



Application Note 1070-209

Getting Started with Visual Basic Scripting in MultiVu

Quantum Design has incorporated a Visual Basic interpreter within MultiVu on the following platforms: PPMS, DynaCool, VersaLab and MPMS SQUID VSM. This provides much more powerful programming options to the advanced user as compared to solely using sequences. The ability to parameterize quantities, use conditional statements to optimize measurements, post-process or analyze data on the fly, and integrate 3rd party instruments into measurements are just a few examples of the benefits that are available through the use of Visual Basic scripting. If starting from scratch, this is the easiest path for integration of 3rd party instruments on Quantum Design systems. If a LabVIEW® interface is preferred, please see PPMS Application Note 1070-210 “Interfacing LabVIEW programs with Quantum Design Instruments” at www.qdusa.com.

What is Visual Basic scripting?

The Visual Basic interpreter included in MultiVu, WinWrap® Basic, is very similar to VBA provided by Microsoft. The executable programs (.BAS files) are called macros in MultiVu. We also refer to WinWrap® Basic as a *scripting* environment.

General guides to programming in Visual Basic are available on the web. For example, here is a link to a Microsoft VBA programming guide:

[http://msdn.microsoft.com/en-us/library/aa716285\(VS.60\).aspx](http://msdn.microsoft.com/en-us/library/aa716285(VS.60).aspx)

WinWrap’s language reference page is a more advanced resource:

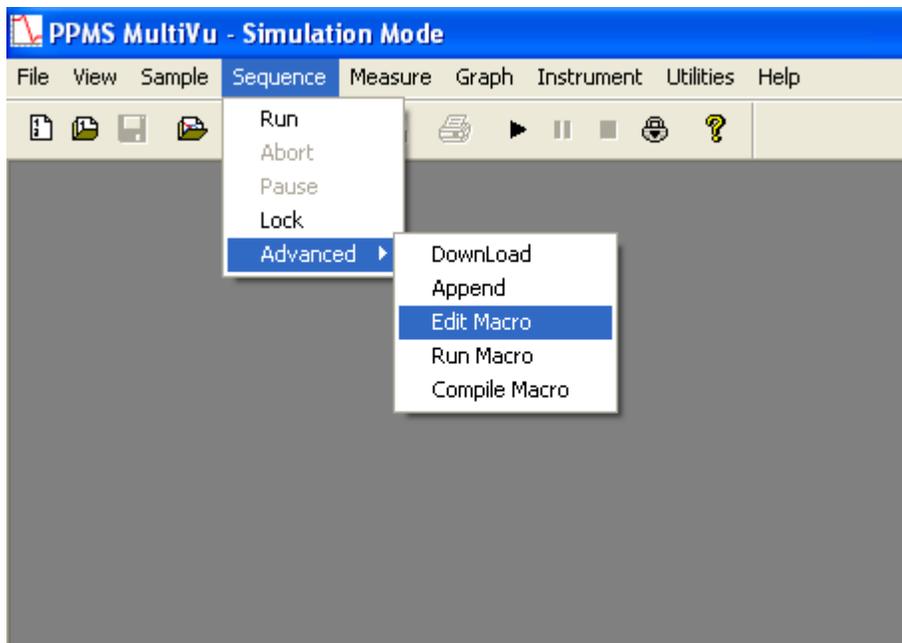
http://www.winwrap.com/web/basic/language/?p=doc_group_overview.htm

WinWrap also provides full function reference embedded into the editor in MultiVu, accessed by pressing F1 or right-clicking in the editor window. This reference material applies only to general WinWrap functions and not the MultiVu platform-specific functions (like setting or reading the temperature or magnetic field). For a dictionary of essential MultiVu function calls, see the section at the end of this application note. This dictionary is also included as a PDF file as part of a .ZIP archive attached to this application note. That .ZIP file contains example programs for interfacing external instruments, controlling the QD instrument, and creating and manipulating data files.

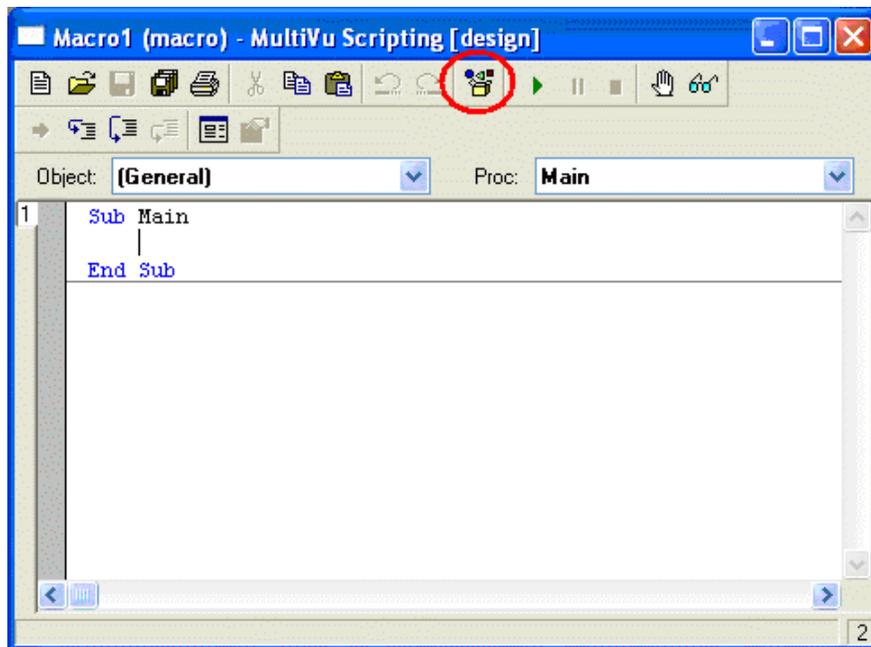
There is also a dictionary of the available function calls which can be pulled up by pressing the *Browse Object* button  (circled in red in the editor screen shot below).

How do I get started?

To start writing a Basic program from scratch, access the Edit Macro command in MultiVu under *Sequence > Advanced > Edit Macro* as shown below:



Selecting *Edit Macro* will then open up the WinWrap® Basic editor:



You can also translate an existing sequence (.SEQ file) into a Basic program using the *Sequence > Advanced > Compile Macro* command when a sequence is open in MultiVu. This can be helpful in getting started with basic instrument calls for setting field, temperature, etc. if

the attached example programs (see zip file with this app note) or the MultiVu OLE dictionary (at the end of this note) are not clear. . **IMPORTANT NOTE:** if the WinWrap editor is already open, the *Compile Macro* operation may erase the contents of the active sheet and replace it with the compiled sequence, so you should open a new blank macro before compiling a sequence.

In the attached ZIPped folder are some example macros (.BAS files), object files (.OBJ), and class files (.CLS). Please consult a programming guide for more information on the creation and use of objects and classes. Unpack this zip file into the folder C:\qdpms\macros (or C:\QDVersaLab\macros or C:\QDSquidVSM\macros, etc.). This is the default location for macro files. We suggest right-clicking on the zip file and using “Open with Compressed (zipped) folders” in order to preserve the structure of the subfolders (note that the zipped folder should unpack with subfolders as the programs will reference objects or class files using relative directory links). After unpacking the library, open macros from the editor using the *Open file* icon. You will see general examples like `MultiVu Scan Field Loop.bas` which is written so that any platform can use it and others which are platform specific like `PPMS scan V output.bas` that uses analog voltage outputs on the Model 6000 of the PPMS. There are macros for interfacing with 3rd party instruments like LCR meters, a resistance bridge, and a lock-in amplifier. Perhaps the best example to start with is `getGPIBdata` and `writetofile.BAS` (screen shot is shown on the next page) which shows the following important operations:

- 1) Interfacing with the Quantum Design instrument (set/read field and temperature) which are written in the cross-platform `MultiVu.xxx` format
- 2) Setting up and interfacing with an external resistance bridge using GPIB
- 3) Opening and writing data to a .DAT file which can be graphed in MultiVu; NOTE: some existing users may have an older version of the class file (`MVUData.cls`) for this, while a newer and more sophisticated version (`MultiVuDataFile.cls`) is included in the attached zip file

In addition to code written by Quantum Design users, some manufacturers of 3rd party instruments (e.g., Agilent) provide Visual Basic example code to interface their devices.

A useful feature of this editor is that it includes code completion as well as highlighting of recognized functions. As an example, type `Wait(` and you will see a tooltip dialog showing the format of the argument of the `Wait` function.

Please note: you use the green arrow button in the editor to run a macro, NOT using the MultiVu *Run Sequence* command.

To easily see the objects or class files that a particular macro uses (seen in the `#uses` declarations at the beginning of the program), right-click in the editor window and select *Sheet > Open Uses* and the modules will appear on new tabs on the left of the window.

When debugging code, it is helpful to keep the editor window always split in order to keep the debug information visible after execution has stopped. Do this by right-clicking in the editor and selecting *View > Always Split*.

```

'#Language "VVB-COM"
'#Uses "Utils\Utils.obm"
'#Uses "MultiVuDataFile\MultiVuDataFile.cls"
'#Uses "LS-370 bridge\LSBridge.obm"

Option Explicit

Dim lStatus As Long
Dim fTemp As Double
Dim fField As Double
Dim fRes As Double
Dim T As Double
Dim B As Double
Dim fResults As New MultiVuDataFile
' the location where the data file will be generated
' (defaults to the root data directory for the architecture the script is being run on)
' NOTE: directory will be automatically created if it does not exist
Dim sDataDirectory As String
' the file name pattern for the data file - we are using the current date/time as part of the file name
' to prevent overwriting of or appending to existing files
Const sFilenamePattern = "Scripting Data_&Y%m%d_&H%M.dat"
' labels for our data columns for easier access
Const T_col = "Temperature (K)"
Const H_col = "Field (Oe)"
Const R_col = "Resistance (Ohm)"

```

Declaration (top) and main program (bottom) of the macro getGPIBdata and writetofile.BAS.

```

Sub Main
' verify that we have the correct version of Utils.obm
Utils.CheckForRequiredVersion("Utils",Utils.Version(),25881)
' clear out any previous debug messages
Debug.Clear
' set up the default (system-dependent) data directory
sDataDirectory = Utils.systemDirectory() & "\Data\"
' initialize LS bridge (or other GPIB device)
If LSBridge.Init <> 1 Then
    Debug.Print "GPIB instrument does not initialize properly"
End If

' create a new MV data file
fResults.AddColumn(T_col, mvStartupAxisY1)
fResults.AddColumn(H_col, mvStartupAxisY2, mvLogScale)
fResults.AddColumn(R_col, mvStartupAxisY3)
fResults.CreateFileAndWriteHeader(sDataDirectory & Utils.DateAndTime(sFilenamePattern), "Example Resistance Data")
' open the newly created data file for display in MultiVu
fResults.OpenInMultiVu()

For T = 300 To 50 Step -50
    MultiVu.SetTemperature(T,50,0)
    WaitFor(1,0,0) 'mvseq:sequencename>0001 Wait for %t
    For B = -20000 To 20000 Step 1000
        MultiVu.SetField(B,300,0,0)
        ' wait for field; bitmask=2
        WaitFor(2,0,0) 'mvseq:sequencename>0002 Wait for %t
        ' read the current temperature (and status)
        MultiVu.GetTemperature(fTemp,lStatus)
        ' read the current field (and status)
        MultiVu.GetField(fField,lStatus)
        ' read the resistance from LSBridge (or other instrument)
        LSBridge.ReadRes(fRes)
        ' write the data to the file
        fResults.WriteDataUsingArray(Array( _
            T_col, fTemp, _
            H_col, fField, _
            R_col, fRes))
    Next B
Next T
MultiVu.SetField(0,300,0,0)
End Sub

```

If you are interested in sharing any code you have written, such as .OBJ files and a simple implementation example, Quantum Design welcomes your contributions at apps@qdusa.com.

Notes on scripting on the PPMS platform

If you do NOT see the *Edit / Run / Compile macro* options under the Advanced menu in PPMS MultiVu, make sure you are using MultiVu version 1.5.0 or later. Also, make sure that the shortcut used to launch MultiVu includes the command line item “-macro” (no quotes) at the end. This is seen when right-clicking the icon and selecting *Properties*, then selecting the *Shortcut* tab and looking at *Target*. Note also that much of the script writing and even testing can be done in simulation mode which saves time at the instrument’s PC.

Note that we have not yet implemented PPMS option measurements (ACMS, ACT, VSM, etc.) in macros. This is a work-in-progress. Please contact us at apps@qdusa.com for updates on this development or if you want to discuss other aspects of your application.

Interfacing GPIB-based instruments on the PPMS is best done using a dedicated GPIB card (or USB-GPIB converter) instead of sharing the same GPIB bus with the PPMS. This is because the GPIB settings for the PPMS are different than typical GPIB instruments (e.g., auto serial polling is OFF for PPMS) and the traffic to the PPMS can be very heavy.

Quantum Design MultiVu OLE Methods

This section lists OLE methods that are commonly used when communicating with MultiVu.

Conventions

Methods signatures are shown twice, first in WinWrap (Visual Basic) syntax, second in C-like syntax. Variable types are as follows (note that some types, e.g. `long`, have different meanings in different languages, so be careful):

- **double:** 64-bit floating point number
- **long:** 32-bit signed integer

Some method arguments are passed by value, and others are passed by reference. By value arguments look like this in WinWrap: `ByVal temperature As Double`, and like this in C: `double temperature`. By reference arguments look like this in WinWrap: `ByRef temperature as Double`, and like this in C: `double* temperature`. Note that you must create appropriate variables to pass as by reference arguments before making a call to a method that takes by reference arguments. Do not attempt to pass a constant number as a by reference argument.

All methods return an error code with type `long`. A value of zero indicates no error. Other values indicate an error.

Methods

SetTemperature

```
MultiVu.SetTemperature(ByVal temperature As Double, ByVal rate  
    As Double, ByVal approach As Long) As Long
```

```
long SetTemperature(double temperature, double rate, long  
approach)
```

temperature: Temperature set point in kelvin

rate: Temperature rate set point in kelvin/second

approach: Approach mode as one of the following codes

0: Fast settle

1: No overshoot

GetTemperature

```
MultiVu.GetTemperature(ByRef temperature As Double, ByRef state  
    As Long) As Long
```

```
long GetTemperature(double* temperature, long* state)
```

temperature: Temperature reported by MultiVu in kelvin.

state: Temperature status reported by MultiVu as one of the following codes

- 0: Unknown
- 1: Stable
- 2: Tracking
- 5: Near
- 6: Chasing
- 7: Filling/emptying reservoir
- 10: Standby
- 15: General failure in temp control

SetField

MultiVu.SetField(ByVal field As Double, ByVal rate as Double,
ByVal approach As Long, ByVal mode as Long) As Long
long SetField(double field, double rate, long approach, long
mode)

field: Field set point in oersted

rate: Field ramp rate set point in oersted/second

approach: Approach mode as one of the following codes

- 0: Linear
- 1: No overshoot
- 2: Oscillate

mode: Mode at end of field ramp as one of the following codes. Note that Persistent mode is not supported by VersaLab and DynaCool

- 0: Persistent
- 1: Driven

GetField

MultiVu.GetField(ByRef field As Double, ByRef state As Double)
As Long
long GetField(double* field, long* state)

field: Field reported by MultiVu in oersted

state: Field state reported by MultiVu as one of the following codes

- 0: Unknown
- 1: Persistent (PPMS), Standby (VersaLab and DynaCool)
- 2: Switch warming
- 3: Switch cooling
- 4: Holding in driven mode
- 5: Iterating
- 6: Charging
- 8: Current error
- 15: General failure in field control

SetChamber

`MultiVu.SetChamber(ByVal code As Long) As Long`

`long SetChamber(long code)`

code: Chamber mode to set as one of the following codes

- 0: Seal
- 1: Purge/Seal
- 2: Vent/Seal
- 3: Pump continuous
- 4: Vent continuous
- 5: High vacuum

GetChamber

`MultiVu.GetChamber(ByRef code As Long) As Long`

`long GetChamber(long* state)`

code: Chamber state reported by MultiVu as one of the following codes:

- 0: Unknown
- 1: Purged/Sealed
- 2: Vented/Sealed
- 3: Sealed (condition unknown)
- 4: Purging
- 5: Venting
- 6: Pre-high vacuum
- 7: High vacuum
- 8: Pumping continuously
- 9: Flooding continuously

WaitFor

`MultiVu.WaitFor(ByVal mask As Long, ByVal delay As Double, ByVal timeout As Double)`

mask: Code that determine which subsystems need to be stable in order to satisfy the WaitFor.

To wait for multiple subsystems, you set mask to the sum of the codes for the desired subsystems. Subsystem codes are:

- 0: No subsystems (useful to create a delay without waiting for stability)
- 1: Temperature
- 2: Field
- 4: Position (rotator)
- 8: Chamber (purge, high vacuum, etc.)

delay: Time in seconds to wait after stability is reached.

timeout: If stability is not reached within timeout (in seconds), the wait is abandoned. Set timeout to 0 in order to turn off this feature (i.e., to wait forever for stability).

Note: The WaitFor method is for use within WinWrap scripting only. In order to wait for stability in other programming environments, use one or more of the “Get” methods to poll status.