



WellMateOCX Software Component

ActiveX Automation component for WellMate
(EPROM version 3.5 or higher)

Software version 1.0.6
Revision dated 28 February 2005

Trademarks and Copyright

© 2004 Matrix Technologies Corp. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, in whole or in part, without the prior written permission of Matrix Technologies Corp. WellMate is a trademark and Matrix Technologies Corp. and the Matrix logo are registered trademarks of Matrix Technologies Corp. The software product is protected by copyright laws and international treaty provisions. Therefore, you must treat the software product like any other copyrighted material except that you may either (a) make one copy of the software product solely for backup or archival purposes, or (b) install the SOFTWARE PRODUCT on a single computer provided you keep the original solely for backup or archival purposes. Third-party trademarked products are trademarks or registered trademarks of their respective companies in the United States and/or other countries.

Contents

Section	Page
1 Introduction	5
1.1 Build and Distribution	5
1.2 Development Environment	5
2 Component Usage	7
2.1 Component Reference	7
2.2 Sample Application	7
2.3 Class Definitions	8
2.3.1 Class Diagram	8
2.3.2 <i>Connect Class</i>	9
2.3.3 Enumerations	15
A Appendix - Sample Application	19
A1 Sample Application Source Code	21

Figures

Figure 1 : Microsoft Visual Basic Project References dialogue example	7
Figure 2 : Sample application dialogue	7
Figure 3 : Class diagram	8
Figure 4 : Sample application dialogue screen 1 (individual command control)	19
Figure 5 : Sample application dialogue screen 2 (protocol scripts)	19
Figure 6 : Sample application dialogue screen 3 (dispense patterns)	20
Figure 7 : Sample application dialogue screen 4 (tweak table)	20

1 Introduction

The WellMate instrument from Matrix Technologies is a high speed, small footprint, 8-channel fluid dispenser for 6, 12, 24, 48, 96 and 384 well microplates. It repetitively dispenses samples and reagents into the plates, with high accuracy and efficiency. You can program the WellMate unit with dispensing protocols, dispense volumes, plate type, and more.

Features and applications of the WellMate instrument include:

- Height-adjustable dispense head that accommodates shallow- and deep-well plates and blocks
- High-resolution, stepper-motor technology that allows fast, accurate dispensing
- Low-cost, replaceable tubing cartridges
- Dynamic dispense volume range (1.0 μ L–2000 μ L), programmable in 1.0 μ L increments
- Easy programming that allows you to select individual plate columns for dispense
- Memory-storage capacity for 18 files
- Full RS-232 programming for ease of integration
- Removable plate stage that allows easy cleaning of the Teflon coated base

The WellMateOCX software component provides an application developer with an ActiveX automation interface for the WellMate device. The component will enable an application to instantiate an object which can directly control a WellMate device.

All code examples in this document are excerpts from the sample application source code.

1.1 Build and Distribution

The WellMateOCX component can be distributed as a required component within the client application. Any component dependencies will be included as part of the application build. The component cannot be distributed, on its own, without prior written consent of Matrix Technologies Corp. or its distributors.

1.2 Development Environment

The WellMateOCX component has been developed for use with the following programming and development application suites :

- Microsoft Visual Basic (versions 5 and 6)
- Microsoft Visual Studio C++ (versions 5 and 6)
- Microsoft Visual Studio .NET (VB and C#) projects (as OLE object reference)
- Borland C++ Builder
- Borland Delphi

2 Component Usage

The component usage examples in this document are shown with Microsoft Visual Basic (version 6) code excerpts from the sample application that is supplied with the component.

2.1 Component Reference

In order to instantiate and use the component in Visual Basic, the *Project References* menu option must be used to add a reference to the Visual Basic project. The component is named *WellMateOCX*.

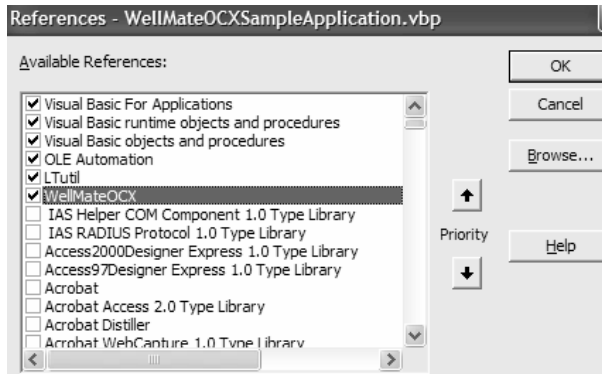


Figure 1 : Microsoft Visual Basic Project References dialogue example

2.2 Sample Application

A sample Visual Basic 6 project is included within the InstallShield installation manifest. To install the necessary file choose the *Custom* installation option and select to install the sample application and source files. The source code for this is listed in Appendix A (note, this code does not cover all Attributes and Procedures and is included here only for illustration and Visual Basic version 6 coding purposes only).

The sample application can be used as a reference or template of attribute and procedure usage. This sample application will allow a user to directly control a WellMate device by issuing individual commands or by utilising a rudimentary built in protocol scripting language.

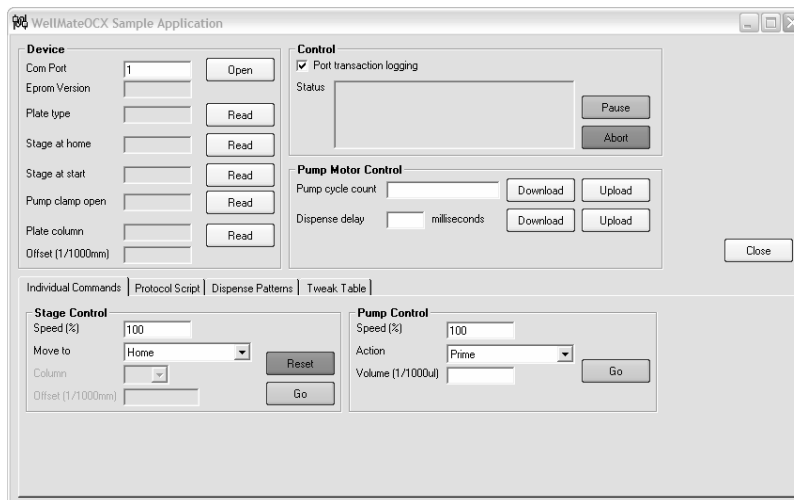


Figure 2 : Sample application dialogue

2.3 Class Definitions

2.3.1 Class Diagram

The class diagram below shows the component architecture and the available procedures and attributes for each class.

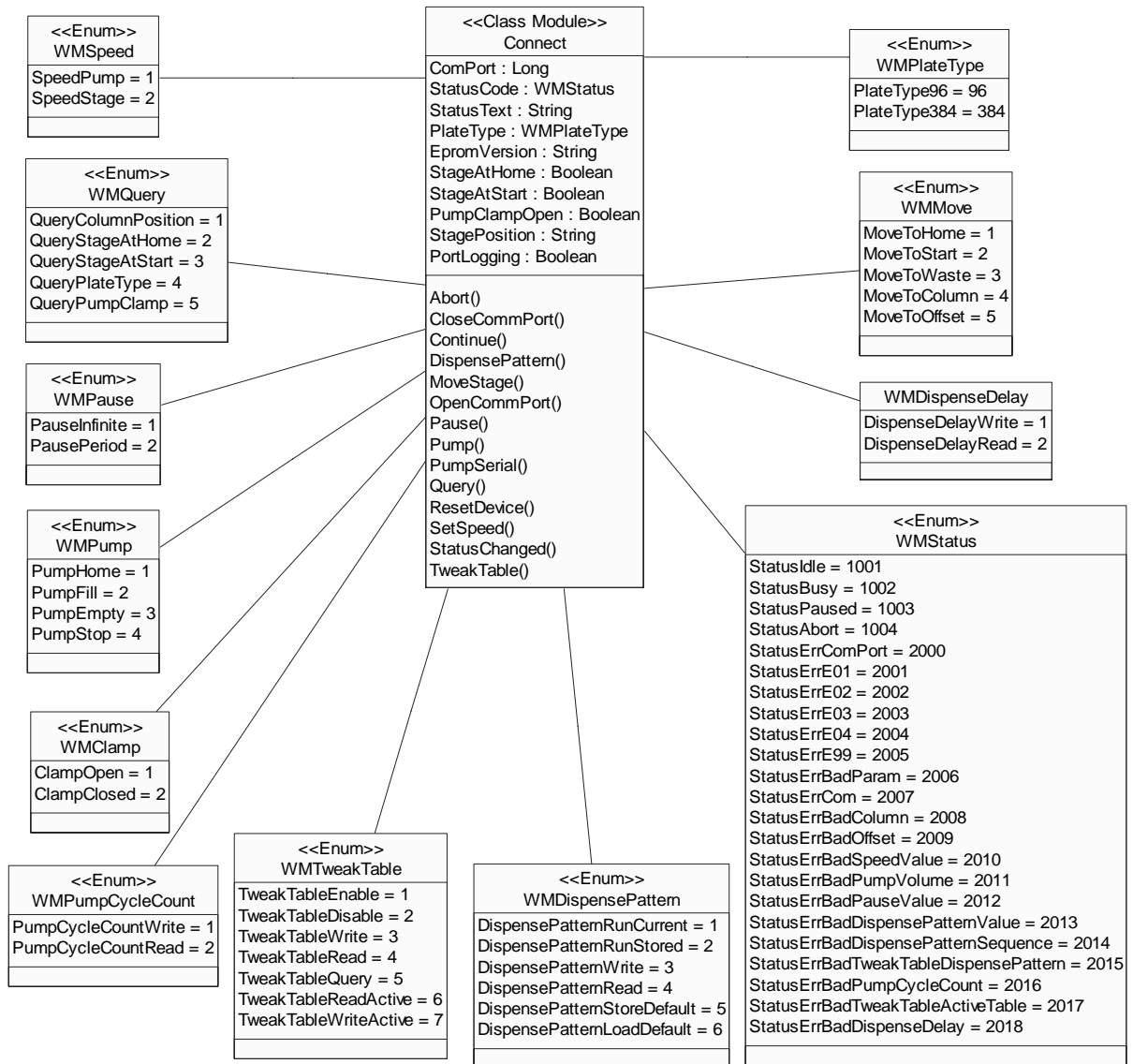


Figure 3 : Class diagram

2.3.2 Connect Class

The connector class is used to instantiate a WellMateOCX object. It is through this object that all procedures and attributes are made available.

An example procedure for instantiating a *Connect* class object is :-

```
Dim WithEvents oWellMateOCX As WellMateOCX.Connect

Private Sub Form_Load()
    'create the WellMateOCX object
    Set oWellMateOCX = New WellMateOCX.Connect
```

Declaring the variable `WithEvents` will provide a means by which event messages are passed back to the client application.

2.3.2.1 Attributes

The attributes available within the *Connect* class are mainly used to retrieve sensor information from the device. For this reason the majority of the properties are **read only**.

Example code usage :-

```
'if all was OK then show the information that is returned by the EPROM when comms are established
If bRetVal Then
    With moWellMateOCX
        lblData(conEPROMVERSION).Caption = .EpromVersion
        lblData(conSTAGEATHOME).Caption = .StageAtHome
        lblData(conSTAGEATSTART).Caption = .StageAtStart
        lblData(conCLAMPOPEN).Caption = .PumpClampOpen
        lblData(conPLATEATYPE).Caption = .PlateType
        lblData(conPLATECOLUMN).Caption = Left(.StagePosition, 2)
        lblData(conWELLOFFSET).Caption = Mid(.StagePosition, 3)
    End With
End If
```

Name	Return Type	Access	Comments						
ComPort	Long	Read	Used to query the current RS232 communication port that has been selected						
EpromVersion	String	Read	The current EPROM version of the connected device. This parameter will only have a value once the <code>OpenCommPort()</code> procedure has been called.						
PlateType	WMPlateType	Read	Enumerated value represented the current WellMate plate type setting (96 or 384)						
PortLogging	None	Write	This attribute is used to switch on or off the RS232 communication port transaction log which will record, if switched on, all communication packets between the PC and the device and store them in the <code>PortLog.txt</code> file located in the WellMateOCX.dll application directory.						
PumpClampOpen	Boolean	Read	A 'true' or 'false' value that is returned by the device relating to the pump clamp sensor						
StageAtHome	Boolean	Read	A 'true' or 'false' value that is returned by the device relating to the stage location based on the 'home' position sensor						
StageAtStart	Boolean	Read	A 'true' or 'false' value that is returned by the device relating to the stage location based on the 'dispensing' position sensor						
StagePosition	String	Read	This attribute is used to identify the current stage position by plate well column and offset within the well. The value is represented as :- <table border="1" data-bbox="802 1551 1468 1640"> <tr> <td>cc</td> <td>Column number (01 is left most)</td> </tr> <tr> <td>±</td> <td>Positive or negative offset value</td> </tr> <tr> <td>oooo</td> <td>Offset from centre of well in 1/1000mm</td> </tr> </table>	cc	Column number (01 is left most)	±	Positive or negative offset value	oooo	Offset from centre of well in 1/1000mm
cc	Column number (01 is left most)								
±	Positive or negative offset value								
oooo	Offset from centre of well in 1/1000mm								
StatusCode	WMStatus	Read	Enumerated value representing the current component 'status'. This could represent an error condition or device status (e.g. idle)						
StatusText	String	Read	A textual dialogue value representing the current status						

2.3.2.2 Procedures

The procedures within the *Connect* class deal mainly with the direct issuing of commands to the device to control stage movement or pump volume displacement.

Example code usage :-

```
'now do the move action
Select Case cboPosition.ListIndex
  Case conMOVEHOME
    bRetVal = moWellMateOCX.MoveStage(MoveToHome)
  Case conMOVESTART
    bRetVal = moWellMateOCX.MoveStage(MoveToStart)
  Case conMOVEWASTE
    bRetVal = moWellMateOCX.MoveStage(MoveToWaste)
  Case conMOVECOLUMN
    bRetVal = moWellMateOCX.MoveStage(MoveToColumn, cboColumn)
  Case conMOVEOFFSET
    bRetVal = moWellMateOCX.MoveStage(MoveToOffset, cboColumn, Val(txtOffset))
End Select
```

Name	Return	Comments									
Abort	Boolean	<p>Issues an immediate instruction to the device to stop after the current command has completed. The procedure will return a value of 'true' if the command was successfully processed or 'false' if the command failed. In the latter scenario a <i>StatusChanged</i> event will also have been triggered. Reading the <i>StatusCode</i> and <i>StatusText</i> attributes will provide information relating to the failure reason.</p> <p>Abort()</p>									
CloseCommPort	Nothing	<p>Closes the RS232 serial communication port that was opened with the <i>OpenCommPort</i> command</p> <p>CloseCommPort()</p>									
Continue	Boolean	<p>Issues an immediate instruction to the device to continue after an error or a pause. The procedure will return a value of 'true' if the command was successfully processed or 'false' if the command failed. In the latter scenario a <i>StatusChanged</i> event will also have been triggered. Reading the <i>StatusCode</i> and <i>StatusText</i> attributes will provide information relating to the failure reason.</p> <p>Continue()</p>									
DispensePattern	Boolean	<p>Issues an immediate instruction to the device relating to dispensing by a stored pattern. A dispense pattern identifies, by plate type and volume, which columns within a plate into which liquid is to be dispensed. The pattern itself is a simple binary representation text string where '1' signifies dispense and '0' signifies no dispense for each column position. The command is multi faceted and allows dispense patterns to be downloaded to the device or uploaded to the client application.</p> <p>Further actions are also available within this function to allow for a pattern to be run, either directly from it's memory location or from the special 'default' location. The default location is the one used when the 'Start' button is pressed on the device front panel.</p> <table border="1"> <thead> <tr> <th>Parameter</th> <th>Type</th> <th>Comment</th> </tr> </thead> <tbody> <tr> <td>IAction</td> <td>WMDispensePattern</td> <td>Enumerated value used to specify the action required. This provides functionality to upload or download a pattern, to copy patterns to different memory locations on the device and to run a stored or default pattern</td> </tr> <tr> <td>IDispensePattern</td> <td>Long</td> <td>Used to specify the actual pattern number when reading or writing, copying and running. The value of this field is specific to plate type, for 96 well plates the pattern can be in the range of 0 to 9 and for 384 well plates in the range of 10 to 19.</td> </tr> </tbody> </table>	Parameter	Type	Comment	IAction	WMDispensePattern	Enumerated value used to specify the action required. This provides functionality to upload or download a pattern, to copy patterns to different memory locations on the device and to run a stored or default pattern	IDispensePattern	Long	Used to specify the actual pattern number when reading or writing, copying and running. The value of this field is specific to plate type, for 96 well plates the pattern can be in the range of 0 to 9 and for 384 well plates in the range of 10 to 19.
Parameter	Type	Comment									
IAction	WMDispensePattern	Enumerated value used to specify the action required. This provides functionality to upload or download a pattern, to copy patterns to different memory locations on the device and to run a stored or default pattern									
IDispensePattern	Long	Used to specify the actual pattern number when reading or writing, copying and running. The value of this field is specific to plate type, for 96 well plates the pattern can be in the range of 0 to 9 and for 384 well plates in the range of 10 to 19.									

Name	Return	Comments												
DispensePattern	Boolean	<table border="1"> <tr> <td>bSetAsDefault</td> <td>Boolean</td> <td>When used in conjunction with a IAction value of DispensePatternRunStored this parameter when set to True will cause the pattern to be copied into the default memory location and then run.</td> </tr> <tr> <td>IVolume</td> <td>Long</td> <td>When used with a IAction value of DispensePatternWrite this parameter is used to specify the volume to be dispensed when the pattern is selected for running.</td> </tr> <tr> <td>sDispensePattern</td> <td>String</td> <td>When used with a IAction value of DispensePatternWrite this parameter contains the binary representation of the dispense pattern where '1' signifies that a column is to be dispensed into and '0' represents no dispense, e.g. '110011111111'. When used with a IAction value of DispensePatternRead this parameter is used as a pointer which will retrieve the value of the current pattern string held on the device.</td> </tr> </table>	bSetAsDefault	Boolean	When used in conjunction with a IAction value of DispensePatternRunStored this parameter when set to True will cause the pattern to be copied into the default memory location and then run.	IVolume	Long	When used with a IAction value of DispensePatternWrite this parameter is used to specify the volume to be dispensed when the pattern is selected for running.	sDispensePattern	String	When used with a IAction value of DispensePatternWrite this parameter contains the binary representation of the dispense pattern where '1' signifies that a column is to be dispensed into and '0' represents no dispense, e.g. '110011111111'. When used with a IAction value of DispensePatternRead this parameter is used as a pointer which will retrieve the value of the current pattern string held on the device.			
		bSetAsDefault	Boolean	When used in conjunction with a IAction value of DispensePatternRunStored this parameter when set to True will cause the pattern to be copied into the default memory location and then run.										
		IVolume	Long	When used with a IAction value of DispensePatternWrite this parameter is used to specify the volume to be dispensed when the pattern is selected for running.										
sDispensePattern	String	When used with a IAction value of DispensePatternWrite this parameter contains the binary representation of the dispense pattern where '1' signifies that a column is to be dispensed into and '0' represents no dispense, e.g. '110011111111'. When used with a IAction value of DispensePatternRead this parameter is used as a pointer which will retrieve the value of the current pattern string held on the device.												
<pre>DispensePattern(DispensePatternRunCurrent) DispensePattern(DispensePatternRunStored, p, b) where p = pattern number and b = save as default flag (true or false) DispensePattern(DispensePatternWrite, p, , v, s) where p = pattern number, v = volume and s = pattern binary mask as text DispensePattern(DispensePatternRead, p, , , *s) where p = pattern number and s = pointer to hold pattern mask string DispensePattern(DispensePatternStoreDefault, p) where p = pattern number DispensePattern(DispensePatternLoadDefault, p) where p = pattern number</pre>														
<pre>DispensePattern(DispensePatternRunCurrent) DispensePattern(DispensePatternRunStored, p, b) where p = pattern number and b = save as default flag (true or false) DispensePattern(DispensePatternWrite, p, , v, s) where p = pattern number, v = volume and s = pattern binary mask as text DispensePattern(DispensePatternRead, p, , , *s) where p = pattern number and s = pointer to hold pattern mask string DispensePattern(DispensePatternStoreDefault, p) where p = pattern number DispensePattern(DispensePatternLoadDefault, p) where p = pattern number</pre>														
MoveStage	Boolean	<p>Issues an immediate instruction to the device to move the stage to the specified location. The command has a number of optional parameters specific to the move action. The procedure will return a value of 'true' if the command was successfully processed or 'false' if the command failed. In the latter scenario an <i>StatusChanged</i> event will also have been triggered. Reading the <i>StatusCode</i> and <i>StatusText</i> attributes will provide information relating to the failure reason.</p> <table border="1"> <thead> <tr> <th>Parameter</th> <th>Type</th> <th>Comment</th> </tr> </thead> <tbody> <tr> <td>IAction</td> <td>WMMove</td> <td>Enumerated value used to specify the action required.</td> </tr> <tr> <td>IColumn</td> <td>Long</td> <td>An optional parameter, used only with an IAction of MoveToColumn or MoveToOffset, which is used to specify the column within the plate. The value of this parameter is from 1 to the maximum allowed for the current plate settings (12 or 24)</td> </tr> <tr> <td>IOffset</td> <td>Long</td> <td>An optional parameter, used only with an IAction of MoveToOffset, which is used to specify the offset from the centre of the well either positive or negative in 1/1000mm units, for the specified column</td> </tr> </tbody> </table> <pre>MoveStage(MoveToHome) MoveStage(MoveToStart) MoveStage(MoveToReservoir) MoveStage(MoveToColumn, c) where c = column MoveStage(MoveToOffset, c, ±o) where c = column and o = offset (±)</pre>	Parameter	Type	Comment	IAction	WMMove	Enumerated value used to specify the action required.	IColumn	Long	An optional parameter, used only with an IAction of MoveToColumn or MoveToOffset, which is used to specify the column within the plate. The value of this parameter is from 1 to the maximum allowed for the current plate settings (12 or 24)	IOffset	Long	An optional parameter, used only with an IAction of MoveToOffset, which is used to specify the offset from the centre of the well either positive or negative in 1/1000mm units, for the specified column
Parameter	Type	Comment												
IAction	WMMove	Enumerated value used to specify the action required.												
IColumn	Long	An optional parameter, used only with an IAction of MoveToColumn or MoveToOffset, which is used to specify the column within the plate. The value of this parameter is from 1 to the maximum allowed for the current plate settings (12 or 24)												
IOffset	Long	An optional parameter, used only with an IAction of MoveToOffset, which is used to specify the offset from the centre of the well either positive or negative in 1/1000mm units, for the specified column												
OpenCommPort	Boolean	<p>This is used to open the RS232 serial communication port and establish communication with the device. The EPROM will respond, if successful, with the current EPROM version and the current sensor values. The procedure will return a value of 'true' if the command was successfully processed or 'false' if the command failed. In the latter scenario a <i>StatusChanged</i> event will also have been triggered. Reading the <i>StatusCode</i> and <i>StatusText</i> attributes will provide information relating to the failure reason.</p>												

Name	Return	Comments									
		<table border="1"> <thead> <tr> <th>Parameter</th> <th>Type</th> <th>Comment</th> </tr> </thead> <tbody> <tr> <td>IComPort</td> <td>Long</td> <td>The required RS232 serial communication port</td> </tr> </tbody> </table> <p>OpenCommPort(1)</p>	Parameter	Type	Comment	IComPort	Long	The required RS232 serial communication port			
Parameter	Type	Comment									
IComPort	Long	The required RS232 serial communication port									
Pause	Boolean	<p>Issues an immediate instruction to the device to pause after the current command has completed. The command has a number of optional parameters relevant to the pause required. The procedure will return a value of 'true' if the command was successfully processed or 'false' if the command failed. In the latter scenario a <i>StatusChanged</i> event will also have been triggered. Reading the <i>StatusCode</i> and <i>StatusText</i> attributes will provide information relating to the failure reason.</p> <table border="1"> <thead> <tr> <th>Parameter</th> <th>Type</th> <th>Comment</th> </tr> </thead> <tbody> <tr> <td>IAction</td> <td>WMPause</td> <td>Enumerated value used to specify the action required.</td> </tr> <tr> <td>IPeriod</td> <td>Long</td> <td>An optional parameter, used only with an IAction of PausePeriod, which is used to pause the device for a number of milliseconds</td> </tr> </tbody> </table> <p>Pause(PauseInfinite) - pause forever Pause(PausePeriod, p) - pause for a number milliseconds, specified by p</p>	Parameter	Type	Comment	IAction	WMPause	Enumerated value used to specify the action required.	IPeriod	Long	An optional parameter, used only with an IAction of PausePeriod, which is used to pause the device for a number of milliseconds
Parameter	Type	Comment									
IAction	WMPause	Enumerated value used to specify the action required.									
IPeriod	Long	An optional parameter, used only with an IAction of PausePeriod, which is used to pause the device for a number of milliseconds									
Pump	Boolean	<p>Issues an immediate instruction to the device to control the pump motor and valves. The command has a number of optional parameters specific to the different types of pump actions. The procedure will return a value of 'true' if the command was successfully processed or 'false' if the command failed. In the latter scenario an <i>StatusChanged</i> event will also have been triggered. Reading the <i>StatusCode</i> and <i>StatusText</i> attributes will provide information relating to the failure reason.</p> <table border="1"> <thead> <tr> <th>Parameter</th> <th>Type</th> <th>Comment</th> </tr> </thead> <tbody> <tr> <td>IAction</td> <td>WMPump</td> <td>Enumerated value used to specify the action required.</td> </tr> <tr> <td>IVolume</td> <td>Long</td> <td>An optional parameter, used only with an IAction of PumpFill or PumpEmpty, which is used to specify the volume of liquid to be moved (filled or emptied)</td> </tr> </tbody> </table> <p>Pump(PumpHome) - used to prime the pump Pump(PumpFill) - used to fill indefinitely Pump(PumpEmpty) - used to empty indefinitely (i.e. to remove liquid from the tubing) Pump(PumpFill, v) - dispense, v = volume in nanolitres (1/1000ul) Pump(PumpEmpty, v) - empty discrete liquid from tubing, v = volume in nanolitres</p>	Parameter	Type	Comment	IAction	WMPump	Enumerated value used to specify the action required.	IVolume	Long	An optional parameter, used only with an IAction of PumpFill or PumpEmpty, which is used to specify the volume of liquid to be moved (filled or emptied)
Parameter	Type	Comment									
IAction	WMPump	Enumerated value used to specify the action required.									
IVolume	Long	An optional parameter, used only with an IAction of PumpFill or PumpEmpty, which is used to specify the volume of liquid to be moved (filled or emptied)									
PumpCycleCount	Boolean	<p>Used to set or retrieve the pump motor cycle count.</p> <table border="1"> <thead> <tr> <th>Parameter</th> <th>Type</th> <th>Comment</th> </tr> </thead> <tbody> <tr> <td>IAction</td> <td>WMPumpCycleCount</td> <td>Enumerated value used to specify the action required. This provides functionality to either read or reset the count on the device.</td> </tr> <tr> <td>ICount</td> <td>Long</td> <td>When used with a IAction value of PumpCycleCountWrite parameter is used to set the serial number. When used with a IAction value of PumpCycleCountRead this parameter is a pointer which will contain the current setting of the pump motor step serial number on the device.</td> </tr> </tbody> </table> <p>PumpCycleCount(PumpCycleCountWrite, l) where l = pump motor cycle count value to write PumpCycleCount(PumpCycleCountRead, *l) where l = pointer for current value from device</p>	Parameter	Type	Comment	IAction	WMPumpCycleCount	Enumerated value used to specify the action required. This provides functionality to either read or reset the count on the device.	ICount	Long	When used with a IAction value of PumpCycleCountWrite parameter is used to set the serial number. When used with a IAction value of PumpCycleCountRead this parameter is a pointer which will contain the current setting of the pump motor step serial number on the device.
Parameter	Type	Comment									
IAction	WMPumpCycleCount	Enumerated value used to specify the action required. This provides functionality to either read or reset the count on the device.									
ICount	Long	When used with a IAction value of PumpCycleCountWrite parameter is used to set the serial number. When used with a IAction value of PumpCycleCountRead this parameter is a pointer which will contain the current setting of the pump motor step serial number on the device.									
Query	Boolean	<p>Issues an immediate instruction to the device to retrieve and return the value of a specified sensor. The command has two parameters, the first specifies the sensor by an enumerated name, the second is a pointer which is passed to the function and is populated with the current sensor value upon a successful query completion. The procedure will return a value of 'true' if the command was successfully processed or 'false' if the command failed. In the latter scenario an <i>StatusChanged</i> event will also</p>									

Name	Return	Comments												
		<p>have been triggered. Reading the <i>StatusCode</i> and <i>StatusText</i> attributes will provide information relating to the failure reason.</p> <table border="1"> <thead> <tr> <th>Parameter</th> <th>Type</th> <th>Comment</th> </tr> </thead> <tbody> <tr> <td>IAction</td> <td>WMQuery</td> <td>Enumerated value used to specify the action required.</td> </tr> <tr> <td>vData</td> <td>Variant</td> <td> <p>This parameter is passed as a pointer and is populated with the sensor value upon successful completion. The data type changes according to the query requested.</p> <p>QueryPlateType returns a WMPlateType value QueryColumnPosition returns a string QueryStageAtHome returns a Boolean QueryStageAtStart returns a Boolean QueryPumpClamp returns a WMClamp value</p> </td> </tr> </tbody> </table> <p> Query(QueryColumnPosition, *p) Query(QueryStageAtHome, *p) Query(QueryStageAtStart, *p) Query(QueryPumpClamp, *p) Query(QueryPlateType, *p) </p>	Parameter	Type	Comment	IAction	WMQuery	Enumerated value used to specify the action required.	vData	Variant	<p>This parameter is passed as a pointer and is populated with the sensor value upon successful completion. The data type changes according to the query requested.</p> <p>QueryPlateType returns a WMPlateType value QueryColumnPosition returns a string QueryStageAtHome returns a Boolean QueryStageAtStart returns a Boolean QueryPumpClamp returns a WMClamp value</p>			
Parameter	Type	Comment												
IAction	WMQuery	Enumerated value used to specify the action required.												
vData	Variant	<p>This parameter is passed as a pointer and is populated with the sensor value upon successful completion. The data type changes according to the query requested.</p> <p>QueryPlateType returns a WMPlateType value QueryColumnPosition returns a string QueryStageAtHome returns a Boolean QueryStageAtStart returns a Boolean QueryPumpClamp returns a WMClamp value</p>												
ResetDevice	Boolean	<p>Issues an immediate instruction to the device to reset all motors to their home positions. The procedure will return a value of 'true' if the command was successfully processed or 'false' if the command failed. In the latter scenario an <i>StatusChanged</i> event will also have been triggered. Reading the <i>StatusCode</i> and <i>StatusText</i> attributes will provide information relating to the failure reason.</p> <p>ResetDevice()</p>												
SetSpeed	Boolean	<p>Issues an immediate instruction to the device to set the stage speed of travel that is used in all subsequent <i>MoveStage</i> commands. The procedure will return a value of 'true' if the command was successfully processed or 'false' if the command failed. In the latter scenario a <i>StatusChanged</i> event will also have been triggered. Reading the <i>StatusCode</i> and <i>StatusText</i> attributes will provide information relating to the failure reason.</p> <table border="1"> <thead> <tr> <th>Parameter</th> <th>Type</th> <th>Comment</th> </tr> </thead> <tbody> <tr> <td>IAction</td> <td>WMSpeed</td> <td>Enumerated value used to specify the action required, either to set the speed of the stage motor or the pump motor.</td> </tr> <tr> <td>IValue</td> <td>Long</td> <td>A mandatory parameter used to specify the speed value required. This is represented as a percentage from 1 to 100.</td> </tr> </tbody> </table> <p> SetSpeed(SpeedStage, v) where v = speed value SetSpeed(SpeedPump, v) </p>	Parameter	Type	Comment	IAction	WMSpeed	Enumerated value used to specify the action required, either to set the speed of the stage motor or the pump motor.	IValue	Long	A mandatory parameter used to specify the speed value required. This is represented as a percentage from 1 to 100.			
Parameter	Type	Comment												
IAction	WMSpeed	Enumerated value used to specify the action required, either to set the speed of the stage motor or the pump motor.												
IValue	Long	A mandatory parameter used to specify the speed value required. This is represented as a percentage from 1 to 100.												
TweakTable	Boolean	<p>Used to control the table of entries on the device which is used to trim the pump motor when dispensing by pattern columns for lower volumes to ensure volume accuracy. The command is multi faceted and allows the enabling or disabling of the functionality as well as for reading and writing a table.</p> <table border="1"> <thead> <tr> <th>Parameter</th> <th>Type</th> <th>Comment</th> </tr> </thead> <tbody> <tr> <td>IAction</td> <td>WMTweakTable</td> <td>Enumerated value used to specify the action required.</td> </tr> <tr> <td>IVolume</td> <td>Long</td> <td>When used with an IAction of TweakTableRead and TweakTableWrite this is used to specify the volume range relating to the tweak table entries.</td> </tr> <tr> <td>sTweakTable Dispense Pattern</td> <td>String</td> <td> <p>Contains the tweak table pattern which consists of concatenated values representing the volume tweaks per column in the format :-</p> <p>Sign + zero padded volume value,</p> <p>e.g. '+012-031+022+016-014'</p> <p>When used with a IAction value of</p> </td> </tr> </tbody> </table>	Parameter	Type	Comment	IAction	WMTweakTable	Enumerated value used to specify the action required.	IVolume	Long	When used with an IAction of TweakTableRead and TweakTableWrite this is used to specify the volume range relating to the tweak table entries.	sTweakTable Dispense Pattern	String	<p>Contains the tweak table pattern which consists of concatenated values representing the volume tweaks per column in the format :-</p> <p>Sign + zero padded volume value,</p> <p>e.g. '+012-031+022+016-014'</p> <p>When used with a IAction value of</p>
Parameter	Type	Comment												
IAction	WMTweakTable	Enumerated value used to specify the action required.												
IVolume	Long	When used with an IAction of TweakTableRead and TweakTableWrite this is used to specify the volume range relating to the tweak table entries.												
sTweakTable Dispense Pattern	String	<p>Contains the tweak table pattern which consists of concatenated values representing the volume tweaks per column in the format :-</p> <p>Sign + zero padded volume value,</p> <p>e.g. '+012-031+022+016-014'</p> <p>When used with a IAction value of</p>												

Name	Return	Comments									
		<p>TweakTableWrite this parameter is used to write a new table entry.</p> <p>When used with a IAction value of TweakTableRead this parameter is pointer based and will be populated with the value contained on the device.</p>									
		<p>IActiveTable Long</p> <p>Used to set or return the current value used to determine which tweak table to use during dispensing.</p> <p>Used with an action of TweakTableWriteActive to set the current table and TweakTableReadActive to retrieve the current setting.</p> <p>A value of zero indicates factory default, a value of '1' (one) indicates user defined table.</p>									
		<p>bStatus Boolean</p> <p>Used to return the current usage status.</p> <p>Used with an action of TweakTableQuery.</p> <p>This is set to True if a tweak table will be used during dispensing.</p>									
		<pre>TweakTable(TweakTableEnable) TweakTable(TweakTableDisable) TweakTable(TweakTableWrite, v, s) where v = volume and s = tweak table dataset text string TweakTable(TweakTableRead, , *s) where s = pointer to tweak table dataset values from device TweakTable(TweakTableQuery, , , , *p) TweakTable(TweakTableWriteActive, , , a) Where a = tweak table to use (0 or 1) TweakTable(TweakTableReadActive, , , *p)</pre>									
DispenseDelay	Boolean	<p>Used to set a pause, in milliseconds, that is performed after each column dispense.</p> <table border="1"> <thead> <tr> <th>Parameter</th> <th>Type</th> <th>Comment</th> </tr> </thead> <tbody> <tr> <td>IAction</td> <td>WMDispenseDelay</td> <td>Enumerated value used to specify the action required.</td> </tr> <tr> <td>ITime</td> <td>Long</td> <td>Used to set or return the current delay value to use. The value represents milliseconds within a range from 0 to 99990. When used with DispenseDelayWrite the value is used to set the delay. When used with DispenseDelayRead, the current value stored on the device is retrieved.</td> </tr> </tbody> </table> <pre>DispenseDelay(DispenseDelayWrite, d) Where d = delay time in milliseconds DispenseDelay(DispenseDelayRead, *p)</pre>	Parameter	Type	Comment	IAction	WMDispenseDelay	Enumerated value used to specify the action required.	ITime	Long	Used to set or return the current delay value to use. The value represents milliseconds within a range from 0 to 99990. When used with DispenseDelayWrite the value is used to set the delay. When used with DispenseDelayRead, the current value stored on the device is retrieved.
Parameter	Type	Comment									
IAction	WMDispenseDelay	Enumerated value used to specify the action required.									
ITime	Long	Used to set or return the current delay value to use. The value represents milliseconds within a range from 0 to 99990. When used with DispenseDelayWrite the value is used to set the delay. When used with DispenseDelayRead, the current value stored on the device is retrieved.									

2.3.2.3 Events

Name	Comments									
StatusChanged	<p>Event is triggered whenever the StatusCode attribute changes.</p> <table border="1"> <thead> <tr> <th>Parameter</th> <th>Type</th> <th>Comment</th> </tr> </thead> <tbody> <tr> <td>IAction</td> <td>WMDispenseDelay</td> <td>Enumerated value representing the current status value</td> </tr> <tr> <td>sStatusText</td> <td>String</td> <td>An expanded textual representation of the status code</td> </tr> </tbody> </table> <pre>Private Sub moWellMateOCX_StatusChanged(ByVal lStatusCode As WellMateOCX.WMStatus, _ ByVal sStatusText As String)</pre>	Parameter	Type	Comment	IAction	WMDispenseDelay	Enumerated value representing the current status value	sStatusText	String	An expanded textual representation of the status code
Parameter	Type	Comment								
IAction	WMDispenseDelay	Enumerated value representing the current status value								
sStatusText	String	An expanded textual representation of the status code								

2.3.3 Enumerations

A number of attributes and procedures use numeric values to represent required actions or to decode device sensor values. These values have been grouped together as enumerations to provide a standard list of meaningful codes.

2.3.3.1 WMSpeed

This list of values is used in conjunction with the SetSpeed procedure.

Enumeration	Value	Comment
SpeedPump	1	Used to represent a speed value relating to the pump motor
SpeedStage	2	Relates to the stage motor speed

2.3.3.2 WMPause

This list of values is used in conjunction with the Pause procedure.

Enumeration	Value	Comment
PauseInfinite	1	Pauses forever until a continue or CloseCommPort is called
PausePeriod	2	Pauses (waits) for a number of milliseconds

2.3.3.3 WMQuery

This list of values is used in conjunction with the Query procedure.

Enumeration	Value	Comment
QueryColumnPosition	1	Queries the current column position
QueryStageAtHome	2	Queries whether the home sensor is tripped
QueryStageAtStart	3	Queries if the stage is positioned for dispensing
QueryPlateType	4	Queries the device plate type setting
QueryPumpClamp	5	Queries the tubing clamp jaw sensor

2.3.3.4 WMClamp

This list of values is used in conjunction with the Query procedure when issuing a QueryPumpClamp command.

Enumeration	Value	Comment
ClampOpen	1	The sensor reports that the clamp jaws are open
ClampClosed	2	The sensor reports that the clamp jaws are closed

2.3.3.5 WMMove

This list of values is used in conjunction with the MoveStage command.

Enumeration	Value	Comment
MoveToHome	1	Causes the stage to move to the home sensor position
MoveToStart	2	Moves the stage to the dispensing position
MoveToWaste	3	Moves the stage to the waste receptacle
MoveToColumn	4	Moves the stage to the specified column within the plate
MoveToOffset	5	Moves the stage to the specified offset within the current well

2.3.3.6 WMPlateType

This list of values is used in conjunction with the PlateType attribute and the QueryPlateType query, which both return the current sensor value for the plate type switch on the device.

Enumeration	Value	Comment
PlateType96	1	The switch is set to a 96 well plate
PlateType384	2	The switch is set to a 384 well plate

2.3.3.7 WMPump

This list of values is used in conjunction with the Pump procedure to specify the pump action required.

Enumeration	Value	Comment
PumpHome	1	This is used to prime the pump and is used to ensure that the pump is at a known start point.
PumpFill	2	This has two uses, both are used to cause the pump motor into a fill motion. This process is normally used after replacing a tube set.
PumpEmpty	3	This has two uses, both are used to cause the pump motor into an empty motion
PumpStop	4	Causes the pump motor to stop it's current activity (typically used after a PumpFill or PumpEmpty action)

2.3.3.8 WMDispensePattern

This list of values is used in conjunction with the DispensePattern procedure to specify the dispense by pattern action required.

Enumeration	Value	Comment
DispensePatternRunCurrent	1	This causes the device to start dispensing by stored pattern based on the 'default' pattern for the current plate type setting. This is equivalent to pressing Start on the device front panel.
DispensePatternRunStored	2	This will cause the device to run the dispense pattern stored at the specified location.
DispensePatternWrite	3	This is used to copy the pattern from the client application into the specified memory location on the device.
DispensePatternRead	4	This action value is used to cause the pattern stored at the specified location on the device to be loaded into the pointer variable specified by the client application.
DispensePatternStoreDefault	5	This is used to copy the default pattern into the specified memory location on the device.
DispensePatternLoadDefault	6	This is used to overwrite the default pattern with the one stored at the specified memory location.

2.3.3.9 WMTweakTable

This list of values is used in conjunction with the TweakTable procedure to specify the actions associated with volume pump motor trim settings.

Enumeration	Value	Comment
TweakTableEnable	1	Used to enable the use of the tweak table parameters during pattern dispensing.
TweakTableDisable	2	Used to disable the use of the tweak table parameters during pattern dispensing.
TweakTableWrite	3	Used to download from the client application a tweak table value dataset.
TweakTableRead	4	Used to populate a pointer variable with the current tweak table stored on the device.
TweakTableQuery	5	Used to populate a pointer variable with the current status which can be used to determine if a tweak table will be used during dispensing.
TweakTableReadActive	6	Used to populate a pointer variable used to determine which tweak table is to be used (factory default or user defined).
TweakTableWriteActive	7	Used to set which tweak table to use (factory default or user defined).

2.3.3.10 WMPumpCycleCount

This list of values is used in conjunction with the PumpCycleCount procedure to specify the actions associated with reading and writing the pump motor cycle usage count.

Enumeration	Value	Comment
PumpCycleCountWrite	1	Used to write a new value to the pump motor. This is useful if the pump motor has been replaced.
PumpCycleCountRead	2	Used to populate a pointer variable with the current value stored on the device.

2.3.3.11 WMDispenseDelay

This list of values is used in conjunction with the PumpCycleCount procedure to specify the actions associated with reading and writing the pump motor step usage count.

Enumeration	Value	Comment
DispenseDelayWrite	1	Used to set the delay time, in milliseconds, that is used after each column dispense.
DispenseDelayRead	2	Used to populate a pointer variable with the current value stored on the device.

2.3.3.12 WMStatus

This list of values represents StatusCode values used for either general device status or to represent error conditions.

Enumeration	Value	Comment
StatusIdle	1001	The RS232 port is open and the device is waiting for a command
StatusBusy	1002	The device is currently performing an action
StatusPaused	1003	A pause command has been issued
StatusAbort	1004	An abort command has been issued
StatusErrComPort	2000	An error occurred when opening the RS232 serial communication port
StatusErrE01	2001	The device issued an E01 error (stage is not at the home position or the stage motor has malfunctioned)
StatusErrE02	2002	The device issued an E02 error (stage is not at the dispensing position or the stage motor has malfunctioned)
StatusErrE03	2003	The device issued an E03 error (the plate type switched was moved when the device was busy with an action or the sensor has malfunctioned)
StatusErrE04	2004	The device issued an E04 error (the pump cover is open)
StatusErrE99	2005	The device issued an E99 error (the device received an incorrect command or the command had a syntax error)
StatusErrBadParam	2006	The client application has passed an invalid action parameter to a procedure
StatusErrBadColumn	2008	The client application has passed an invalid column value to the MoveStage procedure
StatusErrBadOffset	2009	The client application has passed an invalid offset value to the MoveStage procedure
StatusErrBadSpeedValue	2010	The client application has passed an invalid speed value to the SetSpeed procedure
StatusErrBadPumpVolume	2011	The client application has passed an invalid volume value to the Pump procedure
StatusErrBadPauseValue	2012	The client application has passed an invalid period value to the Pause procedure
StatusErrBadDispensePatternValue	2013	The client application has passed an invalid pattern number value to the DispensePattern procedure
StatusErrBadDispensePatternSequence	2014	The client application has passed an invalid dispense pattern sequence to the DispensePattern procedure. The dispense pattern sequence is a text string containing a concatenation of binary values representing whether a column is to have liquid dispensed into it.
StatusErrBadTweakTableDispensePattern	2015	The client application has passed an invalid tweak table pattern string.
StatusErrBadPumpCycleCount	2016	The client application has passed a pump motor cycle value which is out of range.
StatusErrBadTweakTableActiveTable	2017	The client application has passed a tweak table active table value which is out of range.
StatusErrBadDispenseDelay	2018	The client application has passed a dispense delay value which is out of range.

A Appendix - Sample Application

A sample Visual Basic 6 project is included within the InstallShield installation manifest. To install the necessary file choose the *Custom* installation option and select to install the sample application and source files.

The sample application can be used as a reference or template of attribute and procedure usage. This sample application will allow a user to directly control a WellMate device by issuing individual commands or by utilising a rudimentary built in protocol scripting language.

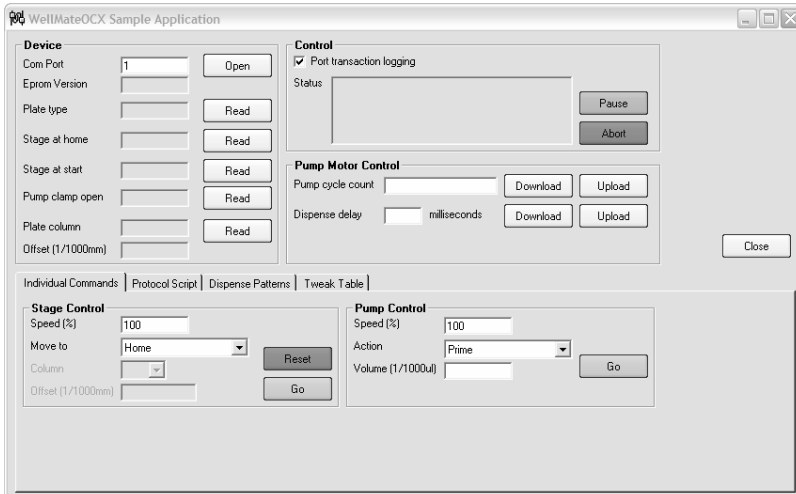


Figure 4 : Sample application dialogue screen 1 (individual command control)

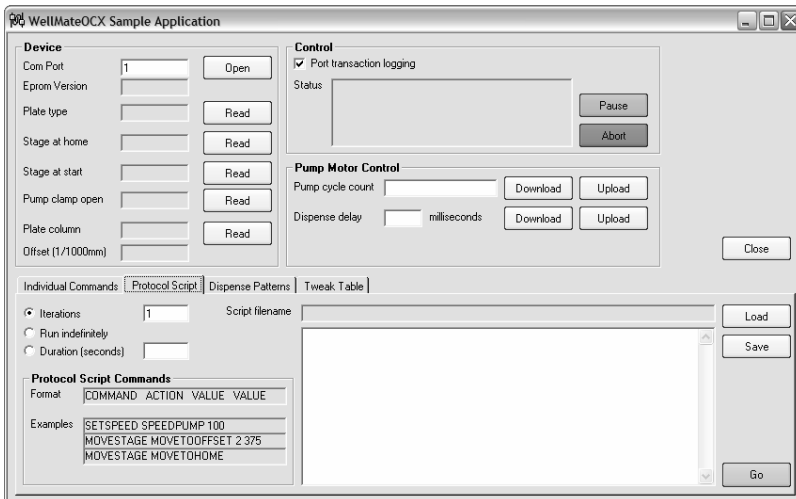


Figure 5 : Sample application dialogue screen 2 (protocol scripts)

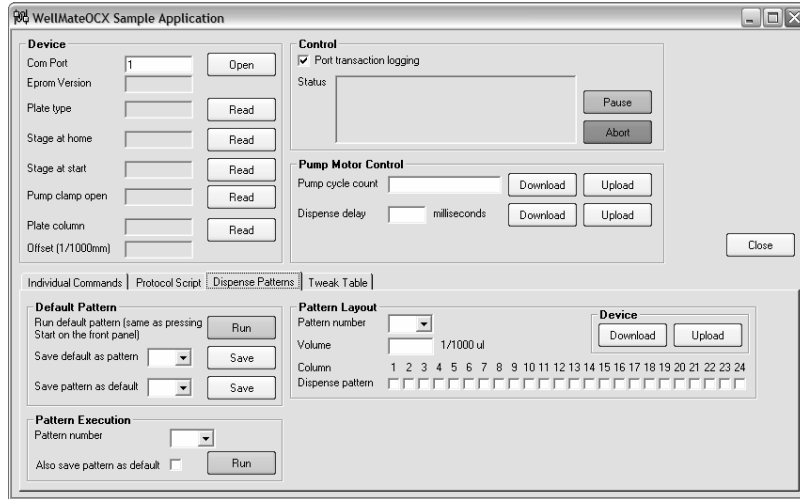


Figure 6 : Sample application dialogue screen 3 (dispense patterns)

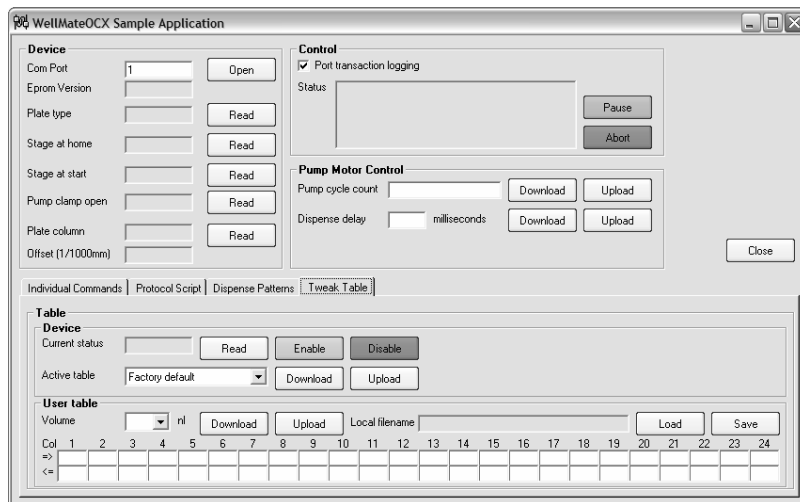


Figure 7 : Sample application dialogue screen 4 (tweak table)

A1 Sample Application Source Code

The source code listed here, for Microsoft Visual Basic version 6 coding purposes only, is included here for example illustration and reference.

The source code listed here is extracted from the sample application which is supplied with the component.

```
Option Explicit

'global variables
Private WithEvents moWellMateOCX As WellMateOCX.Connect
Private mbAbort As Boolean
Private mbPumpSpeedChanged As Boolean
Private mbStageSpeedChanged As Boolean

'declare label index constants
Private Const conEPROMVERSION As Integer = 0
Private Const conPLATETYPE As Integer = 1
Private Const conSTAGEATHOME As Integer = 2
Private Const conSTAGEATSTART As Integer = 3
Private Const conCLAMPOPEN As Integer = 4
Private Const conPLATECOLUMN As Integer = 5
Private Const conWELLOFFSET As Integer = 6
Private Const conCOLUMN As Integer = 11
Private Const conOFFSET As Integer = 10
Private Const conTWEAKSTATUS As Integer = 7

'declare button index constants
Private Const conCMDPORT As Integer = 0
Private Const conCMDRESET As Integer = 1
Private Const conCMDSTAGEGO As Integer = 2
Private Const conCMDPUMPGO As Integer = 3
Private Const conCMDPAUSE As Integer = 4
Private Const conCMDABORT As Integer = 5
Private Const conCMDPROTOCOLGO As Integer = 6
Private Const conCMDOK As Integer = 7
Private Const conCMDPLATETYPE As Integer = 8
Private Const conCMDATHOME As Integer = 9
Private Const conCMDATSTART As Integer = 10
Private Const conCMDCLAMPOPEN As Integer = 11
Private Const conCMDCOLUMN As Integer = 12
Private Const conCMDSCRIPTSAVE As Integer = 13
Private Const conCMDSCRIPTLOAD As Integer = 14
Private Const conCMDRUNDEFAULT As Integer = 21
Private Const conCMDRUNPATTERN As Integer = 16
Private Const conCMDWRITEPATTERN As Integer = 19
Private Const conCMDREADPATTERN As Integer = 18
Private Const conCMDCOPYDEFAULT As Integer = 15
Private Const conCMDCOPYPATTERN As Integer = 22
Private Const conCMDREADPumpCycleCount As Integer = 17
Private Const conCMDWRITEPumpCycleCount As Integer = 20
Private Const conCMDTWEAKLOAD As Integer = 25
Private Const conCMDTWEAKSAVE As Integer = 26
Private Const conCMDTWEAKENABLE As Integer = 23
Private Const conCMDTWEAKDISABLE As Integer = 24
Private Const conCMDTWEAKREAD As Integer = 27
Private Const conCMDTWEAKWRITE As Integer = 28
Private Const conCMDTWEAKQUERY As Integer = 29
Private Const conCMDTWEAKREADACTIVE As Integer = 30
Private Const conCMDTWEAKWRITEACTIVE As Integer = 31
Private Const conCMDDISPENSEDELAYREAD As Integer = 32
Private Const conCMDDISPENSEDELAYWRITE As Integer = 33

'move command index constants
Private Const conMOVEHOME As Integer = 0
Private Const conMOVESTART As Integer = 1
Private Const conMOVEWASTE As Integer = 2
Private Const conMOVECOLUMN As Integer = 3
Private Const conMOVEOFFSET As Integer = 4

'pump action index constants
Private Const conPUMPPRIME As Integer = 0
Private Const conPUMPFILL As Integer = 1
Private Const conPUMPEMPTY As Integer = 2
Private Const conPUMPSTOP As Integer = 3
Private Const conPUMPFILLVOL As Integer = 4
Private Const conPUMPEMPTYVOL As Integer = 5

'protocol run index constants
Private Const conRUNLOOP As Integer = 0
Private Const conRUNINFINITE As Integer = 1
Private Const conRUNTIMED As Integer = 2

Private Sub Form_Load()
    On Error Resume Next

    'instantiate the ocx
    Set moWellMateOCX = New WellMateOCX.Connect

```

```

'set some default parameters
moWellMateOCX.PortLogging = CBool(chkPortLogging.Value) '(outputs to [app.path]\PortLog.txt)
cboPosition.ListIndex = 0
cboPumpAction.ListIndex = 0
cboActiveTable.ListIndex = 0
End Sub

Private Sub Form_Terminate()
On Error Resume Next

'kill objects
Set moWellMateOCX = Nothing
End Sub

Private Sub cmdButton_Click(Index As Integer)
On Error Resume Next

'switch off the abort flag
mbAbort = False

'now act on the button index clicked/pressed
Select Case Index
Case conCMDPORT
'toggle the open port mode
If cmdButton(Index).Caption = "Open" Then
If OpenPort Then
cmdButton(Index).Caption = "Close"
Else
MsgBox "Unable to open the com port"
End If
Else
ClosePort
cmdButton(Index).Caption = "Open"
End If
Case conCMDRESET
DoResetDevice
Case conCMDPAUSE
DoPause
Case conCMDABORT
DoAbort
Case conCMDOK
'this forces a pause to stop the device if it was running when the [OK] was pressed,
'it's tidier to not allow an [OK] button to close the form if the device is busy but this is only a
'sample application
moWellMateOCX.Abort
Unload Me
Case conCMDSTAGEGO
DoMoveCommands
Case conCMDPUMPGO
DoPumpCommands
Case conCMDPROTOCOLGO
DoProtocol
Case conCMDATHOME, conCMDATSTART, conCMDCLAMPOPEN, conCMDCOLUMN, conCMDPLATETYPE
DoQuery Index
Case conCMDRUNDEFAULT
RunDefault
Case conCMDRUNPATTERN
RunPattern
Case conCMDWRITEPATTERN
WritePattern
Case conCMDREADPATTERN
ReadPattern
Case conCMDCOPYDEFAULT
CopyDefault
Case conCMDCOPYPATTERN
CopyPattern
Case conCMDREADPumpCycleCount
ReadPumpCycleCount
Case conCMDWRITEPumpCycleCount
WritePumpCycleCount
Case conCMDSCRIPTSAVE
SaveScriptFile
Case conCMDSCRIPTLOAD
LoadScriptFile
Case conCMDTWEAKDISABLE
SetTweakMode False
Case conCMDTWEAKENABLE
SetTweakMode True
Case conCMDTWEAKLOAD
LoadLocalTweakTable
Case conCMDTWEAKREAD
ReadDeviceTweakTable
Case conCMDTWEAKSAVE
SaveLocalTweakTable
Case conCMDTWEAKWRITE
WriteDeviceTweakTable
Case conCMDTWEAKQUERY
QueryTweakTableStatus
Case conCMDTWEAKWRITEACTIVE
WriteActiveTweakTable
Case conCMDTWEAKREADACTIVE
ReadActiveTweakTable
Case conCMDDISPENSEDELAYREAD

```

```

        ReadDispenseDelay
    Case conCMDDISPENSEDELAYWRITE
        WriteDispenseDelay
    End Select
End Sub

Private Sub cboPosition_Click()
    On Error Resume Next

    ShowMoveFields
End Sub

Private Sub chkPortLogging_Click()
    On Error Resume Next

    moWellMateOCX.PortLogging = CBool(chkPortLogging.Value)
End Sub

Private Function OpenPort() As Boolean
    Dim bRetVal As Boolean

    On Error Resume Next

    'open the port and pick up the return value (boolean)
    bRetVal = moWellMateOCX.OpenCommPort(Val(txtComPort))

    'if all was OK then show the informaiton that is returned by the EPROM when comms are established
    If bRetVal Then
        With moWellMateOCX
            lblData(conEPROMVERSION).Caption = .EpromVersion
            lblData(conSTAGEATHOME).Caption = .StageAtHome
            lblData(conSTAGEATSTART).Caption = .StageAtStart
            lblData(conCLAMPOPEN).Caption = .PumpClampOpen
            lblData(conPLATEATYPE).Caption = .PlateType
            lblData(conPLATECOLUMN).Caption = Left(.StagePosition, 2)
            lblData(conWELLOFFSET).Caption = Mid(.StagePosition, 3)
        End With
    End If

    'set the column list combo with the correct number of columns for this plate type
    PopulateColumnList

    'set the dispense pattern combo boxes for the correct pattern number ranges
    PopulatePatternLists

    'set the tweak table enabled entries based on the plate type setting
    DisplayTweakTable ""

    OpenPort = bRetVal
End Function

Private Sub ClosePort()
    On Error Resume Next

    'close port should always be called when you have finished
    moWellMateOCX.CloseCommPort
End Sub

Private Sub ShowMoveFields()
    On Error Resume Next

    'what type of move is required
    Select Case cboPosition.ListIndex
        Case conMOVEHOME, conMOVESTART, conMOVEWASTE
            'in these modes the column and offset are irrelevant so switch them off
            lblField(conCOLUMN).Enabled = False
            lblField(conOFFSET).Enabled = False
        Case conMOVECOLUMN
            'only the column is relevant here
            PopulateColumnList
            lblField(conCOLUMN).Enabled = True
            lblField(conOFFSET).Enabled = False
        Case conMOVEOFFSET
            'we need both column and well offset in this mode
            PopulateColumnList
            lblField(conCOLUMN).Enabled = True
            lblField(conOFFSET).Enabled = True
    End Select

    'enable or diasble (and change background colours) of the column and offset fields as required
    cboColumn.Enabled = lblField(conCOLUMN).Enabled
    cboColumn.BackColor = IIf(cboColumn.Enabled, vbWindowBackground, vbButtonFace)
    txtOffset.Enabled = lblField(conOFFSET).Enabled
    txtOffset.BackColor = IIf(txtOffset.Enabled, vbWindowBackground, vbButtonFace)
End Sub

Private Sub PopulateColumnList()
    Dim i As Integer
    Dim iMaxCols As Integer

    On Error Resume Next

    'how many columns ?

```

```

iMaxCols = IIf(moWellMateOCX.PlateType = PlateType96, 12, 24)

'lose the current list
cboColumn.Clear

'now build the list from 1 to the maximum columns available for this plate type
For i = 1 To iMaxCols
    cboColumn.AddItem CStr(i)
Next i

'...and show the first value (aesthetically pleasing, that's all)
cboColumn.ListIndex = 0
End Sub

Private Sub DoAbortRetryIgnore()
    Dim iKey As Integer
    Dim bRetVal As Boolean

    On Error Resume Next

    'loop until either someone gives up somewhere or we get a positive response to either the user ignoring the error
    'or the device is OK retrying
    Do
        'do they want to abort or not ?
        iKey = MsgBox("Error (" & moWellMateOCX.StatusCode & ") " & vbCrLf & moWellMateOCX.StatusText,
vbAbortRetryIgnore, "Error")

        Select Case iKey
            Case vbIgnore
                bRetVal = True
            Case vbRetry
                bRetVal = moWellMateOCX.Continue
            Case vbAbort
                moWellMateOCX.ResetDevice
                mbAbort = True
                bRetVal = True
        End Select

        DoEvents

    Loop Until bRetVal Or mbAbort
End Sub

Private Sub DoResetDevice()
    On Error Resume Next

    If Not moWellMateOCX.ResetDevice Then
        DoAbortRetryIgnore
    End If

    mbAbort = False
End Sub

Private Sub DoPause(Optional lDuration As Long = 0)
    On Error Resume Next

    'there are two types of pause options available within the same command...
    If lDuration = 0 Then
        If Not moWellMateOCX.Pause(PauseInfinite) Then
            DoAbortRetryIgnore
        End If
    Else
        If Not moWellMateOCX.Pause(PausePeriod, lDuration) Then
            DoAbortRetryIgnore
        End If
    End If
End Sub

Private Sub DoAbort()
    On Error Resume Next

    If Not moWellMateOCX.Abort Then
        DoAbortRetryIgnore
    End If

    mbAbort = True
End Sub

Private Sub DoSetStageSpeed(ByVal lSpeedValue As Long)
    On Error Resume Next

    If Not moWellMateOCX.SetSpeed(SpeedStage, lSpeedValue) Then
        DoAbortRetryIgnore
    End If
End Sub

Private Sub DoSetPumpSpeed(ByVal lSpeedValue As Long)
    On Error Resume Next

    If Not moWellMateOCX.SetSpeed(SpeedPump, lSpeedValue) Then
        DoAbortRetryIgnore
    End If
End Sub

Private Sub DoMoveCommands()

```

```

Dim bRetVal As Boolean

On Error Resume Next

'if the user has changed the stage speed value then we need to tell the device
If mbStageSpeedChanged Then
    DoSetStageSpeed CLng(txtStageSpeed)
    mbStageSpeedChanged = False
End If

'now do the move action
Select Case cboPosition.ListIndex
    Case conMOVEHOME
        bRetVal = moWellMateOCX.MoveStage(MoveToHome)
    Case conMOVESTART
        bRetVal = moWellMateOCX.MoveStage(MoveToStart)
    Case conMOVEWASTE
        bRetVal = moWellMateOCX.MoveStage(MoveToWaste)
    Case conMOVECOLUMN
        bRetVal = moWellMateOCX.MoveStage(MoveToColumn, cboColumn)
    Case conMOVEOFFSET
        bRetVal = moWellMateOCX.MoveStage(MoveToOffset, cboColumn, Val(txtOffset))
End Select

'if we got an error then tell the user and ask them how to continue
If Not bRetVal Then
    DoAbortRetryIgnore
End If
End Sub

Private Sub DoPumpCommands()
    Dim bRetVal As Boolean

    On Error Resume Next

    'if the user has changed the pump speed value then we need to tell the device
    If mbPumpSpeedChanged Then
        DoSetPumpSpeed CLng(txtPumpSpeed)
        mbPumpSpeedChanged = False
    End If

    'now do the pump action
    Select Case cboPumpAction.ListIndex
        Case conPUMPPRIME
            bRetVal = moWellMateOCX.Pump(PumpHome)
        Case conPUMPFILL
            bRetVal = moWellMateOCX.Pump(PumpFill)
        Case conPUMPEMPTY
            bRetVal = moWellMateOCX.Pump(PumpEmpty)
        Case conPUMPSTOP
            bRetVal = moWellMateOCX.Pump(PumpStop)
        Case conPUMPFILLVOL
            bRetVal = moWellMateOCX.Pump(PumpFill, Val(txtPumpVolume))
        Case conPUMPEMPTYVOL
            bRetVal = moWellMateOCX.Pump(PumpEmpty, Val(txtPumpVolume))
    End Select

    'if we got an error then tell the user and ask them how to continue
    If Not bRetVal Then
        DoAbortRetryIgnore
    End If
End Sub

Private Function DoProtocol() As Boolean
    Dim sTerm() As String
    Dim lTerm As Long
    Dim sParam() As String
    Dim lCmd As Long
    Dim iLoop As Integer
    Dim iMaxLoops As Integer
    Dim nStartTime As Single
    Dim nEndTime As Single
    Dim sCmd As String

    On Error Resume Next

    'determine how we stop the loop
    If optRun(conRUNLOOP).Value Then
        'iteration based
        iMaxLoops = Val(txtRunLoop)
    ElseIf optRun(conRUNTIMED).Value Then
        'run for a set time, the thought process here is to capture the time when this should end and set the maximum
        loops
        'to 1, the catch at the end of each iteration checks to see if the end time has been reached and if it has then
        it
        'sets the current loop to 1 which matches the maximum loops being set here and so exits
        iMaxLoops = 1
        nEndTime = Timer + CSng(txtRunTime)
    Else
        'this is the infinite loop catch and since the loop will never be incremented in this mode this will never be
        true
        '(ie if loop starts at zero then it can never equal 1 if it never gets incremented !)
        iMaxLoops = 1
    End If
End Function

```

```

lTerm = StringToArray(sTerm, vbCrLf, txtProtocol)

Do
  For lCmd = 0 To lTerm - 1
    'parameters are space delimited
    StringToArray sParam, " ", sTerm(lCmd)

    'switch based on command
    Select Case sParam(0)
      Case "RESET"
        DoResetDevice
      Case "SETSPEED"
        'set which speed ?
        Select Case sParam(1)
          Case "SPEEDSTAGE"
            DoSetStageSpeed CLng(sParam(2))
          Case "SPEEDPUMP"
            DoSetPumpSpeed CLng(sParam(2))
        End Select
      Case "MOVESTAGE"
        'what type of move ?
        'the actions set the move position combo and other associated fields on the main form and
        'then call DoMoveCommands()
        Select Case sParam(1)
          Case "MOVETOHOME"
            cboPosition.ListIndex = conMOVEHOME
            DoMoveCommands
          Case "MOVETOSTART"
            cboPosition.ListIndex = conMOVESTART
            DoMoveCommands
          Case "MOVETOWASTE"
            cboPosition.ListIndex = conMOVEWASTE
            DoMoveCommands
          Case "MOVETOCOLUMN"
            cboPosition.ListIndex = conMOVECOLUMN
            cboColumn.ListIndex = Val(sParam(2)) - 1
            DoMoveCommands
          Case "MOVETOOFFSET"
            cboPosition.ListIndex = conMOVEOFFSET
            cboColumn.ListIndex = Val(sParam(2)) - 1
            txtOffset = sParam(3)
            DoMoveCommands
        End Select
      Case "PUMP"
        'pump what ?
        'the actions set the pump action combo and other associated fields on the main form and
        'then call DoPumpCommands()
        Select Case sParam(1)
          Case "PUMPHOME"
            cboPumpAction.ListIndex = conPUMPPRIME
            DoPumpCommands
          Case "PUMPFILL"
            'this is a combined command so if we have a volume then it's a discrete fill otherwise
            'it runs forever
            sCmd = sParam(2)
            If sCmd <> "" Then
              cboPumpAction.ListIndex = conPUMPFILLVOL
              txtPumpVolume = sParam(2)
            Else
              cboPumpAction.ListIndex = conPUMPFILL
            End If
            DoPumpCommands
          Case "PUMPEMPTY"
            'this is a combined command so if we have a volume then it's a discrete empty otherwise
            'it runs forever
            sCmd = sParam(2)
            If sCmd <> "" Then
              cboPumpAction.ListIndex = conPUMPEMPTYVOL
              txtPumpVolume = sParam(2)
            Else
              cboPumpAction.ListIndex = conPUMPEMPTY
            End If
            DoPumpCommands
          Case "PUMPSTOP"
            cboPumpAction.ListIndex = conPUMPSTOP
            DoPumpCommands
        End Select
      Case "PAUSE"
        'what type of pause ?
        Select Case sParam(1)
          Case "PAUSEINFINITE"
            DoPause
          Case "PAUSEWAIT"
            DoPause CLng(sParam(2))
        End Select
    End Select
  Next lCmd

  'workout how to get out of here !
  If optRun(conRUNLOOP).Value Then
    'obviously increments the loop (and also resets the process back to the first command)
    If lCmd >= lTerm - 1 Then
      iLoop = iLoop + 1
    End If
  End If
End Do

```

```

        lCmd = 0
    End If
    ElseIf optRun(conRUNTIMED).Value Then
        'look to see if we have run out of time
        If Timer < nEndTime Then
            'we haven't run out of time, so check to see if we have run out of commands and if we have then start
            'from the first command again (obviously)
            If lCmd = lTerm - 1 Then
                lCmd = 0
            End If
        Else
            'if we have reached here then end time then simple set the loop counter to 1 (which in this mode will
            'be
            'the same as the maximum loops allowed so forces us to get out
            iLoop = 1
        End If
    End If

    'let the cpu catch up
    DoEvents

    Loop Until mbAbort Or iLoop >= iMaxLoops
End Function

Private Sub DoQuery(ByVal iIndex As Integer)
    Dim bRetVal As Boolean
    Dim lQuery As WMQuery
    Dim vData As Variant

    On Error Resume Next

    'determine the query type
    Select Case iIndex
        Case conCMDATHOME
            lQuery = QueryStageAtHome
        Case conCMDATSTART
            lQuery = QueryStageAtStart
        Case conCMDCLAMPOPEN
            lQuery = QueryPumpClamp
        Case conMDCOLUMN
            lQuery = QueryColumnPosition
        Case conCMDPLATETYPE
            lQuery = QueryPlateType
    End Select

    'query the device
    bRetVal = moWellMateOCX.Query(lQuery, vData)

    'if we got an error then tell the user and ask them how to continue otherwise get the data and do something with it
    If bRetVal Then
        'determine the query type and set the associated label
        Select Case iIndex
            Case conCMDATHOME
                lblData(conSTAGEATHOME) = vData
            Case conCMDATSTART
                lblData(conSTAGEATSTART) = vData
            Case conCMDCLAMPOPEN
                lblData(conCLAMPOPEN) = vData
            Case conMDCOLUMN
                lblData(conPLATECOLUMN).Caption = Left(vData, 2)
                lblData(conWELLOFFSET).Caption = Mid(vData, 3)
            Case conCMDPLATETYPE
                lblData(conPLATETYPE).Caption = vData
                PopulatePatternLists
                DisplayTweakTable ""
        End Select
    Else
        'oops! something went wrong
        DoAbortRetryIgnore
    End If
End Sub

Private Sub moWellMateOCX_StatusChanged(ByVal lStatusCode As WellMateOCX.WMStatus, ByVal sStatusText As String)
    lblStatus = "(" & lStatusCode & ")" & vbCrLf & sStatusText
End Sub

Private Sub txtProtocol_KeyPress(KeyAscii As Integer)
    'we want the protocol list in uppercase where applicable
    KeyAscii = Asc(UCase(Chr(KeyAscii)))
End Sub

Private Sub txtPumpSpeed_Change()
    'set the semaphore
    mbPumpSpeedChanged = True
End Sub

Private Sub txtStageSpeed_Change()
    'set the semaphore
    mbStageSpeedChanged = True
End Sub

Public Function StringToArray(ByRef Target() As String, ByVal ParseChr As String, ByVal Source As String) As Long
    Dim i As Long
    Dim LastPosn As Long

```

```

Dim ParseChrLen As Integer

On Error Resume Next

ParseChrLen = Len(ParseChr)

'ensure that this isn't a null string
If Right(Source, ParseChrLen) <> ParseChr Then
    Source = Source & ParseChr
End If

'build the first array dimension
ReDim Target(1) As String
Target(0) = Left(Source, InStr(1, Source, ParseChr) - 1)
LastPosn = Len(Target(0)) + ParseChrLen
i = 0

'now loop until all items read
While LastPosn < Len(Source)
    i = i + 1
    ReDim Preserve Target(i + 1) As String
    Target(i) = Mid(Source, LastPosn + 1, InStr(LastPosn + 1, Source, ParseChr) - LastPosn - 1)
    LastPosn = LastPosn + Len(Target(i)) + ParseChrLen
Wend

'and return the number of lines read
StringToArray = i + 1
End Function

Private Sub RunDefault()
    Dim bRetVal As Boolean

    On Error Resume Next

    'issue the command
    bRetVal = moWellMateOCX.DispensePattern(DispensePatternRunCurrent)

    'if we got an error then tell the user and ask them how to continue
    If Not bRetVal Then
        'oops! something went wrong
        DoAbortRetryIgnore
    End If
End Sub

Private Sub RunPattern()
    Dim bRetVal As Boolean

    On Error Resume Next

    'issue the command
    bRetVal = moWellMateOCX.DispensePattern(DispensePatternRunStored, CLng(cboPatternRun.Text),
CBool(chkSaveAsDefault.Value))

    'if we got an error then tell the user and ask them how to continue
    If Not bRetVal Then
        'oops! something went wrong
        DoAbortRetryIgnore
    End If
End Sub

Private Sub WritePattern()
    Dim i As Integer
    Dim sPattern As String
    Dim bRetVal As Boolean

    On Error Resume Next

    For i = 0 To IIf(moWellMateOCX.PlateType = PlateType96, 11, 23)
        sPattern = sPattern & CStr(chkPatternColumn(i).Value)
    Next i

    'issue the command
    bRetVal = moWellMateOCX.DispensePattern(DispensePatternWrite, CLng(cboPatternQuery.Text),
CLng(Val(txtPatternVolume)), sPattern)

    'if we got an error then tell the user and ask them how to continue
    If Not bRetVal Then
        'oops! something went wrong
        DoAbortRetryIgnore
    End If
End Sub

Private Sub ReadPattern()
    Dim i As Integer
    Dim sData As String
    Dim bRetVal As Boolean
    Dim bEnabled As Boolean

    On Error Resume Next

    'issue the command, a vload call will populate the pointer sData
    bRetVal = moWellMateOCX.DispensePattern(DispensePatternRead, CLng(cboPatternQuery.Text), , , sData)

```

```

'if we got an error then tell the user and ask them how to continue
If bRetVal Then
    'the first 5 character sof the returned data represents the volume
    txtPatternVolume = Val(Left(sData, 5))

'now populate each pattern checkbox based on the return data string representing binary '1' or '0' for on or
off
For i = 0 To IIf(moWellMateOCX.PlateType = PlateType96, 11, 23)
    chkPatternColumn(i).Value = Val(Mid(sData, 6 + i, 1))
Next i

bEnabled = moWellMateOCX.PlateType = PlateType384

For i = 12 To 23
    chkPatternColumn(i).Value = bEnabled
    If Not chkPatternColumn(i).Enabled Then
        chkPatternColumn(i).Value = vbUnchecked
    End If
    lblColumn(i).Enabled = bEnabled
Next i

Else
    'oops! something went wrong
    DoAbortRetryIgnore
End If
End Sub

Private Sub CopyDefault()
    Dim bRetVal As Boolean

    On Error Resume Next

    'issue the command
    bRetVal = moWellMateOCX.DispensePattern(DispensePatternStoreDefault, CLng(cboDefaultToPattern.Text))

    'if we got an error then tell the user and ask them how to continue
    If Not bRetVal Then
        'oops! something went wrong
        DoAbortRetryIgnore
    End If
End Sub

Private Sub CopyPattern()
    Dim bRetVal As Boolean

    On Error Resume Next

    'issue the command
    bRetVal = moWellMateOCX.DispensePattern(DispensePatternLoadDefault, CLng(cboPatternToDefault.Text))

    'if we got an error then tell the user and ask them how to continue
    If Not bRetVal Then
        'oops! something went wrong
        DoAbortRetryIgnore
    End If
End Sub

Private Sub PopulatePatternLists()
    Dim i As Integer
    Dim bEnabled As Boolean

    PopulatePatternCombo cboPatternToDefault
    PopulatePatternCombo cboDefaultToPattern
    PopulatePatternCombo cboPatternQuery
    PopulatePatternCombo cboPatternRun

    bEnabled = moWellMateOCX.PlateType = PlateType384

    For i = 12 To 23
        chkPatternColumn(i).Enabled = bEnabled
        If Not chkPatternColumn(i).Enabled Then
            chkPatternColumn(i).Value = vbUnchecked
            chkPatternColumn(i).BackColor = vbButtonFace
        Else
            chkPatternColumn(i).BackColor = vbWindowBackground
        End If
        lblColumn(i).Enabled = bEnabled
    Next i
End Sub

Private Sub PopulatePatternCombo(ByRef cboTarget As ComboBox)
    Dim iEnd As Integer
    Dim i As Integer

    cboTarget.Clear

    For i = IIf(moWellMateOCX.PlateType = PlateType96, 0, 10) To IIf(moWellMateOCX.PlateType = PlateType96, 9, 19)
        cboTarget.AddItem i
    Next i

    cboTarget.ListIndex = 0
End Sub

```

```

Private Sub WritePumpCycleCount()
    Dim bRetVal As Boolean
    Dim lData As Long

    On Error Resume Next

    lData = Val(txtPumpCycleCount)

    'issue the command
    bRetVal = moWellMateOCX.PumpCycleCount(PumpCycleCountWrite, lData)

    'if we got an error then tell the user and ask them how to continue
    If Not bRetVal Then
        'oops! something went wrong
        DoAbortRetryIgnore
    End If
End Sub

Private Sub ReadPumpCycleCount()
    Dim bRetVal As Boolean
    Dim lData As Long

    On Error Resume Next

    'issue the command
    bRetVal = moWellMateOCX.PumpCycleCount(PumpCycleCountRead, lData)

    'if we got an error then tell the user and ask them how to continue
    If bRetVal Then
        txtPumpCycleCount = lData
    Else
        'oops! something went wrong
        DoAbortRetryIgnore
    End If
End Sub

Private Sub SaveScriptFile()
    Dim sFilename As String

    On Error Resume Next

    'set the CommonDialog parameters
    With cdgFile
        .DialogTitle = "Save WellMate Protocol Script"
        .Filter = "All Files|*.*|WellMate Protocol Files|.wmp|Text Files|.txt"
        .FilterIndex = 2
        .Flags = cdLOFNHideReadOnly Or cdLOFNOverwritePrompt
        .ShowSave
        sFilename = .FileName
    End With

    'show the filename
    lblFilename = sFilename

    'if the user saved the file then the filename field will be populated, so save the file !
    If Len(sFilename) > 0 Then
        'using legacy stuff here. Use the FSO objects instead, much neater !!
        'note, we are opening the file for overwrite mode
        Open sFilename For Output As #1

        'write the text box contents
        Print #1, txtProtocol

        'close the file
        Close #1
    End If
End Sub

Private Sub LoadScriptFile()
    Dim sFilename As String
    Dim sTmp As String
    Dim sBuffer As String

    On Error Resume Next

    'set the CommonDialog parameters
    With cdgFile
        .DialogTitle = "Open WellMate Protocol Script"
        .Filter = "All Files|*.*|WellMate Protocol Files|.wmp|Text Files|.txt"
        .FilterIndex = 2
        .FileName = lblFilename
        .ShowOpen
        sFilename = .FileName
    End With

    'if the user selected a file then the filename field will be populated, so open it !
    If Len(sFilename) > 0 Then
        'show the filename
        lblFilename = sFilename

        'using legacy stuff here. Use the FSO objects instead, much neater !!
        'note, we are opening the file for overwrite mode
        Open sFilename For Input As #1
    End If
End Sub

```

```

        'now read the file
    Do
        'get the data and append it to the temporary buffer
        Input #1, sTmp
        sBuffer = sBuffer & sTmp & vbCrLf
    Loop Until EOF(1)

    'show the script file contents
    txtProtocol.Text = sBuffer

    'close the file
    Close #1
End If

End Sub

Private Sub SetTweakMode(ByVal bMode As Boolean)
    Dim bRetVal As Boolean
    Dim iTweakMode As WMTweakTable

    On Error Resume Next

    'issue the command
    iTweakMode = IIf(bMode, TweakTableEnable, TweakTableDisable)
    bRetVal = moWellMateOCX.TweakTable(iTweakMode)

    'if we got an error then tell the user and ask them how to continue
    If Not bRetVal Then
        'oops! something went wrong
        DoAbortRetryIgnore
    End If
End Sub

Private Sub LoadLocalTweakTable()
    Dim sFilename As String
    Dim sTmp As String
    Dim lVolume As Long
    Dim sTable As String

    On Error Resume Next

    'set the CommonDialog parameters
    With cdgFile
        .DialogTitle = "Open WellMate Tweak Table File"
        .Filter = "All Files|*.*|WellMate Tweak Table Files|.wmt|Text Files|.txt"
        .FilterIndex = 2
        .FileName = lblTweakFilename
        .ShowOpen
        sFilename = .FileName
    End With

    'if the user selected a file then the filename field will be populated, so open it !
    If Len(sFilename) > 0 Then
        'show the filename
        lblTweakFilename = sFilename

        'using legacy stuff here. Use the FSO objects instead, much neater !!
        'note, we are opening the file for overwrite mode
        Open sFilename For Input As #1

        'get the volume
        Input #1, sTmp
        lVolume = Val(sTmp)

        'get the tweak table
        Input #1, sTable

        cboTweakVolume.Text = lVolume

        'show the tweak table
        DisplayTweakTable sTable

        'close the file
        Close #1
    End If
End Sub

Private Sub DisplayTweakTable(ByVal sTable As String)
    Dim iCount As Integer
    Dim i As Integer
    Dim iCol As Integer
    Dim bEnabled As Boolean

    On Error Resume Next

    bEnabled = moWellMateOCX.PlateType = PlateType384

    'get the number of table entries
    iCount = Len(sTable)

    'set the enabled/disabled and background properties based on plate type in use
    For i = 0 To 47
        If i > 11 Then
            txtTweakColumn(i).Enabled = bEnabled
        End If
    Next i
End Sub

```

```

        txtTweakColumn(i).BackColor = IIf(bEnabled, vbWindowBackground, vbButtonFace)
        lblTweakColumn(i).Enabled = bEnabled
    End If

    txtTweakColumn(i) = ""
Next i

lblTweakRow(1).Enabled = bEnabled

'each entry is in the format of sign then 3 digit tweak value (eg +021, -014) so deviding by 4 must result in zero
remainder
'if it doesn't then the table is corrupt or incorrect
If iCount Mod 4 = 0 Then
    For i = 1 To iCount Step 4
        'if the current plate type setting on the device is 96 wells then the tweak table cannot allow entries for
384 wells
        txtTweakColumn(iCol) = Mid(sTable, i, 4)
        iCol = iCol + 1
    Next i
End If
End Sub

Private Sub SaveLocalTweakTable()
    Dim sFilename As String
    Dim i As Integer
    Dim sTable As String

    On Error Resume Next

    'set the CommonDialog parameters
    With cdgFile
        .DialogTitle = "Save WellMate Tweak Table File"
        .Filter = "All Files|*.*|WellMate Tweak Table Files|.wmt|Text Files|.txt"
        .FilterIndex = 2
        .FileName = lblTweakFilename
        .Flags = cdloFNHideReadOnly Or cdloFNOverwritePrompt
        .ShowSave
        sFilename = .FileName
    End With

    'show the filename
    lblTweakFilename = sFilename

    'if the user selected a file then the filename field will be populated, so open it !
    If Len(sFilename) > 0 Then
        'using legacy stuff here. Use the FSO objects instead, much neater !!
        'note, we are opening the file for overwrite mode
        Open sFilename For Output As #1

        'write the volume
        Print #1, cboTweakVolume.Text

        'write the tweak table
        For i = 0 To 47
            'tweak table entries are consecutive columns from the first column, blank columns are not allowed
            If Len(txtTweakColumn(i)) > 0 Then
                Print #1, txtTweakColumn(i);
                sTable = sTable & txtTweakColumn(i)
            Else
                Exit For
            End If
        Next i

        'close the file
        Close #1

        'now redraw the tweak table, this removes any non consecutive values
        DisplayTweakTable sTable
    End If
End Sub

Private Sub txtTweakColumn_LostFocus(Index As Integer)
    On Error Resume Next

    'entry must be prefixed with a sign
    If Left(txtTweakColumn(Index), 1) = "+" Or Left(txtTweakColumn(Index), 1) = "-" Then
        'the entry must be numeric, padded with zeros
        txtTweakColumn(Index) = Left(txtTweakColumn(Index), 1) & Format(Val(Mid(txtTweakColumn(Index), 2)), "000")
    End If
End Sub

Private Sub WriteDeviceTweakTable()
    Dim bRetVal As Boolean
    Dim sCount As String
    Dim i As Integer
    Dim sTable As String

    On Error Resume Next

    'now we need to add up the tweak table entries, simply by finding the first empty one
    For i = 0 To 47
        If Len(txtTweakColumn(i)) = 0 Then
            Exit For
        Else

```

```

        sTable = sTable & txtTweakColumn(i)
    End If
Next i

sCount = Format(i, "00")

'issue the command
bRetVal = moWellMateOCX.TweakTable(TweakTableWrite, CLng(Val(cboTweakVolume.Text)), sTable)

'if we got an error then tell the user and ask them how to continue
If Not bRetVal Then
    'oops! something went wrong
    DoAbortRetryIgnore
End If
End Sub

Private Sub ReadDeviceTweakTable()
    Dim bRetVal As Boolean
    Dim sData As String

    On Error Resume Next

    'issue the command
    bRetVal = moWellMateOCX.TweakTable(TweakTableRead, CLng(Val(cboTweakVolume.Text)), sData)

    'if we got an error then tell the user and ask them how to continue
    If bRetVal Then
        'otherwise show the data
        DisplayTweakTable Mid(sData, 3)
    Else
        'oops! something went wrong
        DoAbortRetryIgnore
    End If
End Sub

Private Sub QueryTweakTableStatus()
    Dim bRetVal As Boolean
    Dim bStatus As Boolean

    On Error Resume Next

    'issue the command
    bRetVal = moWellMateOCX.TweakTable(TweakTableQuery, , , bStatus)

    'if we got an error then tell the user and ask them how to continue
    If bRetVal Then
        'otherwise show the data
        lblData(conTWEAKSTATUS) = CStr(bStatus)
    Else
        'oops! something went wrong
        DoAbortRetryIgnore
    End If
End Sub

Private Sub WriteActiveTweakTable()
    Dim bRetVal As Boolean
    Dim lActiveTable As Long

    On Error Resume Next

    'define the action type (use factory default, use user defined table)
    lActiveTable = cboActiveTable.ListIndex

    'issue the command
    bRetVal = moWellMateOCX.TweakTable(TweakTableWriteActive, , , lActiveTable)

    'if we got an error then tell the user and ask them how to continue
    If Not bRetVal Then
        'oops! something went wrong
        DoAbortRetryIgnore
    End If
End Sub

Private Sub ReadActiveTweakTable()
    Dim bRetVal As Boolean
    Dim lActiveTable As Long

    On Error Resume Next

    'issue the command
    bRetVal = moWellMateOCX.TweakTable(TweakTableReadActive, , , lActiveTable)

    'if we got an error then tell the user and ask them how to continue
    If bRetVal Then
        'define the action type (use factory default, use user defined table) from the return value
        cboActiveTable.ListIndex = lActiveTable
    Else
        'oops! something went wrong
        DoAbortRetryIgnore
    End If
End Sub

Private Sub WriteDispenseDelay()
    Dim bRetVal As Boolean

```

```
Dim lDelay As Long

On Error Resume Next

'define the action type (use factory default, use user defined table)
lDelay = CLng(Val(txtDispenseDelay))

'issue the command
bRetVal = moWellMateOCX.DispenseDelay(DispenseDelayWrite, lDelay)

'if we got an error then tell the user and ask them how to continue
If Not bRetVal Then
    'oops! something went wrong
    DoAbortRetryIgnore
End If
End Sub

Private Sub ReadDispenseDelay()
Dim bRetVal As Boolean
Dim lDelay As Long

On Error Resume Next

'issue the command
bRetVal = moWellMateOCX.DispenseDelay(DispenseDelayRead, lDelay)

'if we got an error then tell the user and ask them how to continue
If bRetVal Then
    'show the return value
    txtDispenseDelay = lDelay * 10
Else
    'oops! something went wrong
    DoAbortRetryIgnore
End If
End Sub
```


Matrix Technologies Corp.
22 Friars Drive
Hudson, NH 03051, USA
Toll-free: (800) 345 0206
www.matrixtechcorp.com