variable in regards to write cycle limitations and the amount of time it takes to write to the object.

Example Program

This program reads and writes the first 5 elements of a byte vector object:

```
Private Vector As New ByteVectorDataRW   ' Read-write.

Public Sub Main()

    Dim N As Byte
    Dim B As Byte

    Call Vector.Source("ByteVector.txt")

    Debug.Print
    Debug.Print "   Data using object access:"

    For N = 1 To 5
        Debug.Print "      Vector("; CStr(N); ") = ";
        Debug.Print Cstr(Vector(N))
    Next

    Debug.Print
    Debug.Print "   EEPROM address of Vector:  ";
    Debug.Print CStr(Vector.DataAddress)

    Debug.Print
    Debug.Print "   Data using direct EEPROM access:"

    For N = 0 To 4   ' Note shift to 0-base.
        Call GetEEPROM( Vector.DataAddress + CLng(N), B, 1 )
        Debug.Print "      Vector("; CStr(N+1); ") = "; CStr(B)
    Next

    ' Modify the second element upon first download.
    If ( FirstTime() ) Then
        Vector(2) = 255
    End If

End Sub
```

Text file ByteVector.txt contains 5 lines of text:

```
10
20
30
40
50
```

This is the output of the program after download:

```
Data using object access:
   Vector(1) = 10
   Vector(2) = 20
   Vector(3) = 30
   Vector(4) = 40
   Vector(5) = 50

EEPROM address of Vector:  1495

Data using direct EEPROM access:
   Vector(1) = 10
   Vector(2) = 20
   Vector(3) = 30
   Vector(4) = 40
   Vector(5) = 50
```

Note that the FirstTime function causes the program to run differently the first time after download. The second time you run the program after a download, this is the output:

```
Data using object access:
   Vector(1) = 10
   Vector(2) = 255
   Vector(3) = 30
   Vector(4) = 40
   Vector(5) = 50

EEPROM address of Vector:  1495

Data using direct EEPROM access:
   Vector(1) = 10
   Vector(2) = 255
   Vector(3) = 30
   Vector(4) = 40
   Vector(5) = 50
```

You can see that the value of element 5 was changed from 20 to 255.

Also note that you can get access to the EEPROM address of the object. In this case, the object is located at address 1495. You can double check the address by referring to the MPP map file, which lists the location of the object, as well as the full PC pathname of the text file ByteVector.txt.

Other block data classes

The above example illustrates how to use a 1-D Byte array. You can also declare 2-D arrays of type Byte, Integer, Long and Single. In addition, you can control whether the object is read-only, or read-write.