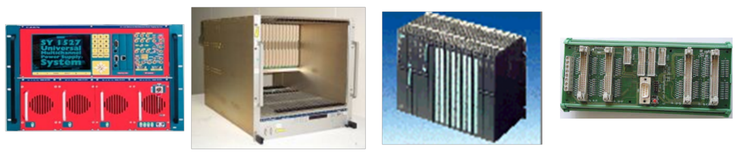
Automated Testing of OPC Servers

Author : B. Farnham

OPC at CERN

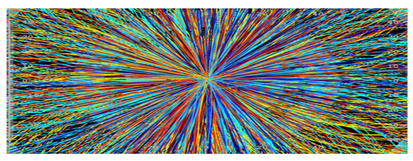
OPC is used at CERN to provide a standard software interface for the control and monitoring of various devices front end devices including industrial power supplies, VME crates, PLCs and ELMBs.



Some OPC servers are custom implementations driving custom devices such as the ELMB, other OPC servers are provided external vendors.

Regression testing and OPC

The experiments have working systems. Any change carries the risk of inadvertently degrading these systems.



Regression testing aims to minimize this problem by 'capturing" critical behaviour in tests and running these against new versions OPC is an industry standard for control and monitoring. It provides a common vocabulary for testing critical behaviour that is the same for every vendor.

Automated testing

Regression testing is a boring job for a human. Verifying critical functionality across multiple versions entails checking that the same input sequence produces the same output sequence. Time after time after time. An OPC client can be made to programmatically issue a defined sequence of commands to an OPC server. The effect of these commands can be observed via an OPC client inspecting the correct OPC items exposed by the server. This is the basis of the application, scripts direct a machine to issue OPC commands to an OPC Server and asserts the observable results.

An OPC testing DSL

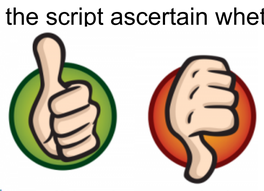
Domain Specific Languages are small languages, specially tailored to suit a given domain. OPC test scripts are written in a DSL specially created for OPC testing. The language consists of nouns (e.g. groups and items) and verbs (e.g. writes and assertions) to more clearly express and communicate the scripts intent. The basic idea is that anyone with domain knowledge (i.e. OPC plus the equipment it controls) can read and understand what the script does in terms of instructing the OPC Server to drive the device and what the script expects to happen as a result.

Runnable bug reports

A common problem in complex systems is the non-repeatable bug: Users describe a bug to the vendor who tries, and fails, to recreate it in a debug rig based on their interpretation of the description.
Using the Automated Tester, bugs can be described in a test script - with failing assertions highlighting errant behaviour. The same script is sent to the vendor who runs it against their equipment in a debug rig. No interpretation is required. Often scripts include a pre-test setup section. This section makes assertions about the environment (e.g equipment and state) to ensure that the script is being executed under appropriate conditions.

Pass or fail: Assertions

Assertions are the means by which tests are deemed to have either passed or failed. A test script contains a series of commands in order to have an effect on the system, assertions in the script ascertain whether the effects were correct or otherwise.

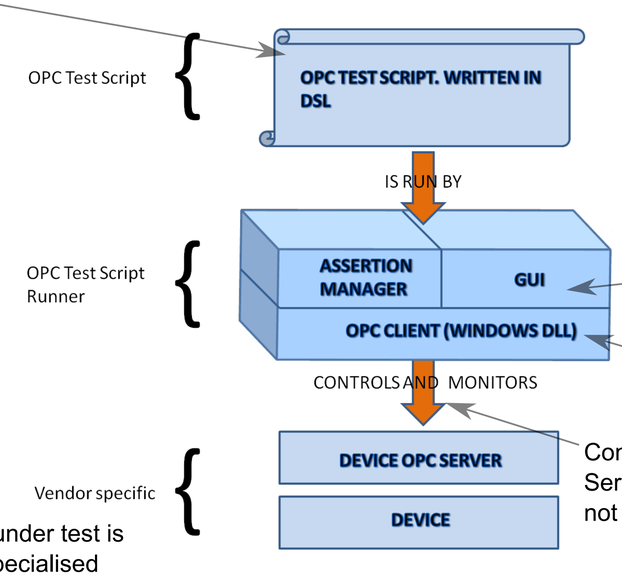


The DSL currently supports 2 assertion types: Synchronous assertions are immediately evaluated to return a pass or a fail; Asynchronous assertions are evaluated over a specified period, if the condition is met within that period, the assertion passes, otherwise it fails.

The Automated tester: An exploded view

An example test script used for testing a CAEN power supply

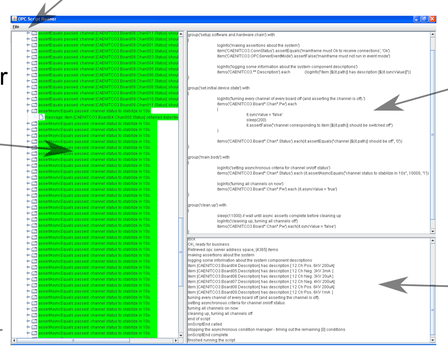
```
logInfo('script started...')
init('', 'CAEN.HVOPCServer')

group('setup.software.and.hardware.chain').with
{
 logInfo('making assertions about the system')
 item('CAENITCO3.ConnStatus').assertEquals('mainframe must Ok to receive connections', 'Ok')
 item('CAENITCO3.OPCServerEventMode').assertFalse('mainframe must not run in event mode')

 logInfo('logging some information about the system component descriptions')
 items('CAENITCO3.**.Description').each {logInfo("item [${it.path}] has description [${it.syncValue}]")}
}

group('set.initial.device.state').with
{
 logInfo('turning every channel of every board off (and asserting the channel is off).')
 items('CAENITCO3.Board*.Chan*.Pw').each
 {
  it.syncValue = 'false'
  sleep(200)
  it.assertFalse("channel corresponding to item [${it.path}] should be switched off")
 }

 items('CAENITCO3.Board*.Chan*.Status').each{it.assertEquals("channel [${it.path}] should be off", '0')}
}

group('main.body').with
{
 logInfo('setting asynchronous criteria for channel on/off status')
 items('CAENITCO3.Board*.Chan*.Status').each {it.assertAsyncEquals("channel status to stabilize in 10s", 10000, '1')}

 logInfo('turning all channels on now')
 items('CAENITCO3.Board*.Chan*.Pw').each {it.asyncValue = 'true'}
}

group('clean.up').with
{
 sleep(11000) // wait until async asserts complete before cleaning up
 logInfo('cleaning up, turning all channels off')
 items('CAENITCO3.Board*.Chan*.Pw').each{it.syncValue = 'false'}
}

logInfo('end of script')
```



Note scripts are device specific. A scripts view of the device under test is through its OPC Server which publishes an address space specialised to the device.

The results tree.Nodes are dynamically added to the tree as assertions are made in the script. The tree updates the node colour as assertions pass, fail or (in the case of asynchronous assertions) are undetermined

Simple user controls including option to output script's assertion results in junit style XML report



The script currently running (read only).

The output of user specified log commands, updated as the script executes.

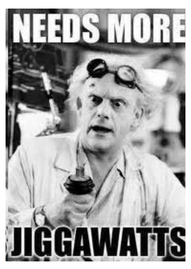
The OPC (OPC-DA) client is encapsulated in a windows DLL, written in C++..
The script execution layer on a JVM passes OPC commands (create group, read/write item values etc) over the interface exposed by the DLL through Java Native Access (JNA).

Communications between the OPC client embedded in the OPC Automated Tester and the OPC Server is pure OPC. The Automated tester implementation is dependent only on the OPC standard, not the device or OPC Server under test.

Current use cases

A regression test library is under construction for industrial power supply (and VME crate) vendors CAEN, ISEG and Wiener. In the case of one vendor these regression test scripts have already been put to use to test new OPC Server versions and power supply mainframe firmware.
Another vendor's OPC Server is suffering from a memory leak, multiple scripts have been written, each focusing on a different type of OPC interaction, e.g. item reads, item writes or group creation etc, and run in conjunction with the Windows perfmon utility to check which interactions cause the highest memory consumption. The scripts have been sent to the vendor to aid diagnosis.
A proposed case (under construction) is to incorporate regression test scripts into CERN's release and verification process for the firmware of individual power supply modules.

Future work



Wider coverage for regression tests. Currently only a small fraction of functionality is covered by scripts. The coverage should be increased over time to reduce the areas in which regressions could go unnoticed.
A **batch mode** for running and reporting on a batch of tests without human intervention. Currently each test runs automatically, however, human intervention is still required between each script to record the results and load and start the next test. A batch mode would enable users to define and run a test playlist.
Methods, tests are currently defined and run as a single continuous sequence. The ability to define parameterised methods has the potential to make scripts more expressive.
OPC Unified Architecture. CERN's current OPC implementations are OPC-DA, which is being superseded by the OPC-UA specification. CERN has been evaluating this specification and OPC-UA implementations are expected in the next few years. OPC-UA support will include extending the DSL to encapsulate new features in the specification, handling OPC-UA security measures and a cross platform version.

JACOW ID WEPMS006

Author : B. Farnham

10 - 14 October 2011

CERN EN Engineering Department