AN00113-001 - Downloading data to MintMT

Related Applications or Terminology

- **Downloading variable data to MintMT**
- **Use of ActiveX in Visual Basic**

**Overview**

ActiveX controls, formerly known as OLE controls or OCX controls, are components (or objects) that can be inserted into an application to reuse packaged functionality. For example, the ActiveX controls that are included with MintMT controllers allow PC applications to communicate with controllers to allow complete machine control from a PC.

A key advantage of ActiveX controls is that they can be used in applications written in many programming languages, including development environments such as:

- Microsoft Visual C++
- Microsoft Visual Basic
- Borland Delphi
- National Instruments LabView

Any development environment that supports ActiveX controls can use the MintMT ActiveX control making it a very versatile interface to the controller.

The MintMT ActiveX Control is installed as part of Workbench v5 and is suitable for use under Windows 95, 98, NT, 2000 and ME.

Often a MintMT application requires data from an external source, such as an Excel spreadsheet for example, to be utilized. An example of this could be elements of a Cam table derived from complex formulae embedded within the spreadsheet.

MintMT allows variables (including array information) to be updated at run-time via the **VariableData** property of the ActiveX control. This application note describes a simple method for transferring data from Excel into the MintMT run-time environment.
Creating a simple MintMT program

For the purposes of this application note we will create a very simple MintMT program that effectively loops indefinitely. This program will include a declaration for a 10-element array of integer values.

Our MintMT code will be entered as follows:

```plaintext
Dim ExcelData(10) As Integer

'Set array contents to 0 initially
ExcelData = 0;

Loop
    Wait = 1000
End Loop
```

The array has been declared at a global level and therefore forms part of the ParentTask environment.

Compile, download and run this program. We can now monitor the contents of the array via the variable watch window in Workbench v5. Whilst in the Edit and Debug workspace, select the Watch tab in the Output pane.

To monitor a variable or an array, click on the Add button and then enter the name of the variable or array to monitor. In our example we want to monitor the contents of the ExcelData array.

After entering the name Workbench will continuously monitor the value of the variable (or the contents of the array) at a rate determined by the drop down box on the right hand side of the dialogue.

Further variables or arrays can be added to this list by repeating the above procedure.

Double click the edge of the Value column to expand the width of this column to suit the amount of data to be displayed. The dialogue will look something like the one shown below...

Our MintMT program set the contents of the ExcelData array to 0 initially and the variable watch window confirms this.
Importing Data

There are many different ways in which we could transfer data from a host application to our MintMT controller using the ActiveX control’s properties or methods. Examples of these include:

- Writing to the Comms array
- DataFile download
- VariableData transfer

This application note concentrates on the VariableData property of the MintMT ActiveX control and utilizes a Visual Basic programming example.

Creating a simple Visual Basic application

Start Visual Basic and open a new Standard Exe project. We need to add the MintMT ActiveX control to the project so select the Project > Components menu. Scroll through the list of components installed on the PC until you reach MintControls for MintMT. The Workbench v5 installation installs every version (build) of this control for backwards compatibility so select the latest version and click on OK.
Three new icons will be added to the Visual Basic toolbox. We are only interested in the Mint Controller for the purposes of this example:

Select the MintController control, the cursor will change to a crosshair. Click and drag anywhere on the form, release the mouse button and a Mint icon will appear.

The default name for this control is MintController1 – this can be modified if required via the Properties window for the control. We will leave it named as MintController1.

We now need to add some code to establish a connection to the MintMT controller when the Visual Basic application starts (the exact code required will vary from controller to controller – refer to the PC Programming Guide MN1906 for further details).

In this example we will be connecting to a Mintdrive II (configured as Node 2 and with its serial port setup for operation at 57.6kBaud).

Double click the form and the existing code for the Form Load event will appear. Click somewhere within this procedure and start enter the name of the Mint control (MintController1). Now enter a dot (.) – Visual Basic will automatically pop up a dialogue listing all of the available ActiveX methods and properties.
Scroll through the list and select SetMintDrive2Link and then press the Spacebar.

Visual Basic will now display the prototype for the ActiveX method or property and highlight the particular parameter that should be entered:

```
setMintDrive2Link(mNodeNumber As Integer, nCommPort As Integer, iBaudRate As Long, bOpenPort As Boolean)
```

Our drive is configured as Node 2. It is connected to Com1: on our PC and is operating at 57600 baud. We wish to connect to it now so we should complete the code as follows:

**MintController1.setMintDrive2Link 2,1,57600,True**

This is all that is required to establish a connection to the drive.

Run this program. If all is correct (i.e. the node address, baud rate etc.. are as expected) a blank form should appear. Close this and save the Visual Basic project and form.

**Reading Data from Excel**

We now need to read data from some cells in an Excel spreadsheet. Again there are many ways to achieve this but we will make use of the Excel object model.

First we need to add the Excel object library to our list of project references. Select Project > References and scroll through the list until a reference to the Microsoft Excel Object Library is found (we have installed Office 2000 so we have located the Excel 9.0 library).
Select this and click on OK to continue.

Return to the Visual Basic code editor and navigate to the General – Declarations section for our Form. Add the code shown below to declare some Excel objects:

```vbnet
Dim app As New Excel.Application
Dim Book As Excel.Workbook
Dim Sheet As Excel.Worksheet
```
Return to the Form design window and select the Visual Basic command button tool from the toolbox. Click and drag a button onto the form. Again the properties window can be used to rename this control if necessary (we called ours cmdDownload and changed the caption to Download).

Double-click the button and the code window for the button’s click event will open.

Enter the code shown below:

```vba
Private Sub cmdDownload_Click()
    Set Book = Workbooks.Open("c:\Data.xls")
    Set Sheet = Book.ActiveSheet

    Dim XLData(1 To 10) As Long
    Dim idx As Integer

    For idx = 1 To 10
        XLData(idx) = Sheet.Cells(idx, 1)
    Next idx

    MintController1.VariableData("ParentTask", "ExcelData") = XLData

    Book.Close

    Set Book = Nothing
    Set Sheet = Nothing
    Set App = Nothing

End Sub
```

This code performs a number of functions:

- Sets the path to our Workbook (in this example this is c:\Data.xls)
- Sets our Worksheet to be the active sheet in the Workbook (our example Excel file will only include 1 sheet)
- Declares a local 10 element array and reads the contents of the first ten rows in column 1 of our spreadsheet into this array
- Uses the VariableData property of the MintMT ActiveX control to pass our local array to the array defined in our MintMT program – note that this property requires the name of the MintMT task in which the target array or variable is declared (in this case the ParentTask) and the name of the MintMT array or variable (in this case ExcelData)
- Clears/Closes all of the Excel objects

We now need to create the spreadsheet itself. Run Excel and enter 10 numbers in the first 10 rows of column A. We chose to enter the numbers 10 to 1 in descending order:
Close Excel and save the file as c:\Data.xls (i.e. it must match the path specified in the Visual Basic code for the button’s click event).

Run the Visual Basic program and click on the Download button.

Return to Workbench v5 and reopen the Watch for the ExcelData array. It should now reflect the data imported from the Excel spreadsheet: