High Resistance Bridge Control (HRBC) and High Resistance Bridge Analysis (HRBA) software

Application Description & User’s Manual

Tim Lawson, 25/08/2017

1. **Introduction**

This software is intended as a direct replacement of the original TestPoint application *CCEM-K2\_2012 21 March 2015.tst*, and is written in Python 2.7, allowing it to run on newer platforms, such as Windows 7. Ultimately, it is intended to replace all TestPoint code with Python applications, whilst simultaneously building up a library of useful Python code / idioms, for future re-use.

Throughout the software and this manual ‘R1’ is used to refer to the circuit position of the unknown resistor (‘Rx’) and ‘R2’ is used to refer to the circuit position of the known resistance standard (‘Rs’).

**2. Requirements**

A version of Python (v 2.7 or greater) is required to run this software and can be downloaded from the [Python.org](https://www.python.org/downloads/) website. Alternatively, a complete Python development environment, such as [Python(x,y)](http://python-xy.github.io/) includes a Python interpreter and editor ([Spyder](https://github.com/spyder-ide/spyder)) plus many additional libraries including those required by this application (listed in the following section) . A development environment is recommended for developing and maintaining this software. The GMH temperature probes use functions contained in a dll, GMH3x32E.dll, included in this distribution (alternatively, look [here](file:///\\int.irl.cri.nz\data\MSL\Private\Electricity\Staff\TBL\Python\High_Res_Bridge\GMHdll)).

An Excel file is required to provide initial setup parameters for the instruments and to store the resulting measurement data. A template file is provided [here](file:///I:\MSL\Private\Electricity\Staff\TBL\GitHub\HRBC\HRBC_template.xlsx). Do not use the template directly -make a copy first!

**3. Modules**

Python makes accessible a huge number of function and class libraries for many purposes that can be imported into the source code. Many of these modules form a core part of Python, however, additional third-party modules may also be imported for specialised functionality. Several modules are employed in this application as described below:

* [wxPython](http://www.wxpython.org/) is a third-party graphical user interface (GUI) library allowing an application to be built that incorporates a number of ‘widgets’ such as buttons, data-entry / display fields and selection lists.
* [PyVisa](https://pypi.python.org/pypi/PyVISA) is a third party library that provides functions for communicating with external instruments via several different protocols including IEEE488 (GPIB) and RS232. It requires the VISA libraries that provide the low-level functionality and VISA protocol definitions, e.g. [NI-VISA runtime engine](http://www.ni.com/download/ni-visa-run-time-engine-15.0/5379/en/).
* [OpenPyxl](https://pypi.python.org/pypi/openpyxl/2.2.5) is a third-party library that provides a means to read, write and create Excel xlsx files. This allows HRBC to read configuration settings from a pre-existing spreadsheet and write results back into the same spreadsheet. The library operates on the Excel data in memory before saving the data to file. While this happens the file *cannot* be open simultaneously in another application (e.g. Excel). (NOTE: there is a bug/feature in Openpyxl that causes the saved Excel file to become corrupt if it contains any comments, therefore ensure all comments have been removed from the workbook prior to use!)
* [Matplotlib](http://www.matplotlib.org) is an extensive third-party plotting library that supports 2D (and 3D with additional add-ons) plots.
* [NumPy](http://www.numpy.org/) extends Python’s core mathematical functions. HRBC makes use of its functions for calculating the mean and standard deviation.
* [GTC](../../../../../../MST/GTC) is available as a Python module only for use within MSL; otherwise it must be run through the GTC application. HRBA makes use of GTC in its raw Python module form.

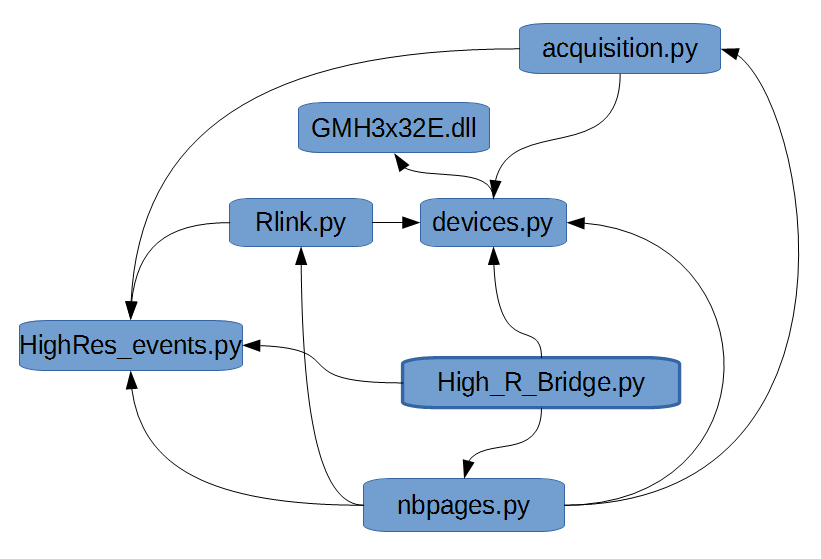
**4. HRBC**

*4.1 Source files*

The application is divided into several Python script files that separate the functionality:

* The core file is **High\_R\_Bridge.py** which defines the overall GUI ‘frame’ in which the application resides and starts the application event-loop. The application is started by opening and running this file in a Python interpreter (such as [Spyder](https://github.com/spyder-ide/spyder) or [SciTe](http://www.scintilla.org/SciTE.html)), in much the same way that a TestPoint application (XXX.tst) would be opened and run from within TestPoint.
* Within the application main frame is a notebook widget, consisting of three tab-selectable pages: ‘Setup’,’ Run’ and ‘Plots’ that are concerned with experiment setup (instrument assignment, etc.), running the measurement routines and live-plotting the resulting data. The code defining the contents and behaviour of these pages is in **nbpages.py**.
* **devices.py** handles information on all available instruments, obtained from the ‘Parameters’ sheet of the same Excel spreadsheet used to store measurement data. It also contains a reference to **GMH3x32E.dll** that contains several low-level functions used to read from and open / close connection to the GMH temperature probes. (Although the dll is written in C, Python provides the ctypes module to call foreign-language functions from within a Python script.) Much of the information is collected in a Python data structures called dictionaries that consist of a non-ordered collection of key-value pairs. The keys are used to reference the stored dictionary values. In this case, the keys are human-readable instrument description strings, such as ‘DVM: HP3458, s/n230’, which uniquely identify the instrument. Each dictionary value is a further dictionary consisting of a list of relevant information (as keys) for that instrument (such as its address) and command strings defining behaviour under specific conditions (e.g. instrument initiation, voltage or range setting…). Instrument information should be entered in the Excel file before running the application. In addition, there are also an *instrument* and a *GMH\_Sensor* class definitions, so that the above information can be captured within a single software object for each instrument. The class includes methods (functions) that set the address; cause the *instrument* (or *GMH\_Sensor*) object to be associated with a VISA resource (or physical GMH device), linked to the real physical instrument (Open() function); run the initiation commands (Init() function); close the VISA resource(Close() function) and send an arbitrary command, retrieving any response, (SendCmd() function). Each *instrument/ GMH\_Sensor* object has a *Demo* flag, which if set True causes many of the functions to do nothing, allowing instruments to be safely declared in software when their physical counterpart is not connected.
* Within a GUI application, *events* are the standard way to signal a change of state of a widget, allowing a handler function to take appropriate action. This mechanism is used to pass information between the measurement threads and the main GUI thread in a thread-safe manner. **HighRes\_events.py** contains definitions of user-defined event structures used within HRBC.
* When a long-running task is started within the main thread of a GUI application, the main event-loop is suspended until the long-running task completes, causing the GUI to become unresponsive. To avoid this effect, HRBC makes use of Python’s threading module to run long tasks in a separate thread of execution, running in parallel with the main GUI thread. There are two long-running tasks available: (i) Measure Rlink, (ii) Acquire Resistance Data (the main purpose of this application). Only one of these tasks should be run at once to avoid a conflict in accessing the same instrument at the same time with two different VISA resources. In any case, a different wiring configuration is required in each case. The actions of each of these tasks are defined within the files **RLink.py** and **Acquisition.py**. Each file contains a definition of a new thread class that includes the methods \_\_init\_\_() , run() and abort(). The \_\_init\_\_() method runs whenever a new instance of the class is created; the run() method runs whenever the start() method of the parent class (Thread) is called. When the abort() method is called it sets a flag that is periodically checked within the run() function (usually before a delay) that causes the run() function to make the sources safe, message the rest of the application concerning the change of status and exit early. When the code within the run() function has executed fully, some housekeeping is carried out to leave the instruments in a safe standby mode. At this point the ‘Start’ or ‘RLink’ buttons are re-enabled allowing a new run to be started. Any changes to the desired measurement sequence should be made within the run() methods of these classes.

Figure 1 illustrates the file dependencies:

**

*Figure 1. File dependencies.*

*4.2. User Interface*

Setup Page.

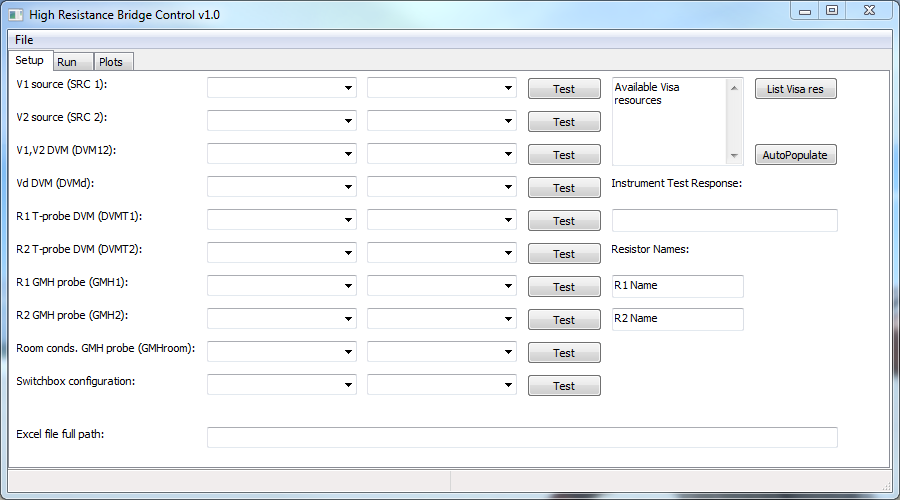
When the application is first run the ‘Setup’ page is displayed as shown in Figure 2. Before making any selections, select an Excel file by going to **File**>**Open**. This should be a copy of the [template file](../../../GitHub/HRBC/HRBC_template.xlsx), that has been modified, where necessary, to ensure all potential Rs and instrument parameters have been entered correctly. Ensure this file contains no comments (See note about Openpyxl in Section 3, above). Opening the Excel file causes all the instrument information on the ‘Parameters’ sheet to be imported to a dictionary named INSTR\_DATA. Instruments are then available for selection for each role (SRC1, SRC2, DVM12, etc.) in the first column of selection boxes as shown in Figure 2 and 3.

Figure 3 shows a partially-populated Setup Page. Note that the resistor names have changed to ‘CHANGE\_THIS!...’ to prompt the user to enter the correct information.

In addition to the two sources, four DVMs may also be specified:

* One for measuring the source voltages,
* One for measuring the null voltage and
* Another two for monitoring auxiliary temperature-probes on R1 and R2.

The second column of drop-down lists set the addresses of the instruments. Initially, there are no address options listed, however, by pressing the ‘List Visa res’ button, available visa resources are found and listed in the box to the left of the button. Once the resources have been found, they become available for selection in the drop-down ‘addresses’ lists – only GPIB addresses are listed for GPIB-controlled instruments. For RS232 devices, such as the switchbox and GMH probes, only available COM ports are listed. The ‘AutoPopulate’ button can be used to quickly fill-in a typical configuration. The default options specified by ‘AutoPopulate’ can be changed in the definition of the *OnAutoPop* function definition (file **nbpages.py**, lines 374-383).

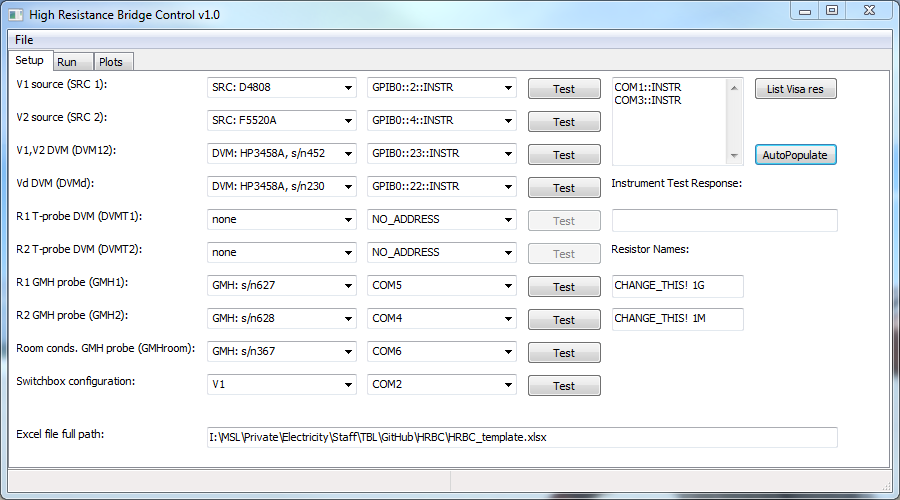


*Figure2. Setup page, prior to population.*

The next column consists of a series of ‘Test’ buttons for each instrument that check that the physical instrument responds to commands. Note that if any instrument description is ‘none’, the instrument is assumed not to be physically present and the corresponding ‘Test’ button is disabled. In the case of the GPIB instruments the standard command **\*IDN?** (or an equivalent alternative) is sent and the response appears in the ‘Instrument Test Response’ text field on the right of the Setup page. Each GMH probe function (usually temperature) is identified by a unique hardware address within the range 1-99. Occasionally the COM ports used for the GMH probes fail to be recognised by the program. The error can usually be resolved by unplugging and re-inserting the USB connectors leading to the GMH devices or freeing up and re-assigning the COM ports in Windows.

A quick way to determine which COM port the switchbox uses is to unplug the USB-serial adapter, rerun ‘List Visa Res’ and note which resource is no longer listed (or alternatively, check the ‘ports’ section in Windows Device Manager). To test communication with the switchbox, specify the switch configuration from the drop-down list (V1, Vd1, Vd2, V2) and then press ‘Test’ – success is indicated by the sound of the relays and the LCD display changing to the chosen configuration (the USB-RS232 interface will also flash).

The ‘Resistor Names’ fields identify the physical resistance standards to the operator and are used to construct a human- readable comment (see next section, **Run Page**). The resistor names should always take the form: **<*name*><*space*><value><*multiplier symbol*>**, e.g. ‘*Unknown 10M*’. This allows the ‘RLink’ code to extract the nominal resistance value and write it to the Excel sheet.

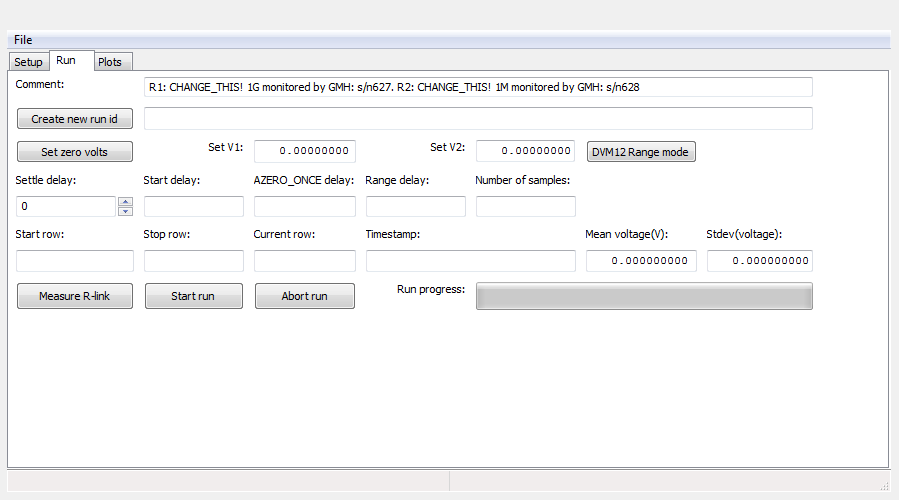


*Figure 3. Partially-populated Setup page, demonstrating a ‘Test’ result.*

Run Page.

On the Run Page an auto-generated comment is constructed from information entered on the Setup Page. Additional notes may be added to the end of the text string. Below this is a ‘Create Run Id’ button and associated text field. A run id should be created for each measurement (i.e. whenever the resistors are changed in the circuit) and is used to match Rlink data with the main measurement data. The next row of widgets are numeric-entry fields for setting or displaying the source voltages. The sources and their corresponding displays can be set to zero using the ‘Set zero volts’ button. On the right is a button that selects the range mode of the DVM monitoring the source voltages (‘DVM12’) – choices are Auto-Range (select range to match the measured voltage) or Fixed Range (fix range to suit the maximum voltage).

Information read from the Excel file ‘Data’ sheet (such as start row, stop row and delays) appears on the next row of displays once the ‘Start run’ button has been pressed. Prior to starting a measurement run an initial settle delay should be specified. The ‘Measure R-link’ button performs the measurement sequence required to determine the link resistance; all data is written to the ‘Rlink’ sheet of the Excel file. A progress bar shows the proportion of the measurement sequence that has been completed. At any time the measurement sequence can be aborted by pressing the ‘Abort run’ button. Once the ‘Start run’ button has been pressed, the ‘Timestamp:’, ‘Mean voltage(V):’ and ‘Stdev(voltage):’ displays are updated after every *n* measurements, where *n* is the number of samples. The ‘Current row:’ display shows which Excel row is currently being processed.

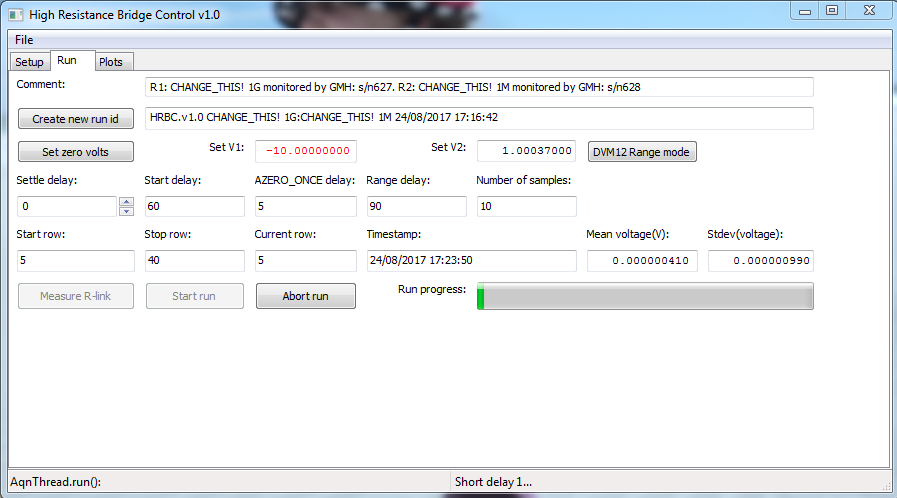
*Figure 4. Run Page before measurements have started.*

Note that whichever run type is in progress, the corresponding run button is disabled and only becomes enabled again at the end of the run or when the run has been aborted.

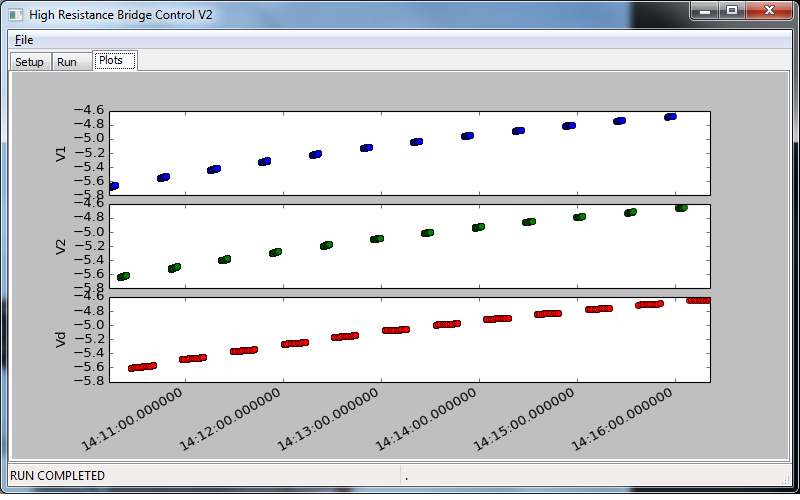
The status bar at the bottom of the application main frame displays messages indicating the current stage of the measurement sequence, for instance in Figure 5 the program execution has entered the run() method within the acquisition thread and is pausing prior to applying measurement settings for row 5.

Plot Page.

During the main resistance measurement sequence, every voltage measurement is plotted on one of three plots (V1, V2, Vd) on the Plot Page (see Figure 6). Resizing the window resizes the plots proportionately, allowing closer inspection of measurement trends. The plots are not used for the R-link measurement.

****

*Figure 5. Run Page during measurement sequence.*

**

*Figure 6. Plot of measurement results after completion of a run. Note that in this case only one DVM was in use.*

**4. HRBA (High Resistance Bridge Analysis)**

This program is also written in Python and makes use of the GTC library. It is intended to interact with the same Excel file as HRBC. HRBA is terminal-based and has no graphical user interface (GUI).

*4.1 Source Files.*

The code is split between two Python scripts:

* **HRBA.py** is the main script and should be run from a Python interpreter, such as Spyder or SciTE.
* **R\_Info.py** collects useful auxiliary functions that are called from the main script.

*4.2 Data Analysis Procedure.*

Prior to running HRBA, there should exist a spreadsheet containing data acquired by HRBC. The analysis proceeds through a block of data defined by the start and stop rows in the ‘Data’ sheet. A block of Rlink data corresponding to the main resistance data block (i.e. having the same Run id) should be present in the ‘RLink’ sheet. The analysis also requires resistance standard and instrument parameter values to be listed in the ‘Parameters’ sheet. The analysis results are written to the ‘Results’ sheet. The file must be closed before running HRBA, otherwise the code will be denied write-access.

The program automatically proceeds through the following steps:

* Open the Excel file, specified by command-line prompts.
* Read the start and stop rows from the ‘Data’ sheet.
* Read the instrument assignment block from the ‘Data’ sheet. This creates a mapping between the physical instruments and their roles.
* Extract resistor and instrument parameters from the ‘parameters’ sheet. Two Python dictionaries are created, I\_INFO and R\_INFO, holding information on, respectively, instruments keyed by their description (e.g. ‘DVM: HP3458A, s/n129’), and resistors also keyed by description (e.g. ‘SR104r 10k’).
* Where measurements are taken at two voltage levels, establish which voltages are ‘low’ and ‘high’. Results will be analysed at these two voltages separately and, ultimately, can be used to establish a rough estimate of the voltage coefficient.
* Housekeeping: Extract resistor names and nominal values from comment. Write column headings, Run id and comment on ‘Results’ sheet.
* Loop over measurements specified by start and stop rows in ‘Data’ sheet (each measurement spans four rows):
  + Assign reference resistor (Rs) parameters, such as calibration value, reference temperature, reference voltage, temperature and voltage coefficients, etc.
  + Select an appropriate value of voltage ratio correction (VRC) based on knowledge of instruments used and voltage settings.
  + Start building a list of influence quantities for calculating the uncertainty budget.
  + Calculate measurement temperatures for R1 (Rs) and R2 (Rx), based on GMH probes and auxiliary temperature sensors (via a DVM), if any. A zero-valued type B temperature influence variable is defined, having an uncertainty equal to half the difference between the mean GMH temperatures and mean DVM temperatures. If no auxiliary temperature information is available, the default temperature definition has uncertainty 0.01 °C.
  + Calculate mean voltages, including zero-valued type B drift terms.
  + Look up corresponding Rlink data and calculate Rlink.
  + Calculate R2, gain factor and thence R1.
  + Write this measurement result to the ‘Results’ sheet.
  + Calculate an uncertainty budget table for this measurement and output it to the ‘Results’ sheet.
  + Separate results by test voltage, if different.
* Perform a weighted least squares fit against temperature for all results at each test voltage to extract Resistance value (as a function of mean temperature). The mean of the slopes of the fits give a rough estimate of linear temperature coefficient α.
* Calculate linear voltage coefficient γ, using above results for the two different test voltages (if there is only one test voltage, no voltage coefficient calculation is performed).
* Finally, if R1 is a resistor that is not included in the 'current knowledge’, write all known information (value, calibration date, temperature coefficient, etc.) to the resistor section of the 'parameters' sheet for use during subsequent analysis runs.
* Save the workbook.

A limitation of the OpenPyxl module is that the saved workbook loses information on charts or other imported objects that were present in the file before HRBA operated on it. If these are to be retained the relevant parts of the workbook should be copied to a safe place before analysis.

It is intended that once a full measurement campaign has been analysed, the resulting Excel file contains a full record of the data capture and its analysis.