Title	**Memo automated regression tests for the release of the LORC controller**	
File:	sw_test.pdf	
Rev:	0.0	
Date:	2017-03-28	
Authors:	William A Coolidge	

Purpose

The purpose of this document is to define the architecture and bound the development effort for the capability to perform automated regression tests on the LORC controller.

Concept

A large number of the pieces required to build a sw regression tester are now in place such as the testFramework, the scripting build ecosystem, document handling, and server runtimes thanks to the maturity and completeness of the components in the sw development community, and developments at R&D AS. The ADS interface to the PLC has been developed in another context, the server runtimes, scripting, and document handling have been developed in other contexts. The original effort that remains on the framework is in the development of the testLibrary based on standard components and the integration effort. The main original effort is in the targeted domain, i.e. in the development of the test procedures themselves.

The development of test procedures for the controller needs to be done in any case and this effort ensures that tests are defined interms of documents that can be developed and maintained by a non programmer.

Test procedures will define the operations required to bring the controller into the stated desired for the start of the test. This state will be validated by the test procedure and a series of time based stimuli will then be applied to simulate the asynchronous inputs to the controller. The test procedure follows with time based assertions to determine is the expected results of the test were achieved. The log of the test procedure is logged to create a test report. A test plan consists of a sequence of test procedures. A regression test consists of the sum of all test plans. A scripting environment controls the execution of the regression test.

The LORC controller is designed with testability in mind by its use of function arguments that carry all state, a design which lends itself to test instrument-

ation.

Goals

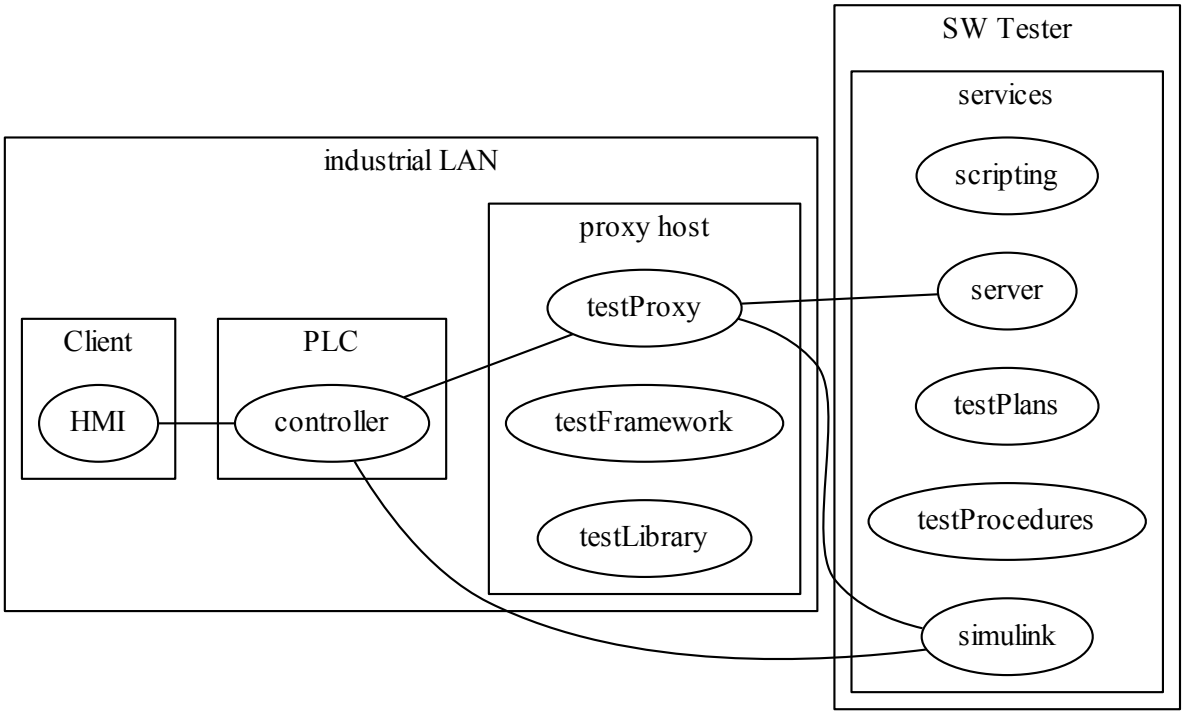
The goal of the effort is to:

1. To have the ability to test the controller and make the tested code coverage visible
2. To have a set of reference test cases to validate the design and implementation
3. To have the ability to automate regression testing of changes to the code base
4. Provide the ability to discover both design and programming errors early
5. Provide the ability to show development progress and a have basis for release management
6. To have a reference test for use in post release maintenance
7. Use document based test procedures so that non-programmers can create, use, or review test cases
8. Provide automated scripting to remove human time from test time
9. Provide a test environment for both local testing by individual developers as well as shared release testing

System

The system is illustrated and described in the following diagram and tables respectively.

Client	The scope of this tester does not include testing of client code pre se. However, the tester does act as a client so it does act as a refererence test of the controller for HMI interactions.
PLC	The PLC is the test target. The test interface is ADS with simulink using the UDP protocol from the Demonstrator project
proxy host	A Windows 7 host shall be provided that is separate from the IPC of the host and on the industrial LAN with a static IP. This requirement is due to the limitations of Beckhoff's router. The same proxy host that is used for DAQ will be re-used for the test proxy. The ADS library from the DAQ will be re-used in the testProxy.



testProxy	The testProxy will use the testFramework and testLibrary to communicate with the controller and run the test procedures according to the scripting service (Gulp) of the SW Tester. The testProxy will read the test plan and perform the individual operations according to the time sequencing of the test. The testFramework provides a result that is logged.
testFramework	Mocha is the test framework that will be used as is along with the Chai assertions library to build the routines of the testLibary
testLibrary	A small R&D specific library will include only generic functions for the sourcing and assertions on ADS values based on Mocha and Chai. These generic test functions will be parametrized by the JSON objects (text files) that define the test procedures. Rather than creating a large library of functions for testing the targeted controller, a few generic functions will be used for sourcing and assertions that are parametrized by test data in the JSON file that defines a test procedure. The ADS library from the DAQ will be re-used in the testLibary.

SW Tester	This is hosted on a R&D server to testing of each change to the git repository as the basis for the release of the controller. It can also be hosted on a developer's machine for developer specific testing.
scripting	All build scripting is based on Gulp. Integration with git and task automation such as nightly testing will be based on tasks defined in gulp
server	The test server's role is to provide the testProxy with testPlans and testProcedures
testProcedures	The test procedures will be defined by JSON files. The name value pairs that define the JSON objects will parsed for use under the Mocha/Chai test framework. A test procedure will include a number of source or assertion operations that are sequenced according to a relative time sequencing to allow for expected settling times, system delays etc.
testPlan	The test procedures will be defined by JSON files. The name value pairs that define the JSON objects will parsed for use under the Mocha/Chai test framework.
simulink	A second level of controller code coverage is achieved by using simulink to simulate the plant. This is a better alternaive than doing a pure emulation of the plant in sw to increase code coverage of the controller. In this case, the test procedure will include setting simulink in the appropriate state. Specific error injection scenarios will need to be instrumented in simulink such as hydraulic pressure loss will be used to test critical code paths.