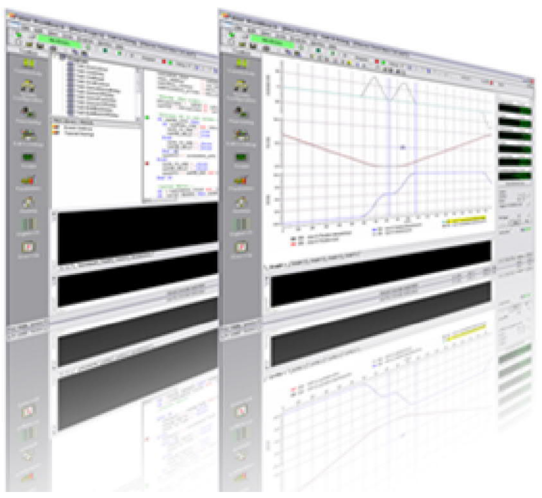
Motion Control Products

Application note
Accessing Mint variables via ActiveX

AN00113

Rev B (EN)

Mint program variables can be accessed (both read and write) directly using the ActiveX controls installed as part of Mint Workbench



Introduction

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++/C#
- Microsoft Visual Basic
- Borland Delphi
- National Instruments LabView
- Microsoft Office applications

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

The Mint ActiveX control is installed automatically as part of Mint Workbench, or can be installed on its own as part of the Mint Workbench installation process. It is suitable for use under Windows 95, 98, NT, 2000, Vista, XP, 7 and 10 (depending on ActiveX version utilized).

Often a Mint 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. Or a host PC application may want to access program variables directly to display information about machine operation to a user.

Mint allows all 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 Mint run-time environment.

Creating a simple Mint program

For the purposes of this application note we will create a very simple Mint program that effectively loops indefinitely. This program will include a declaration for a 10-element array of integer values and a variable we can use as a counter to read later on. Our Mint code will be entered as follows:

```
Dim ExcelData(10) As Integer
Dim Counter As Integer

'Set array contents to 0 initially
ExcelData = 0;

Loop
   Wait (1000)
   Counter = Counter + 1
End Loop
```

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

Start Mint Workbench and connect to the Virtual Controller (a virtual NextMove e100). Compile, download and run this program. We can now monitor the contents of the array via the variable watch window in Workbench. Whilst in the Edit and Debug workspace, select the Watch tab in the Watching pane.

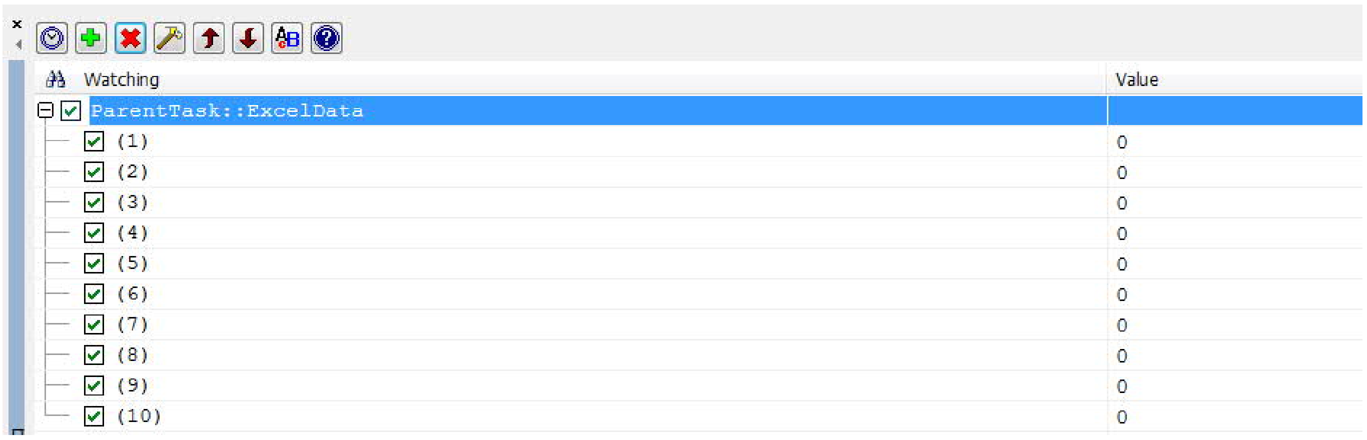
To monitor a variable or an array, click on the Add button and then enter the name of the variable or array to monitor, pre-fixing this with the name of the task the data is declared in and ::.
In our example we want to monitor the contents of the ExcelData array that was declared in the ParentTask so we would enter…

ParentTask::ExcelData

Alternatively you can right click the data you want to monitor in the program editor and select ‘Send ParentTask::ExcelData to Watching window’.

After adding the watch, Workbench will continuously monitor the value of the variable (or the contents of the array) at a rate determined by the settings accessed via the ‘clock’ button above the variable watch. Further variables or arrays can be added to this list by repeating the above procedure.

Because we have added an array to the watch window it will initially be displayed in its collapsed state (i.e. just the array name will be displayed with a + icon next to it). Expand this to view the entire array contents as shown below…



Watching	Value
ParentTask::ExcelData	
(1)	0
(2)	0
(3)	0
(4)	0
(5)	0
(6)	0
(7)	0
(8)	0
(9)	0
(10)	0

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

Importing Data

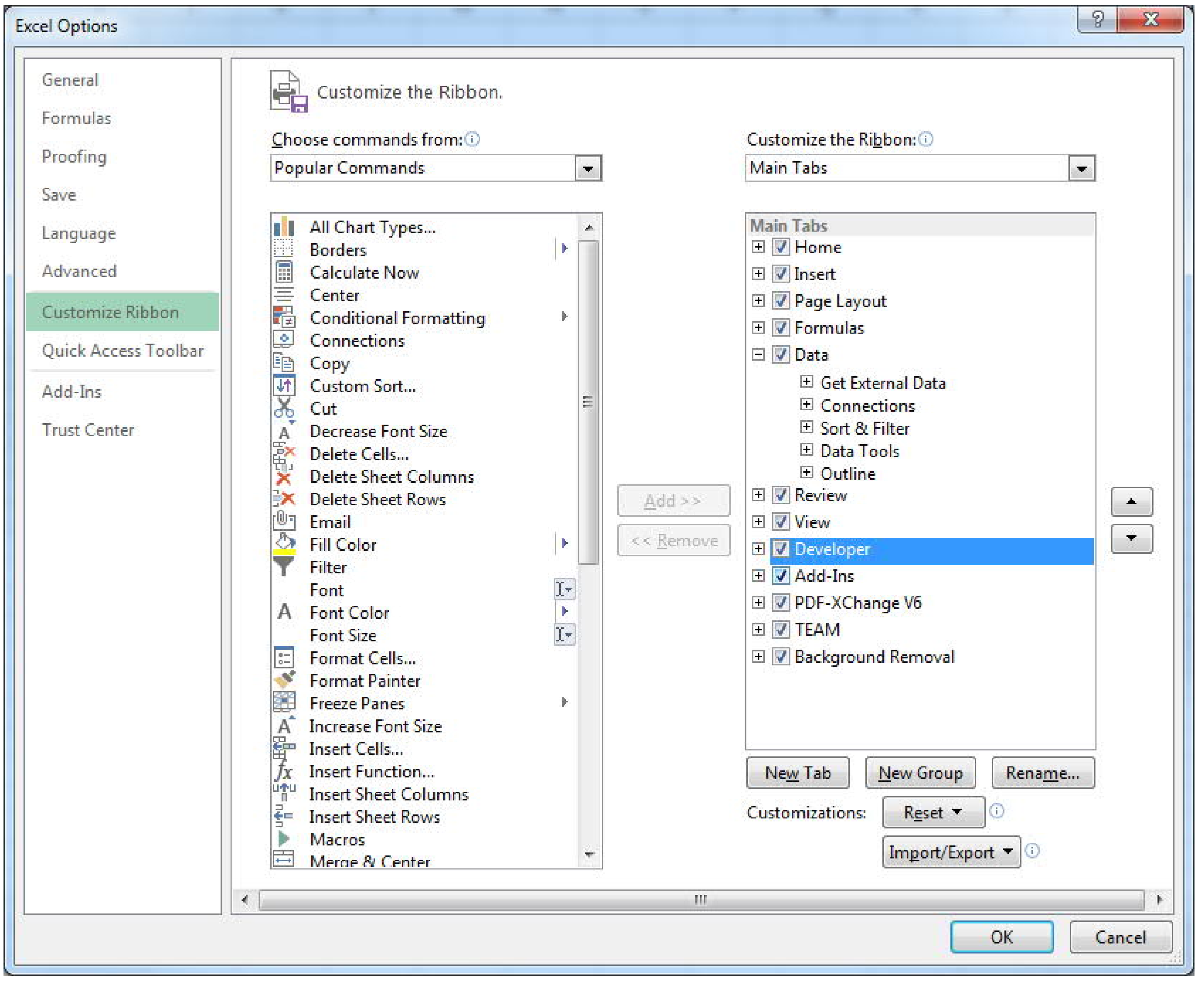
There are many different ways in which we could transfer data from a host application to our Mint controller using the ActiveX control's properties or methods. Examples of these include:

- Writing to the Comms array
- Writing to the Netdata array
- DataFile download (downloading a whole set of arrays)
- VariableData transfer (to access individual variables or arrays)

This application note concentrates on the VariableData property of the Mint ActiveX control and utilizes a Microsoft Excel 2013 programming example, but the same principles apply to any development environment supporting the use of ActiveX / COM controls.

Creating a simple Excel example

Start Microsoft Excel – by default you may not have the 'Developer' toolbar available. The method of adding this varies according to the version of Excel being used….in Excel 2013 this is added by selecting the 'File' tab, clicking 'Options' to enter the Excel options dialog, clicking 'Customise ribbon' and then in the right hand pane selecting 'Main Tabs' from the 'Customise ribbon' drop down list. Ensure the 'Developer' check box is selected…



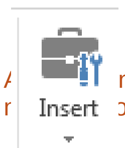
Choose commands from: Popular Commands

Customize the Ribbon: Main Tabs

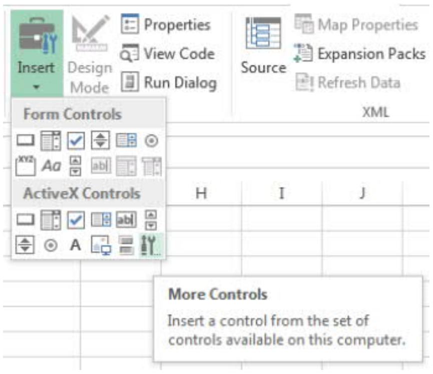
☑ Home, ☑ Insert, ☑ Page Layout, ☑ Formulas, ☑ Data, ☑ Review, ☑ View, ☑ Developer, ☑ Add-Ins, ☑ PDF-XChange V6, ☑ TEAM, ☑ Background Removal

Customizations: Reset

Once the developer tab is present in the Excel main menu you can now add the Mint ActiveX controls to the workbook/sheet. Click on the 'Developer' tab and then click on 'Insert' button…

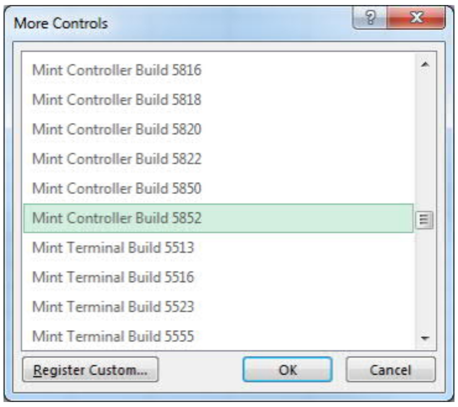


Now click on the ‘More Controls’ button in the Control Toolbox….



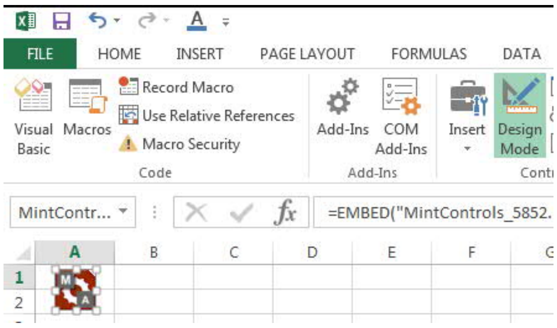
Clicking the ‘More Controls’ button will cause your PC to read the system registry to discover what ActiveX controls are installed so don’t be worried if there is a small delay whilst it does this

From the list of available ActiveX controls select the latest version of the ‘Mint Controller’ (in our example this is Build 5852). Be sure to select the Mint Controller as there are also controls available for the Mint Terminal Window and the Mint Command Prompt...

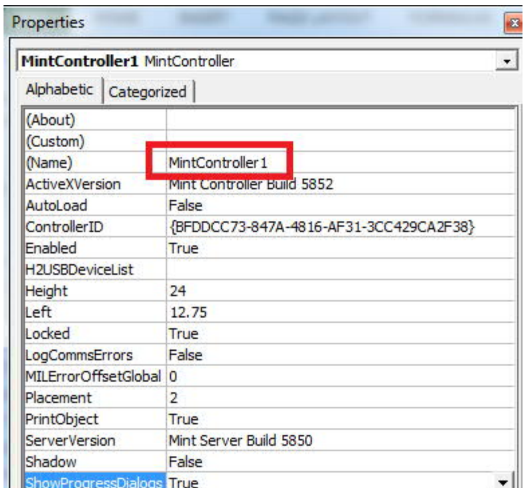

More Controls

- Mint Controller Build 5816
- Mint Controller Build 5818
- Mint Controller Build 5820
- Mint Controller Build 5822
- Mint Controller Build 5850
- Mint Controller Build 5852
- Mint Terminal Build 5513
- Mint Terminal Build 5516
- Mint Terminal Build 5523
- Mint Terminal Build 5555

The cursor will change to a crosshair – click and drag the mouse to draw a box on the Excel sheet (putting this at cell A1 is a good idea so you know where it is later).



MintContr... =EMBED("MintControls_5852.


Properties

MintController1 MintController

(About)	
(Custom)	
(Name)	MintController1
ActiveXVersion	Mint Controller Build 5852
AutoLoad	False
ControllerID	{BFDDCC73-847A-4816-AF31-3CC429CA2F38}
Enabled	True
H2USBDeviceList	
Height	24
Left	12.75
Locked	True
LogCommsErrors	False
MILErrorOffsetGlobal	0
Placement	2
PrintObject	True
ServerVersion	Mint Server Build 5850
Shadow	False
ShowProgressDialogs	True

Right click on the Mint icon and select ‘Properties’ from the resulting menu. The Properties dialog will show a name for the control (MintController1 by default). This name can be changed if required to something more meaningful to the application…we might call it ‘axVC’ for example (for ActiveX Virtual Controller)

Using a Reference instead of an ActiveX object

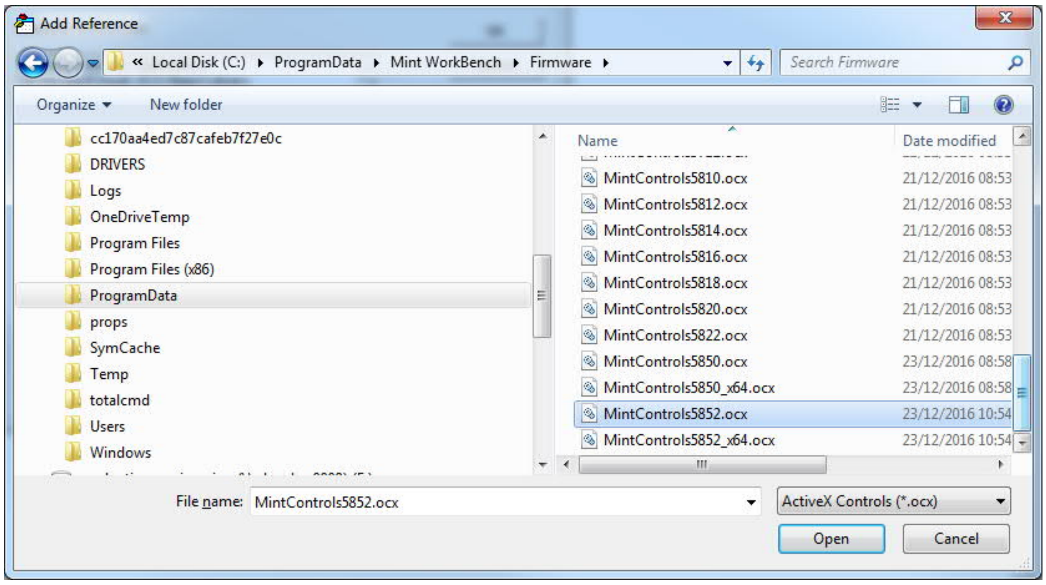
Some applications may not provide a 'container' for the ActiveX control to be placed in (such as a form or a sheet). In these cases it is necessary to create a reference to the ActiveX control instead of placing an instance of the control in the container.

To include a Reference to the Mint ActiveX Control navigate to the VBA Editor (the procedure for this will vary according to the application in use).

In Excel 2013 the VBA editor is displayed by clicking on the 'View code' button on the Developer tab

Once in the VBA editor environment select Tools>References… from the top menu and then use the Browse… button to navigate to the installation directory for the Mint ActiveX components (e.g. if using Windows 7 you will find this at C:\ProgramData\Mint Workbench\Firmware by default).

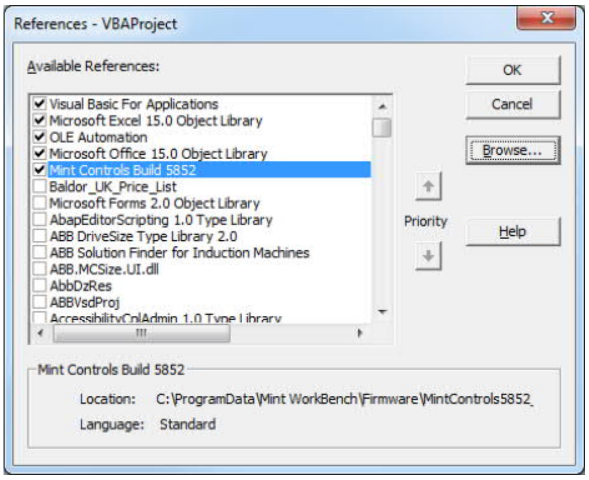
Change the 'Files of Type' dropdown to 'ActiveX Controls (*.ocx)' and the available references will be shown (again select the latest version – 5852 in our example below). Click on 'Open' and then on 'OK' at the next dialog.


Add Reference
« Local Disk (C:) ▸ ProgramData ▸ Mint WorkBench ▸ Firmware ▸
Search Firmware
Organize ▾
New folder

cc170aa4ed7c87cafeb7f27e0c
DRIVERS
Logs
OneDriveTemp
Program Files
Program Files (x86)
ProgramData
props
SymCache
Temp
totalcmd
Users
Windows

Name	Date modified
MintControls5810.ocx	21/12/2016 08:53
MintControls5812.ocx	21/12/2016 08:53
MintControls5814.ocx	21/12/2016 08:53
MintControls5816.ocx	21/12/2016 08:53
MintControls5818.ocx	21/12/2016 08:53
MintControls5820.ocx	21/12/2016 08:53
MintControls5822.ocx	21/12/2016 08:53
MintControls5850.ocx	23/12/2016 08:58
MintControls5850_x64.ocx	23/12/2016 08:58
MintControls5852.ocx	23/12/2016 10:54
MintControls5852_x64.ocx	23/12/2016 10:54

File name: MintControls5852.ocx
ActiveX Controls (*.ocx)



Available References:

- [✓] Visual Basic For Applications
- [✓] Microsoft Excel 15.0 Object Library
- [✓] OLE Automation
- [✓] Microsoft Office 15.0 Object Library
- [✓] Mint Controls Build 5852
- [ ] Baldor_UK_Price_List
- [ ] Microsoft Forms 2.0 Object Library
- [ ] AbapEditorScripting 1.0 Type Library
- [ ] ABB DriveSize Type Library 2.0
- [ ] ABB Solution Finder for Induction Machines
- [ ] ABB.MCSize.UI.dll
- [ ] AbbDzRes
- [ ] ABBVsdProj
- [ ] AccessibilityCplAdmin 1.0 Type Library

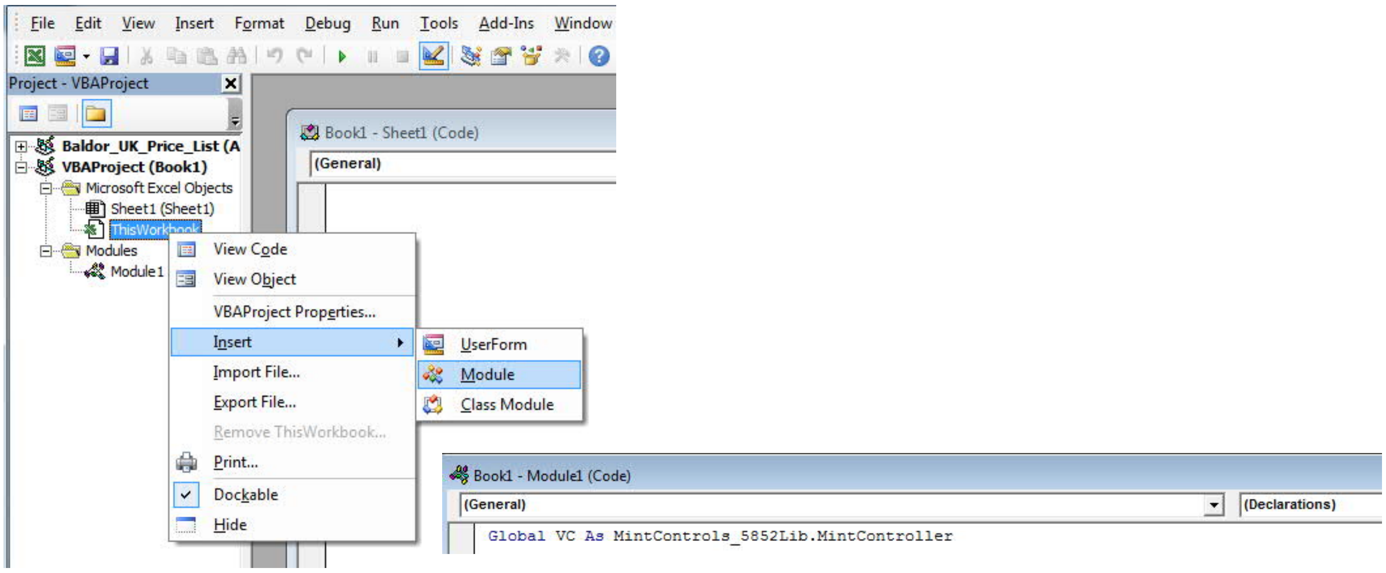
Mint Controls Build 5852
Location: C:\ProgramData\Mint WorkBench\Firmware\MintControls5852_
Language: Standard

Now the Reference has been created we need to declare an instance of the MintController in the VBA application (and make this globally available to the whole project).

In Excel we could do this by including a new VBA code module in the Workbook and adding the line…

Global VC As New MintControls5852Lib.MintController

…to the General>Declarations section of the module (Note that ‘VC’ is the name we’ve decided to give the control in this example, in much the same way as the Properties dialog could be used to give a name such as ‘axVC’ to the ActiveX component earlier).



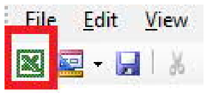
```
Global VC As MintControls_5852Lib.MintController
```

…and then including a subroutine in this code module to initialise an instance of this as follows:

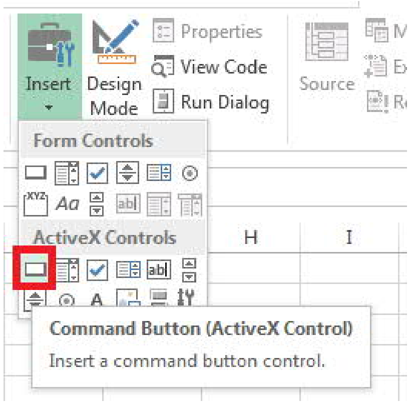
```
Sub Initialise()
 Set VC = New MintControls_5852Lib.MintController
End Sub
```

Establishing Communications

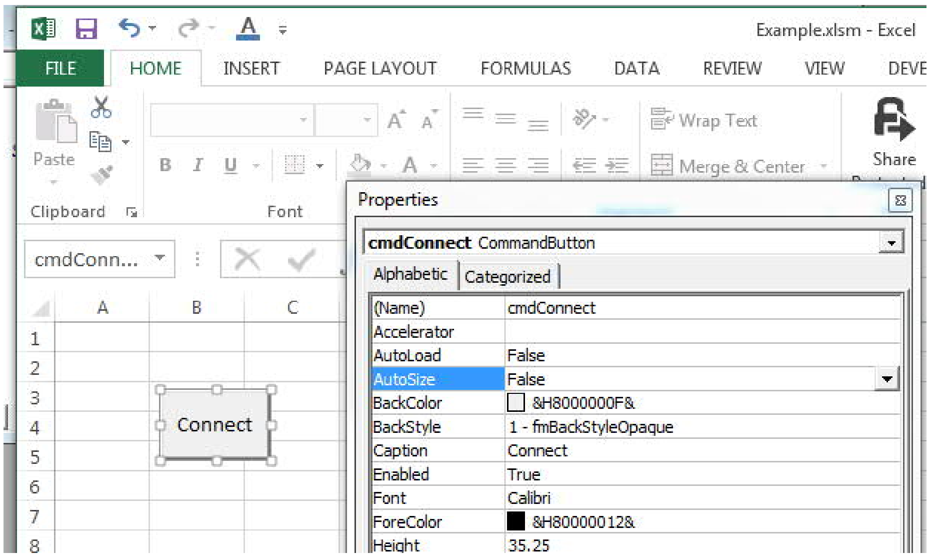
Before the ActiveX control can be used to issue any Mint commands we need to establish a communications link between the PC and the Mint controller. We will add a button to our Excel sheet that does this when pressed (to automate the process as far as possible this could be added to the inbuilt Excel ‘Workbook – Open’ procedure instead, but closing and opening the Excel document during development is time consuming so we’ve stuck to using a button for this example).

Switch back to the standard Excel view by clicking the Excel icon button 

Click on Insert again and this time select a command button from the tool palette…



The cursor will now let you draw a button on the Excel sheet….once added to the sheet right click the button and access the Properties dialog. Give the button a name (we called ours cmdConnect) and change the Caption property to ‘Connect’ for example….



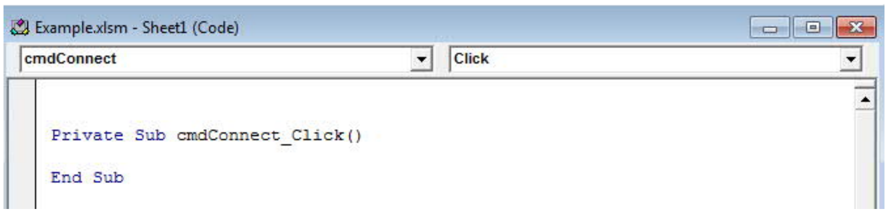
Properties

cmdConnect CommandButton

Alphabetic | Categorized

(Name)	cmdConnect
Accelerator	
AutoLoad	False
AutoSize	False
BackColor	&H8000000F&
BackStyle	1 - fmBackStyleOpaque
Caption	Connect
Enabled	True
Font	Calibri
ForeColor	&H80000012&
Height	35.25

Now double-click the Connect button and Excel will open the VBA editor and display the click event procedure for this button automatically…



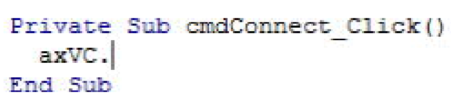
```
Private Sub cmdConnect_Click()

End Sub
```

This is where we will add some code to establish our communication connection with the controller. If you placed a Mint control instance on the sheet then type the name of the ActiveX control followed by a dot (.). If a reference was used instead then we need to include an additional step to instantiate the Mintcontroller object….if you used the reference method simply add the line…

Initialise

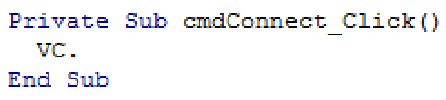
…or whatever you called your subroutine that was added earlier to the VB code module. You can now also type the name of the object followed by a dot. The screenshots below show how the click event would look in these two cases…

ActiveX object on form:



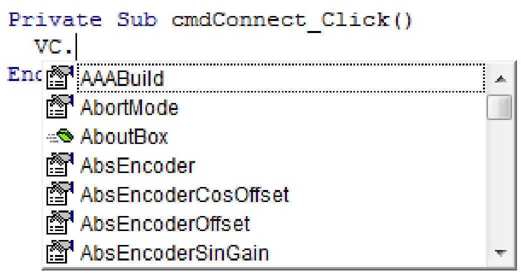
```
Private Sub cmdConnect_Click()
  axVC.
End Sub
```

Reference to Mintcontroller:



```
Private Sub cmdConnect_Click()
  VC.
End Sub
```

As soon as the dot is entered you should see that VBA displays a list of all of the available ActiveX methods and properties that can be coded (a lot of these correspond to equivalent Mint keywords, some may be prefixed with set, get or do).



```
Private Sub cmdConnect_Click()
  VC.
End
```

Scroll down through the list until the appropriate connection method is highlighted (the table below details which method should be selected for each Mint controller)…

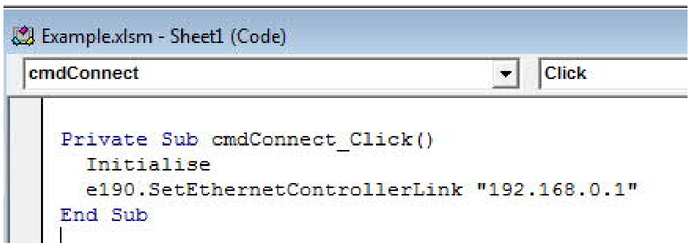
Controller	**Connection Type**	**ActiveX Method**	**Note**
NextMove ESB-2	Serial (RS232/RS422)	SetSerialControllerLink	Use platform type 36
NextMove ESB-2	USB	SetUSBControllerLink	
NextMove e100	Serial (RS232/RS422)	SetSerialControllerLink	Use platform type 31
NextMove e100	USB	SetUSBControllerLink	
NextMove e100	Ethernet	SetEthernetControllerLink	
MicroFlex e190	Ethernet	SetEthernetControllerLink	
MotiFlex e180	Ethernet	SetEthernetControllerLink	
Virtual Controller	Internal	SetVirtualControllerLink	

As we're using a Virtual Controller in this example we'll select SetVirtualControllerLink from the list and then press the SPACEBAR on our keyboard (we will use the reference in our example, if you have used an ActiveX object on the sheet then simply replace all instances of 'VC' with 'axVC', or whatever you named your ActiveX control).

VBA displays a list of any parameters that are needed to complete the ActiveX method…

```
Private Sub cmdConnect_Click()
  Initialise
  VC.SetVirtualControllerLink
End Sub
```

In the case of the Virtual Controller no additional parameters are needed. Just for reference, if we were using a MicroFlex e190 servo drive we would enter the drive's IP address (our drive is using the default address of 192.168.0.1 below).


Example.xlsm - Sheet1 (Code)

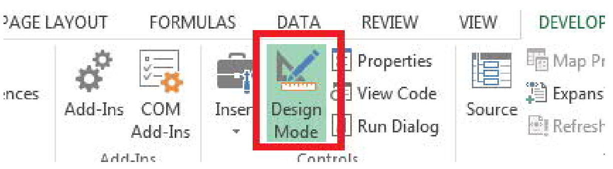
cmdConnect | Click

```
Private Sub cmdConnect_Click()
  Initialise
  e190.SetEthernetControllerLink "192.168.0.1"
End Sub
```

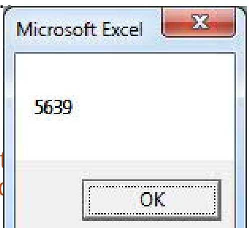
Setting the appropriate link is the only command needed to setup a connection to the Mint controller but it would be good to know if the connection has been successful or not. One easy way to do this is to attempt to read something from the controller. We could read the firmware version loaded for example by calling the AAABuild method and passing the result of this to the VBA 'Msgbox' function. The code to do this will look like this….

```
Private Sub cmdConnect_Click()
  Initialise
  VC.SetVirtualControllerLink
  MsgBox VC.AAABuild
End Sub
```

If we now switch back to our Excel sheet view and exit design mode by deselecting the 'Design Mode' button….

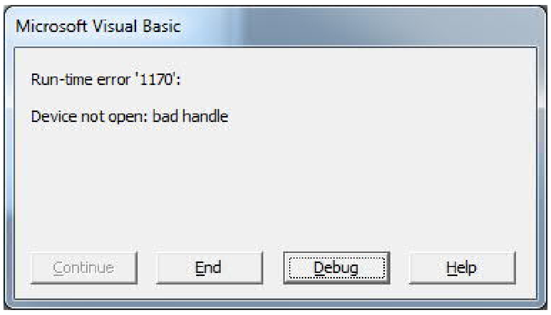


…we can now click on our Connect button to see if a successful connection is made or not. If all is well we should see a dialog like this appear…


Microsoft Excel

5639

If the connection fails a run-time error dialog will appear like this…


Microsoft Visual Basic

Run-time error '1170':

Device not open: bad handle

In this case click the ‘End’ button and check your code and check the connection details for the controller you’re using (e.g. have you used the correct IP, USB or serial address…is Workbench still running and connected to the Virtual Controller if you’re using this). If you make any changes repeat the test until the controller firmware version is reported.

Writing Excel data to the controller

We now need to read data from some cells in our Excel spreadsheet.

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:

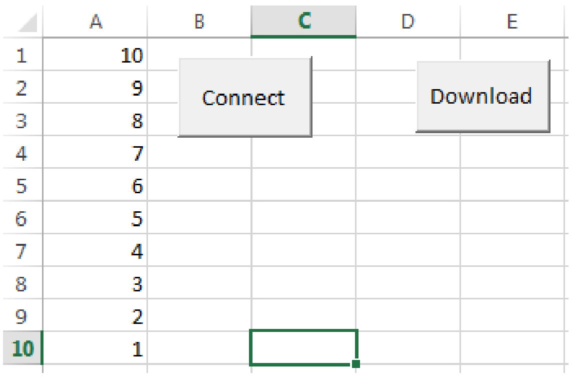
```
Private Sub cmdDownload_Click()
  Dim XLData(1 To 10) As Long
  Dim idx As Integer

  For idx = 1 To 10
    XLData(idx) = Sheet1.Cells(idx, 1)
  Next idx

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

End Sub
```

Note that we have declared XLData as an array of LONG data type to suit our 32 bit integer array in Mint. If the Mint program was using a FLOAT array then we would declare the Excel array as type SINGLE.

This code performs a number of functions:

- Declares a local 10 element array and reads the contents of the first ten rows in column 1 (A) of our spreadsheet into this array
- Uses the VariableData property of the Mint ActiveX control to pass our local array to the array defined in our Mint program – note that this property requires the name of the Mint task in which the target array or variable is declared (in this case the ParentTask) and the name of the Mint array or variable (in this case ExcelData)

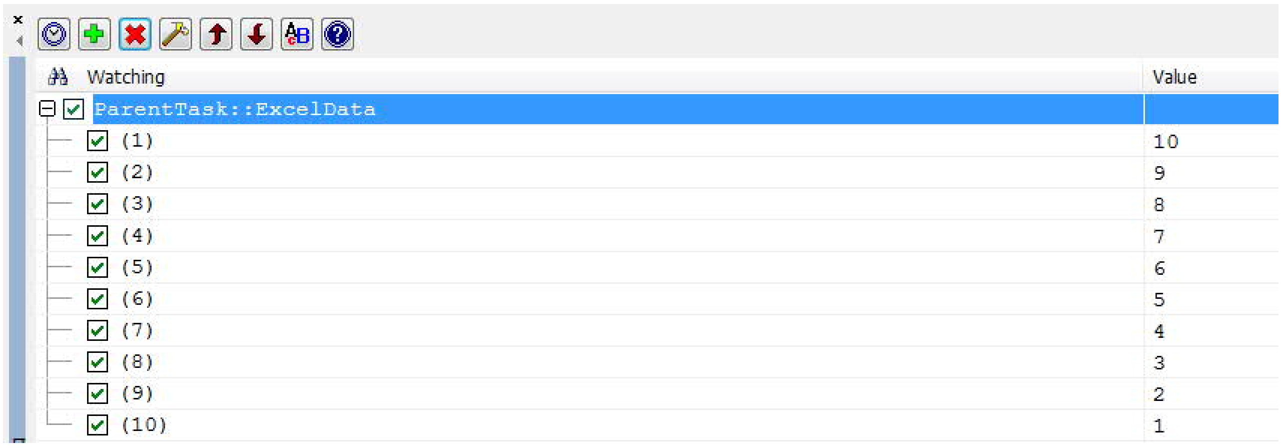
We now need to enter the spreadsheet data so switch back to the Excel sheet view and enter 10 integer values into the first 10 rows of column A. We chose to enter the numbers 10 to 1 in descending order:



	A	B	C	D	E
1	10				
2	9				
3	8				
4	7				
5	6				
6	5				
7	4				
8	3				
9	2				
10	1				

Exit Design Mode if necessary and now click the Connect button followed by 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:



Watching	Value
ParentTask::ExcelData	
(1)	10
(2)	9
(3)	8
(4)	7
(5)	6
(6)	5
(7)	4
(8)	3
(9)	2
(10)	1

Reading data from the controller

We now need to read the counter from the Mint program into a cell in our Excel spreadsheet. Repeat the previous steps to add a button to the spreadsheet and this time call it cmdUpload and caption it with “Upload”.
The code for the upload button should look like this (to display the Counter at Cell H1)…

```
Private Sub cmdUpload_Click()
    Sheet1.Cells(1, 8) = VC.VariableData("ParentTask", "Counter")
End Sub
```

Every time you click the Upload button the sheet will update to show the current value of the Counter variable.

Contact us

For more information please contact your local ABB representative or one of the following:

new.abb.com/motion
new.abb.com/drives
new.abb.com/drivespartners
new.abb.com/PLC