Bermondsey Electronics LTD

BELIeVE User Manual

1. TABLE OF CONTENTS

2. HISTORY

Full Name	Date	Describe	Version
Peter Wrigley	19/10/2022	Initial release	0.1
Peter Wrigley	25/10/2022	Add glossary	0.2
Peter Wrigley	09/02/2023	Correct link, typos	0.3
Peter Wrigley	23/02/2023	Add TCP tip	0.4

3. INTRODUCTION

The purpose of this document is to describe the internal workings of the BELIeVE application. It presents to the user a detailed walk-through of the project setup and test run process. This document describes the internal operations of that process such that users can get better value from the product.

4. AN IMPORTANT NOTE ON JINT

Under the hood, the application uses the open source Jint engine to execute ECMAScript on .NET. This has the benefit of .NET interoperability. The Jint site itself documents some of its features, such as the .NET types used for Javascript objects. At the time of writing, we use 2.11.58, which implements ECMAScript v5.1. The manual describing the language is freely available and is recommended reading. As of February 2023, the manual is at https://www.ecma-international.org/wp-content/uploads/ECMA-262_5.1_edition_june_2011.pdf.

5. PROJECT SETUP

BELIeVE projects are a collection of test scripts organised into test groups. Scripts can be run either individually or as groups. During test runs, data is logged to disk and to the debug console.

1. CREATE A PROJECT

On creation, a project file is an XML file on disk. Its contents can be edited by hand if preferred, as there is no special checksumming or encryption. By default, a couple of folders are created alongside as well.

2. PROJECT FOLDER STRUCTURE

The idea of the project folders is to contain project data (hex files, dynamic-link libraries [DLLs] etc.) in one place such that everything related to a given project can be version controlled using a single folder/repository.

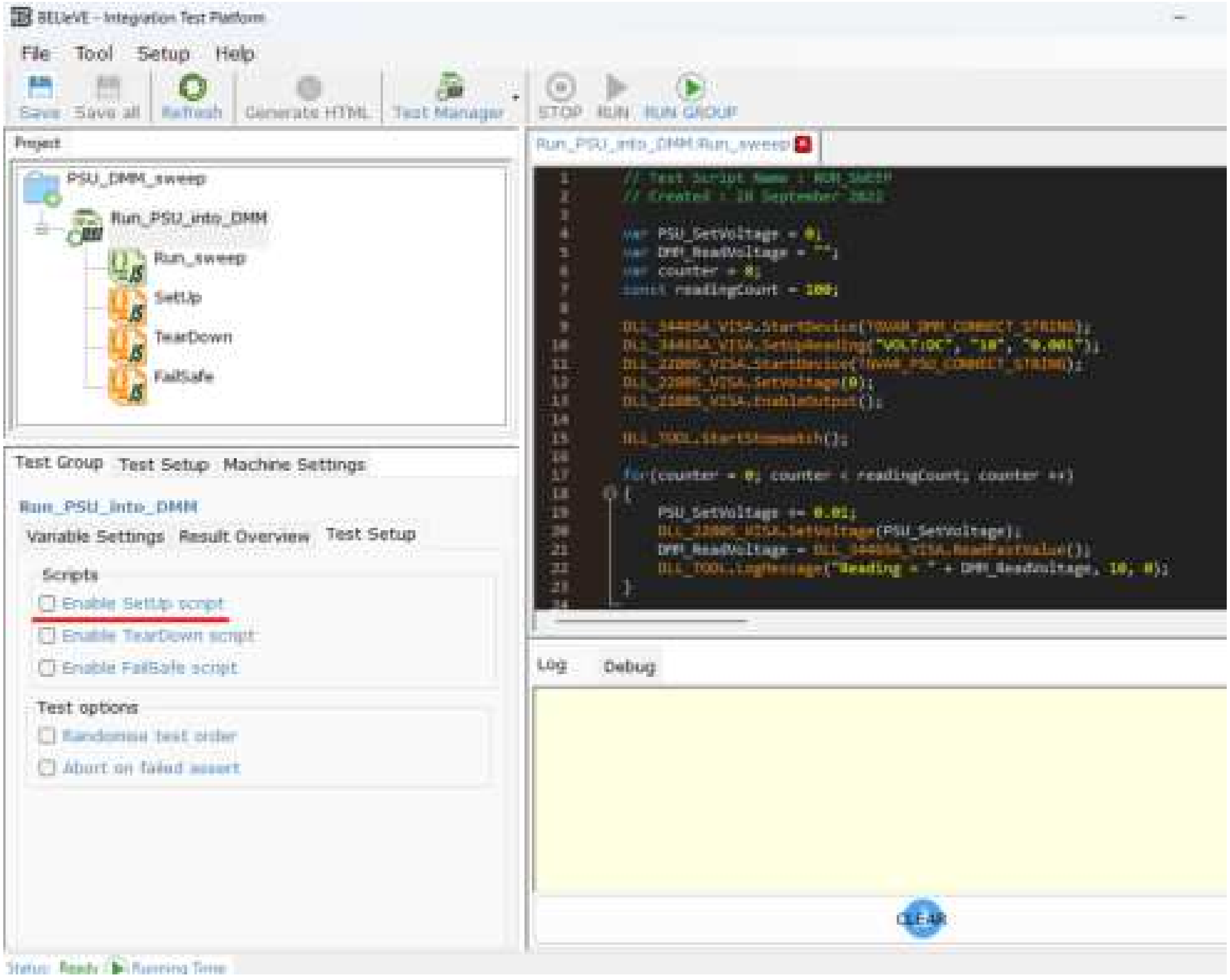
A 'Log' folder is created inside the 'Workspace' folder when the first test is run. This is a text file containing all text written to the "Log" and "Debug" windows of the main screen and is timestamped. This happens automatically at runtime without user intervention. File writes happen in a separate thread to the main Jint runtime, so the runtime overhead is minimal.

6. GROUPS AND TESTS

Tests are broken down into “Groups” with shared characteristics. Groups share “Test Group variables”. These are prefixed with TGVAR and are shown in autocomplete. Common variables used across a group might be a feature of the product to be activated, for example.

1. SETUP SCRIPTS

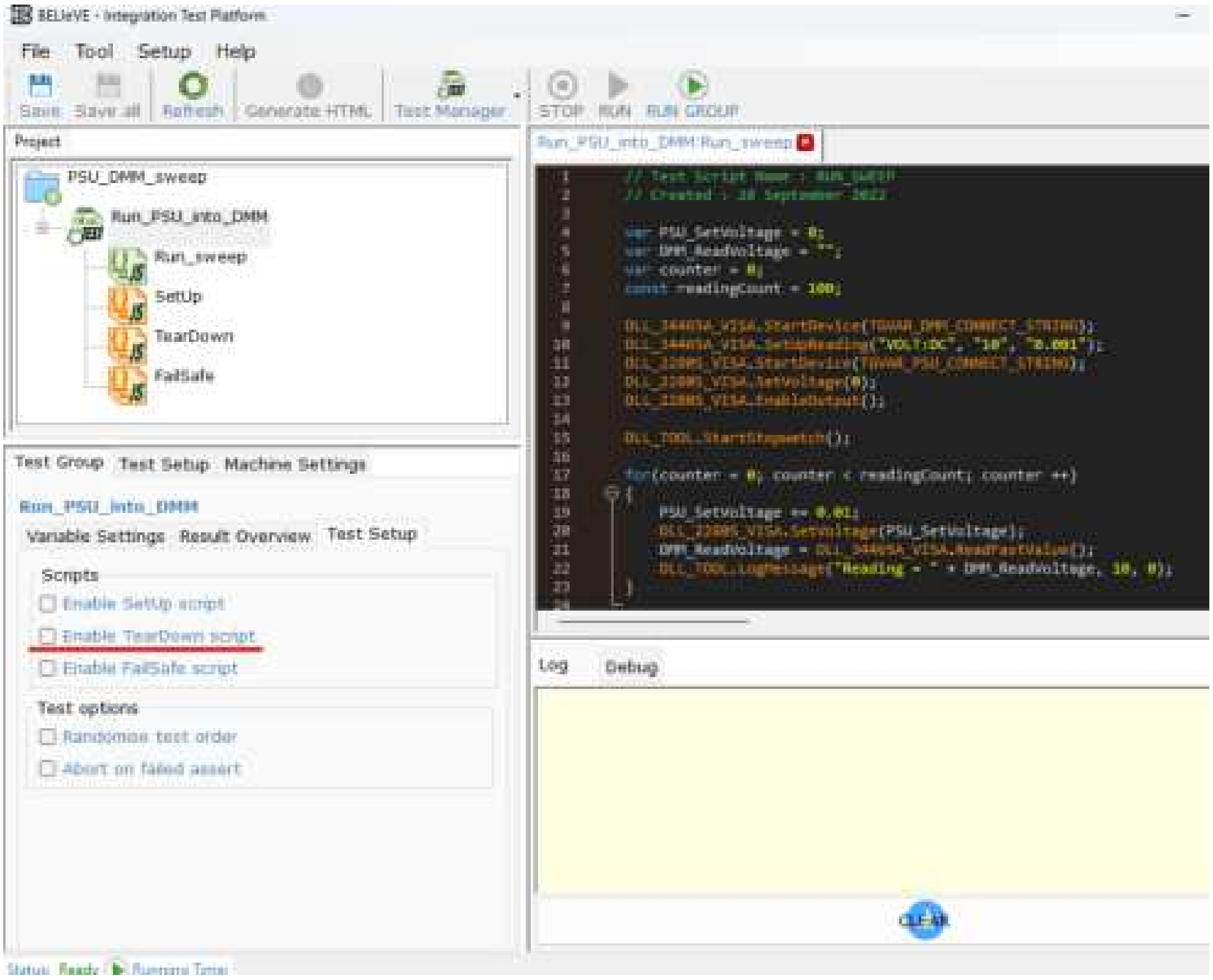
Each group can have a SetUp script associated with it. When enabled, the SetUp script is executed before each and every test script in a group.



```
// Test Script Name : RUN_SWEEP
// Created : 28 September 2022

var PSU_SetVoltage = 0;
var DMM_ReadVoltage = "";
var counter = 0;
const readingCount = 100;

DLL_34465A_VISA.StartDevice(TGVAR_DMM_CONNECT_STRING);
DLL_34465A_VISA.SetUpReading("VOLT:DC", "10", "0.001");
DLL_2280S_VISA.StartDevice(TGVAR_PSU_CONNECT_STRING);
DLL_2280S_VISA.SetVoltage(0);
DLL_2280S_VISA.EnableOutput();

DLL_TOOL.StartStopwatch();

for(counter = 0; counter < readingCount; counter ++)
{
    PSU_SetVoltage += 0.01;
    DLL_2280S_VISA.SetVoltage(PSU_SetVoltage);
    DMM_ReadVoltage = DLL_34465A_VISA.ReadFastValue();
    DLL_TOOL.LogMessage("Reading = " + DMM_ReadVoltage, 10, 0);
}
```

2. TEARDOWN SCRIPTS

Similarly, every test can run a TearDown script after execution, if one is activated. The combination of SetUp and TearDown scripts could be used to reset a target between scripts, for example. The TearDown script is not executed once at the end of the Group, it is executed after every script in a group.


BELIeVE - Integration Test Platform
File Tool Setup Help
Save Save all Refresh Generate HTML Test Manager STOP RUN RUN GROUP
Project
PSU_DMM_sweep
Run_PSU_into_DMM
Run_sweep
SetUp
TearDown
FailSafe
Test Group Test Setup Machine Settings
Run_PSU_into_DMM
Variable Settings Result Overview Test Setup
Run_PSU_into_DMM:Run_sweep
Log Debug
CLEAR
Status: Ready Running Time:

Scripts
- ☐ Enable SetUp script
- ☐ Enable TearDown script
- ☐ Enable FailSafe script

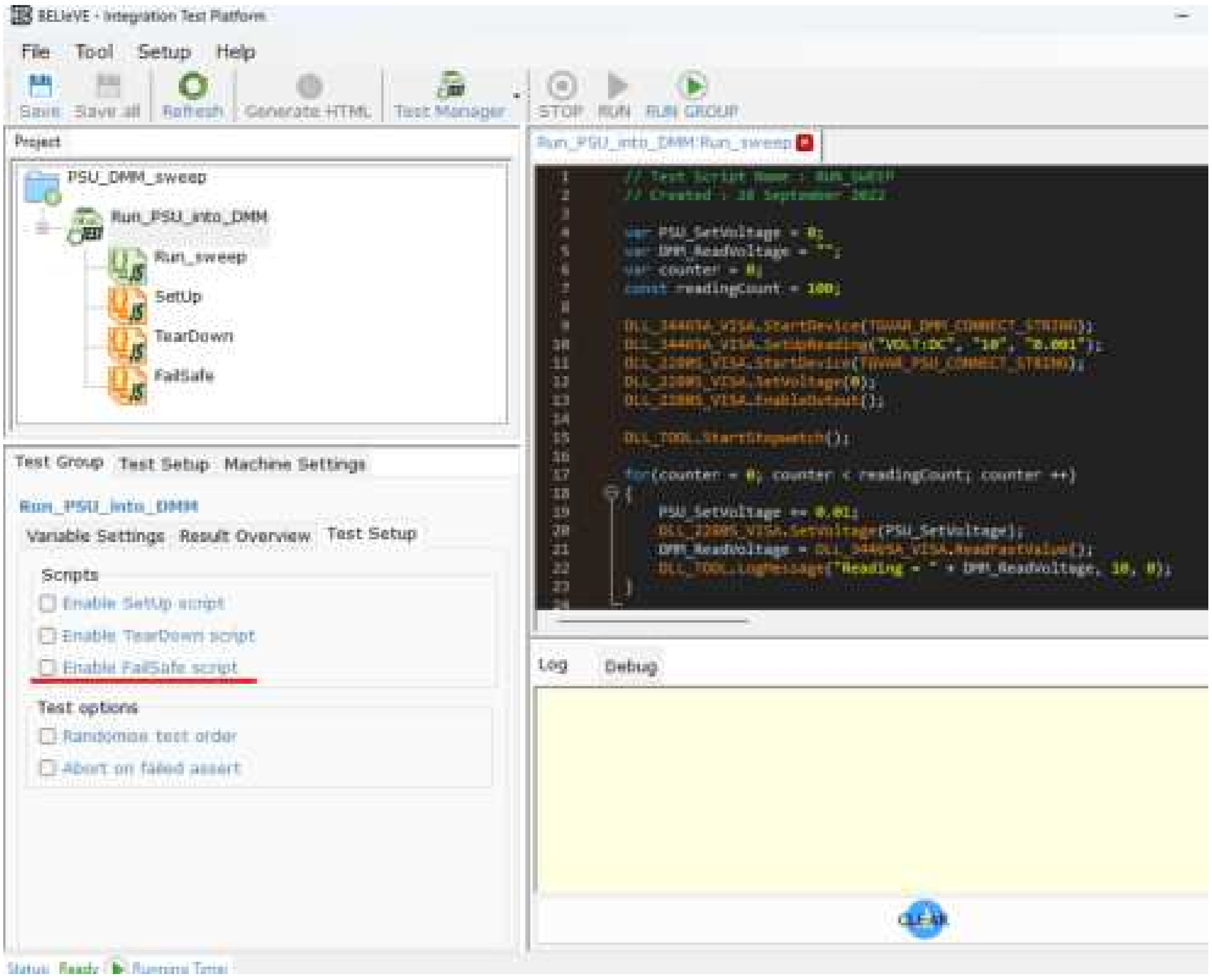
Test options
- ☐ Randomise test order
- ☐ Abort on failed assert

```
// Test Script Name : RUN_SWEEP
// Created : 28 September 2022

var PSU_SetVoltage = 0;
var DMM_ReadVoltage = "";
var counter = 0;
const readingCount = 100;

DLL_34465A_VISA.StartDevice(TDVAR_DMM_CONNECT_STRING);
DLL_34465A_VISA.SetupReading("VOLT:DC", "10", "0.001");
DLL_2280S_VISA.StartDevice(TDVAR_PSU_CONNECT_STRING);
DLL_2280S_VISA.SetVoltage(0);
DLL_2280S_VISA.EnableOutput();

DLL_TOOL.StartStopwatch();

for(counter = 0; counter < readingCount; counter ++)
{
    PSU_SetVoltage += 0.01;
    DLL_2280S_VISA.SetVoltage(PSU_SetVoltage);
    DMM_ReadVoltage = DLL_34465A_VISA.ReadFastValue();
    DLL_TOOL.LogMessage("Reading = " + DMM_ReadVoltage, 10, 0);
}
```

3. FAILSAFE SCRIPTS

If a script goes wrong, you may wish to take some protective action. For example, you might choose to remove power from the DUT. In the Test Setup view, there is a checkbox for "Enable FailSafe script". When this is checked, any script error will launch the FailSafe script (including SetUp and TearDown). Unhandled exceptions in a FailSafe script cause the FailSafe script itself to abort, potentially early.

Note this is linked to, but separate from, the "abort on fail" checkbox. Abort on fail will cause the FailSafe script to execute if the option is enabled but is expected to be used to skip running any further tests after a failure occurs.


BELIeVE - Integration Test Platform
File Tool Setup Help
Save Save all Refresh Generate HTML Test Manager STOP RUN RUN GROUP
Project
PSU_DMM_sweep
Run_PSU_into_DMM
Run_sweep
SetUp
TearDown
FailSafe
Run_PSU_into_DMM:Run_sweep
Log Debug
CLEAR
Status: Ready Running Time:

Test Group | Test Setup | Machine Settings

Run_PSU_into_DMM

Variable Settings | Result Overview | Test Setup

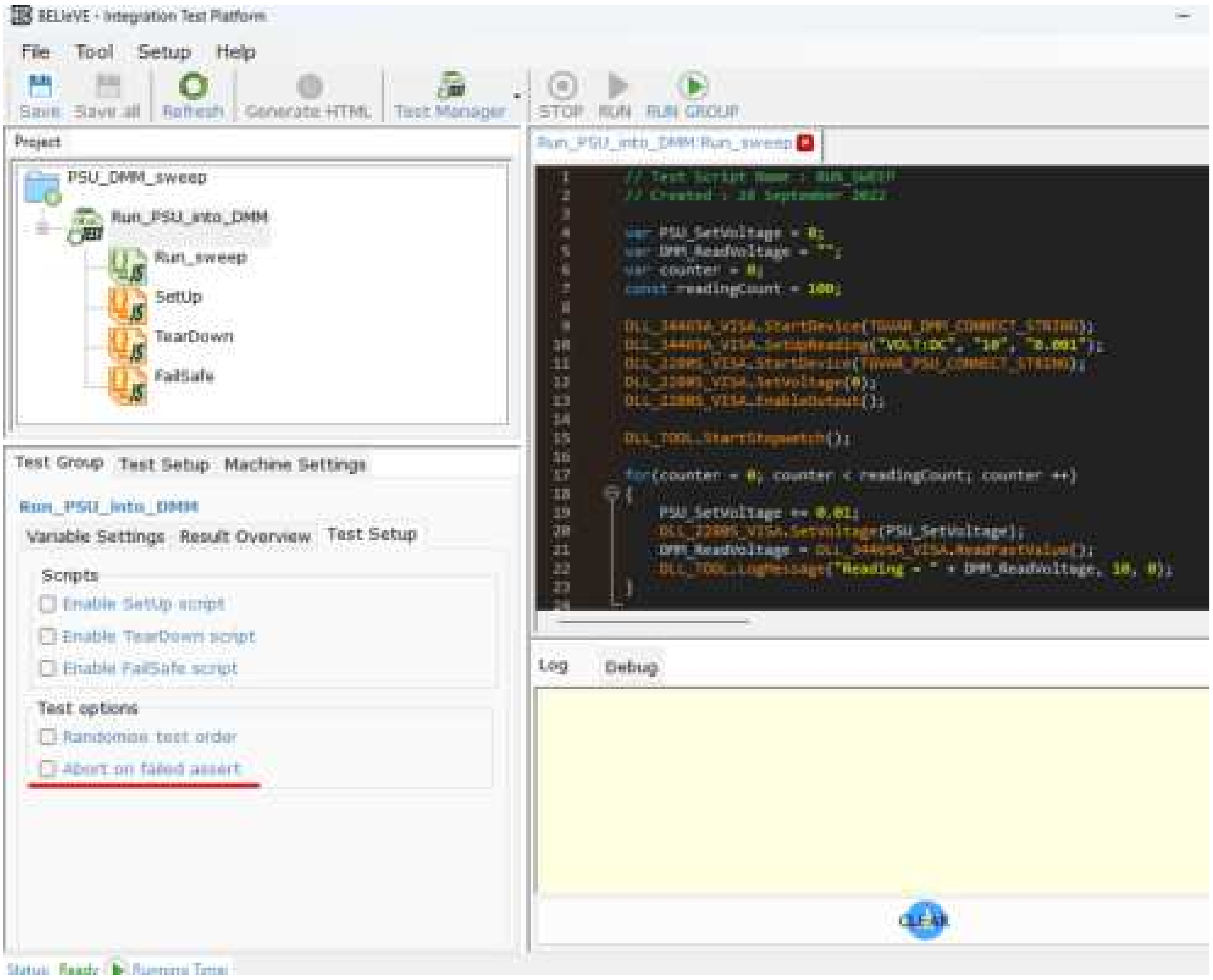
Scripts
- ☐ Enable SetUp script
- ☐ Enable TearDown script
- ☐ Enable FailSafe script

Test options
- ☐ Randomise test order
- ☐ Abort on failed assert

```
// Test Script Name : RUN_SWEEP
// Created : 28 September 2022

var PSU_SetVoltage = 0;
var DMM_ReadVoltage = "";
var counter = 0;
const readingCount = 100;

DLL_34465A_VISA.StartDevice(TSVAR_DMM_CONNECT_STRING);
DLL_34465A_VISA.SetupReading("VOLT:DC", "10", "0.001");
DLL_2200S_VISA.StartDevice(TSVAR_PSU_CONNECT_STRING);
DLL_2200S_VISA.SetVoltage(0);
DLL_2200S_VISA.EnableOutput();

DLL_TOOL.StartStopwatch();

for(counter = 0; counter < readingCount; counter ++)
{
    PSU_SetVoltage += 0.01;
    DLL_2200S_VISA.SetVoltage(PSU_SetVoltage);
    DMM_ReadVoltage = DLL_34465A_VISA.ReadFastValue();
    DLL_TOOL.LogMessage("Reading = " + DMM_ReadVoltage, 10, 0);
}
```

4. ABORT ON FAILURE

Sometimes you may choose to abandon test group execution should any test fail. You can do so by selecting the appropriate option in the UI. The FailSafe script will be executed, if enabled, and Project execution will end when any ASSERT fails. This is intended to speed up execution if you are only interested in seeing all tests pass, and not interested in continuing if any test fails. You might run this ahead of a release of the DUT, for example.


BELIeVE - Integration Test Platform
File Tool Setup Help
Save Save all Refresh Generate HTML Test Manager STOP RUN RUN GROUP
Project
PSU_DMM_sweep
Run_PSU_into_DMM
Run_sweep
SetUp
TearDown
FailSafe
Run_PSU_into_DMM:Run_sweep
Log Debug
CLEAR
Status: Ready Running Time:

```
// Test Script Name : RUN_SWEEP
// Created : 18 September 2022

var PSU_SetVoltage = 0;
var DMM_ReadVoltage = "";
var counter = 0;
const readingCount = 100;

DLL_34465A_VISA.StartDevice(TGVAR_DMM_CONNECT_STRING);
DLL_34465A_VISA.SetupReading("VOLT:DC", "10", "0.001");
DLL_2280S_VISA.StartDevice(TGVAR_PSU_CONNECT_STRING);
DLL_2280S_VISA.SetVoltage(0);
DLL_2280S_VISA.EnableOutput();

DLL_TOOL.StartStopwatch();

for(counter = 0; counter < readingCount; counter ++)
{
    PSU_SetVoltage += 0.01;
    DLL_2280S_VISA.SetVoltage(PSU_SetVoltage);
    DMM_ReadVoltage = DLL_34465A_VISA.ReadFastValue();
    DLL_TOOL.LogMessage("Reading = " + DMM_ReadVoltage, 10, 0);
}
```

Test Group | Test Setup | Machine Settings

Run_PSU_into_DMM

Variable Settings | Result Overview | Test Setup

Scripts
- ☐ Enable SetUp script
- ☐ Enable TearDown script
- ☐ Enable FailSafe script

Test options
- ☐ Randomise test order
- ☐ Abort on failed assert

5. SCRIPT EXECUTION IN DETAIL

Before running a script, associated DLLs are scanned for an Init() method (an empty one is included in the template). This is run before anything else happens, for all DLLs in use by the project regardless of whether or not those DLLs have methods invoked in the script being run. In other words, adding a DLL to the project will cause its Init() method to be run before every script in the Project.

After this ends, some Javascript is generated to create variables described in the UI (Group, Test and Machine variables). A Jint instance is created and runs this auto-generated code. The SetUp script, when used, is run next. The test script is then executed. Finally, the TearDown script is executed, when used, within the same Jint instance as the other scripts (so variables and instances are still accessible). If necessary, a FailSafe script may be run at any time (again, in the same Jint instance). When all this is complete, the Jint instance is disposed, then DLL Exit()

methods are called for every DLL required by the project. After this completes, the runtime timer in the UI is stopped. No particular order of DLL function call execution is guaranteed. These must be designed to run independently of external considerations.

All of this means every script runs in its own standalone instance, with no persistence. Note the environment the DUT experiences may or may not change during this time – it will depend on how often the device is reset or power cycled.

When executing a Test Group, scripts are ordinarily run in UI order from the top left hand ("Project") panel. Tests may be rearranged by drag-and-drop. In the "Test Setup" tab of the "Test Group" box, there is a "Randomise test order" checkbox, which will shuffle the tests randomly. When this option is enabled, the order of SetUp->Test->TearDown is not affected. Each script will be run once. The order of execution is recorded in the test log.

6. ASSERT THRESHOLDS

DUTs are manufactured to a tolerance, as is every piece of test and measurement equipment. ASSERTs in code need to deal with this tolerance. Ensure your ASSERTs set an appropriate tolerance for the test you are running. If measuring timing on the desktop, be aware the StopWatch methods supplied are tied to the OS system clock and its scheduler, and plan appropriately.

7. VARIABLES

1. TSTVAR

Individual test variables (TSTVARs) are defined in the UI / project file. Similar test scripts can be copied, pasted and customised with, for example, different firmware files or thresholds. Syntax highlighting will pick up these variables in the editor.

2. TGVAR

Test Group variables (TGVARs) apply to all tests within a group but otherwise behave identically to TSTVARs.

3. MSVAR

Machine-specific variables (MSVARs) are custom to each test runner. These might define things like which COM port to use to talk to a serial device, or where in the filesystem to find a particular file. New machines can be added via the UI, or manually in the project file. Internally, the application generates a fingerprint (which can be read programmatically using the DLL_TOOL.GetMachineFingerprint()

method). Each machine can also be given a friendly name. MSVARs must then be customised for each test runner. At test runtime, the machine fingerprint is compared with any defined MSVARs to define those variables for that runner.

8. DLLS

1. (ANOTHER) IMPORTANT NOTE ON JINT

Jint's interoperability exposes classes to the runtime. This means classes can be defined in an external DLL and accessed in-script using dot notation. These interfaces are not necessarily known to the syntax highlighter at test writing time, so autocomplete is not available. Jint will happily use methods or class members defined in a DLL, even if not highlighted by the UI. This allows passing objects similar to C structs between DLL methods and the test script – handy for exposing BLE Device Information Service members, for example.

2. THE GETINFOS() METHOD

To expose DLL members to the test editor, the GetInfos() method can be populated. A sample is installed in the User Documents folder. The GetInfos() method returns JSON describing the methods in the DLL. It is strongly recommended to follow the rough outline supplied. The test editor calls the GetInfos() method of the DLL to populate the autocomplete options, and to offer a popup box. You can, for example, use this to describe the methods in terms of parameters and return types.

3. ECMASCRIPT IS SINGLE THREADED, DLLS NEED NOT BE

Very importantly, ECMAScript strictly follows a single threaded execution model. This means scripts can only consider a single threaded environment. Interrupts and callbacks are not available within a script. They confuse the environment, and multi-threaded access to scripts will cause the environment to crash, often with an inscrutable error code and/or message.

DLLs run in a .NET environment and are not constrained by this. Should the user wish to launch new threads from a DLL method, no limitation is put on this. Users must be careful that scripts only access thread safe DLL methods. For instance, a DLL could spawn a thread to start communication with a DUT. Using cross threaded methods, return values from the DUT could populate a "GetComms" method, which can safely be polled from a Jint thread.

Scripts and the Jint engine execute as individual threads, spawned by the runtime on demand. Each test script executed is created in a new Jint instance and passed various scripts as described in earlier sections. This helps prevent scripts cross-polluting each other and provides a clean runtime every time a script is started.

Jint overhead is not significant. OS overheads are likely to be far higher than context switches or interpretation time.

4. INIT / EXIT METHODS

Init / Exit DLL methods should be designed to be run independently of any test. DLLs should be portable to allow code re-use. They are run before a script starts and after any FailSafe or TearDown script ends, and outside any Jint environment. No particular order of execution nor concurrency promises are made for Init and Exit methods of DLLs.

9. JS TIPS (FROM THE PERSPECTIVE OF A C DEVELOPER)

1. DYNAMIC TYPING IS NOT THE SAME AS UNTYPED

ECMAScript is dynamically typed. That means that often type conversion can be ignored. The exceptions then become the problem and are typically only seen at runtime. ECMAScript exposes a number of explicit type conversion methods which are of great convenience. ParseFloat will take a string and turn it into a double. ParseInt, ParseString and so on are all useful explicit type conversion methods whose liberal use is encouraged. Note the underlying Number type is double.

2. ALL THOSE ASSERTS ARE THERE FOR A REASON

The runtime may dynamically type your values, so the ASSERTs supplied force different types (in .NET Common Language Runtime terms) on the comparisons used. Jint’s base type for Number is double, which forces us down some interesting paths, so the ASSERTs are provided to disambiguate the process. Consider each typed ASSERT (e.g. ASSERT.IS_EQUAL_STRING) to carry out explicit type conversion before testing the ASSERT condition. It is not infallible, so the user can still cause errors by, for example, passing an instrument string value to a float ASSERT.

The untyped ASSERTs will treat Numbers as double but will throw runtime errors if an Object type is passed.

3. IVI/VISA/SCIPI TYPICALLY DEALS IN STRINGS

Strings like “5.1452E-06A”, for example. This is a real value returned by a DMM. To turn this into a mathematical value you can use in an ASSERT, for example, first you must strip out the ‘A’ at the end (for amps). You can use the ECMAScript substring method for this. Also note the string has a .length property you can use. When you have stripped out the units field, you can use the parseFloat method to return a Number.

Should you choose to implement a DLL, it is up to the writer to choose how to implement methods to get numbers. Type conversion can happen either in the DLL or in Jint, but regardless, be aware many instruments deal in strings, not numbers.

4. THREADING

The threading model in ECMAScript is strictly single threaded. Do not attempt to work around this; it ends badly. Tests must be designed for single threaded operation. Use DLLs to achieve multi-threaded operation.

10. INSTRUMENT TIPS

1. NOT ALL INSTRUMENTS ARE CREATED EQUAL

Depending on the TME, different string formatting might be used.

Depending on the internal design of the TME, a method to get its most recent value may or may not be updated if read too quickly after the last reading. Often this is not well signposted in documentation. It can be inferred from timing but the OS scheduler may interfere with precise timing of e.g. the DLL_TOOL.WaitMs method.

2. NOT ALL INSTRUMENTS IN THE SAME SERIES ARE CREATED EQUAL

We have witnessed wrinkles in communication performance between devices which externally appear identical.

3. VENDORS LIKE TO IMPLEMENT VENDOR-SPECIFIC STUFF

Fair enough and we do not blame them. Read the small print. The underlying protocols may not have been designed for file transfer, so when transferring files using a vendor-specific workaround, exercise care and respect the protocol definition.

4. TRY THE OTHER COMMS INTERFACES TO YOUR TEST & MEASUREMENT KIT

Depends on the specific device, but TCP/IP can be lower latency and/or higher bandwidth than USB, particularly USB through a hub. It is worth exploring the possibilities of VPN. We have used the application over VPN and while there is an increase in latency (roughly the ping time), there are times when this is less important than being able to run a test remotely.

5. EVEN LOW-END TEST GEAR CAN BE USEFUL

If you are manually putting values into a PSU today then you know how long it takes. A low-cost modern bench DMM may return say 4-10 readings per second. This is still massively quicker than taking readings by hand. This means your potentially-obsolete older TME may still be able to serve a useful purpose. If you need high end accuracy, or high throughput, you may have to acquire the TME to reflect the requirement (we offer no shortcut to high accuracy and high sample rates). By testing automatically, not manually, a test run can be in progress while other tasks happen. This increases development efficiency – the developer no longer needs to pause for hours to run a release test and can work on other tasks in the meantime.

6. REMEMBER TO CLOSE YOUR TCP CONNECTIONS

If using TCP/IP to connect to TME, there is a separate state machine running on the TME. Just because the test has completed does not mean this state machine has ended. If the connection is not gracefully closed, it may end up timing out on the target. Limited memory in TME often means they support exactly one connection at a time, so if your script does not gracefully close the connection, you may have to wait for the internal state machine on the TME to time out before you can connect again. Make sure your TCP/IP sessions end gracefully.

11. GENERAL TIPS

1. FORGET PRECISE TIMING ON THE DESKTOP

The OS scheduler can be slept in units of milliseconds. In reality, if another process blocks the CPU there can be jitter of double digit milliseconds. Assume the desktop runtime is good for about 30ms accuracy and there will be few problems. For better timing accuracy, you will require offloaded TME. We often use development kits running a simple serial protocol to time PWM, for example. Our first ever dedicated test station was/is a second hand NUC8i3 with 8GB RAM and a SATA SSD. It is typically 10ms jitter, or up to 100ms if it is writing a file and has to erase a block.

2. YOUR FAVOURITE EMBEDDED DEV KIT IS REALLY USEFUL

Do you need to press some buttons? Hook up a dev kit to some optocouplers and write a trivial serial protocol. Now you can close a circuit to whatever timing accuracy (and synchronisation with external signals) you see fit. Give it a list feature to step through different buttons with different timing. Go wild. It's the best way to achieve sub-10msec timing accuracy without buying dedicated bench equipment. Arduino provides a controlled hardware platform and pinout supporting a number of different peripherals. Many vendors use this pinout – so pick your favourite and use it.

3. SOMETIMES YOU NEED A BACK CHANNEL TO YOUR DUT

It is very useful to be able to mock function return values or invoke specific behaviour in your DUT. It can often be best to do this out-of-band through a communication channel the final product will not use, to decouple it from other activity. The channel must be removed before production. Typical back-channel commands including changing system time, invoking features, running a dedicated debug script on the target, etc. A shim layer running on your target can intercept mocked function calls and plug in your desired mocked values.

Segger RTT is useful on Cortex-M targets.

4. TEST YOUR ERROR HANDLERS

Untested error handlers can cause serious production defects. Use the system to test them. Create error states in your DUT and ensure the vendor library responds appropriately. Use field error reports to increase your test suite.

5. YOU CAN TEST OVERNIGHT

Faced with a shortage of TME or DUT PCBs, you can run automated tests at times to suit availability of the equipment.

6. LET THE RANDOMISER FIND THINGS

It can be tempting to reset the DUT between every test, or between test groups, but this is rarely the best outcome. The randomiser causes the DUT state to be retained between tests. Unless the test includes an explicit requirement to test a power cycle, it is recommended not to power down the DUT between tests. This helps to weed out “onceability” in the DUT.

7. BIG LOG FILES CAN BE FIDDLY

A multi-GB log file sounds tempting but is not easily opened or processed. Text editors and CSV viewers will have trouble (even those designed for large data sets will at best be slow). In general, it is preferable to log as little as necessary to the log file. Modern SSDs mean rewriting large log files is no longer the error source it was in the days of HDDs but that does not make it desirable or necessary. The console window can be an alternative but is not always practical. Wherever possible, it is easier to break down a big log into smaller files (e.g. time, size).

1. GLOSSARY

DUT	Device Under Test
BLE	Bluetooth Low Energy
SSD	Solid State Disk
HDD	Hard Disk Drive
TME	Test and Measurement Equipment
DLL	Dynamically Linked Library
PWM	Pulse Width Modulation
PCB	Printed Circuit Board