# NIF ICCS TEST CONTROLLER FOR AUTOMATED & MANUAL TESTING*

J. Zielinski#, LLNL, Livermore, CA 94551, U.S.A.

*Abstract*

The National Ignition Facility (NIF) Integrated Computer Control System (ICCS) is a large (1.5 MSLOC), hierarchical, distributed system that controls all aspects of the NIF laser [1]. The ICCS team delivers software updates to the NIF facility throughout the year to support shot operations and commissioning activities. In 2006, there were 48 releases of ICCS: 29 full releases, 19 patches. To ensure the quality of each delivery, thousands of manual and automated tests are performed using the ICCS TestController test infrastructure. The TestController system provides test inventory management, test planning, automated test execution and manual test logging, release testing summaries and test results search, all through a web browser interface. Automated tests include command line based frameworks server tests and Graphical User Interface (GUI) based Java tests. Manual tests are presented as a checklist-style web form to be completed by the tester. The results of all tests, automated and manual, are kept in a common repository that provides data to dynamic status reports. As part of the 3-stage ICCS release testing strategy, the TestController system helps plan, evaluate and track the readiness of each release to the NIF facility.

## INTRODUCTION

The ICCS team executes nearly 7,000 automated and manual regression tests against the control system before delivering a major release. Tests are executed during each of the three test phases of the ICCS development lifecycle [2, 3]:

- Developer integration testing
- Offline regression testing
- Online regression testing in the NIF

While manual tests are gradually being converted to automated tests, the majority of testing is performed manually by the ICCS development staff and Verification and Validation team. A web-based test infrastructure, the ICCS TestController, was created to help manage the entire testing effort. This paper highlights the key features of the TestController.

## CORE COMPONENTS

Running on a central server, there are three core components that form the structure of the TestController:

- Database
- Custom server daemon processes (written in PERL)
- Web server running PERL Common Gateway Interface (CGI) scripts

The custom daemon processes running on the server include the following five managers:

- **Profile Manager** – This is the interface for the retrieval and update of user profiles. Each profile contains the automated test scenarios executed by that user in the past, the set of machines available and a set of user preferences used by the web interface.
- **Execution Manager** – Orchestrates automated test execution scenarios from start to finish.
- **Status Manager** – The Status Manager satisfies all queries for automated test execution status.
- **Results Manager** – This is the interface for the storage and retrieval of all test results.
- **Test Inventory Manager** – Test inventories are important for both manual and automated testing. Inventories are updated and accessed through the Test Inventory Manager.

To support the remote execution of automated tests, a multi-platform daemon called the TestController Agent runs on test lab machines in order to service requests from the central TestController. The Agent will be discussed in further detail later.

## TEST INVENTORY MANAGEMENT

Test inventories can be viewed and updated through the web interface. Though the inventories are stored as XML files, they are passed through the user interface as Excel spreadsheets to simplify editing. Users can download a file from the current inventory, make changes to it, then upload the modified file. The uploaded file is compared to the current inventory and the differences are displayed to the user. Once satisfied with the changes, the user saves the new inventory.

The team's automated Java tests are documented using the javadoc mechanism. Instead of requiring the developers of these tests to also update the inventory through the Test Inventory Manager, the javadocs are automatically parsed and converted to the XML format.

For all automated tests, the test inventories are used by the web interface when a user is assembling an automated test run. The user selects a test suite and then chooses any subset of the available tests within that suite. The list of suites and available tests come from the set of test inventories.

The test inventories also serve as the basis for test planning, as described in the following section.

* This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

# zielinsk2@llnl.gov

## TEST PLANNING

Each release of the ICCS software has a unique test plan. The plan is updated by members of the development and test teams so that the testing is appropriate for the content of the particular release. Test plans for small patch releases may include only a few tests from the one or two areas that have been changed, while plans for major releases will include thousands of tests that cover the entire system. A release test plan has a hierarchical structure as shown in Figure 1.

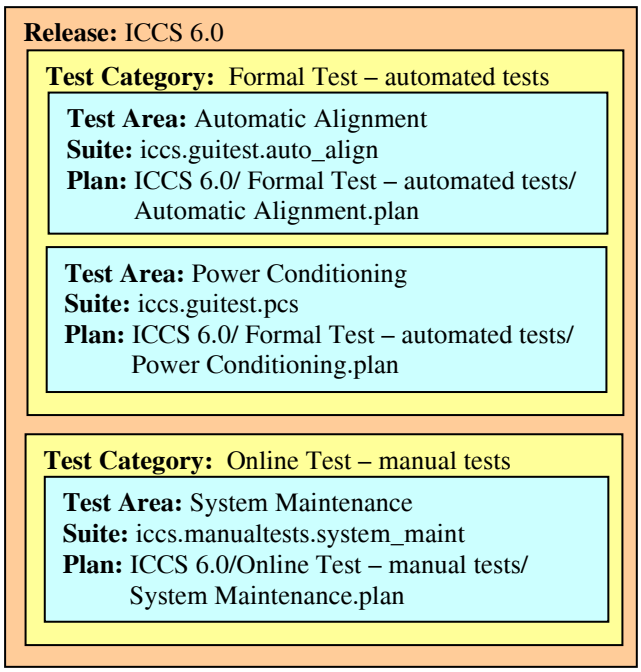


Figure 1: The hierarchical structure of a release test plan. The plan contains any number of *test categories*, each with any number of *test areas*.

Adding or removing tests from the plan is done using the test planning web interface. Once the release and test area are selected, the available tests (from the test inventory) are presented in a selectable tree structure. Only the selected tests are included in the plan for that release.

The release test plan is used to generate the manual test checklists discussed in the following section as well as the release testing summary reports discussed later.

## MANUAL TEST RESULTS ENTRY

For manually executed tests, a checklist-style web form is generated automatically from the test plan. From the form, the user performing the tests sets the disposition for each item and optionally logs any notes relevant to that test. If the note text contains a reference to an issue from our issue tracking system, a direct link to that issue is dynamically generated and placed alongside the note in the form. When the form is submitted, the results are stored in the test results database along with the product version, a current timestamp and the user's name. When a manual test checklist is retrieved, any tests that have already been performed will show the current status as well as the timestamp and user information.

## AUTOMATED TEST EXECUTION

The TestController includes a complete framework and web interface for the execution of automated tests. The user assembles a test run *task list* by selecting which tests to execute and the machines to execute on. The task list is then submitted to the Execution Manager daemon. The Execution Manager is responsible for coordinating the test run, tracking the status while it's running and ensuring the results are deposited in the results database at the conclusion of test execution.

### *TestController Agents*

Tests are never executed on the server where the Execution Manager and the other TestController processes are running. Machines included in the test lab execute a small daemon, the TestController Agent, to help with remote test execution. The Agent processes requests from the Execution Manager whenever a user has included that machine in a task list submitted for execution.

We have Agents that run on Windows and Solaris machines. Each Agent has a local configuration file that includes the installation information (location and version) for the test suites and tools available on that machine. If the Agent is asked to execute a test suite that hasn't been installed, it responds with an appropriate error.

Also included in the Agent installation is an *execution module* for each unique kind of test suite that it needs to execute. Running an API-based framework test is much different than running a GUI-based regression test, as is interpreting the results to determine the status and log information for each test that was run. The execution modules take care of these differences so that they all look the same to the Agent. When asked to run a particular test suite, the Agent actually runs the execution module that knows about that kind of suite. The execution module is then responsible for preparing for the test run, generating the command line and parsing the results at the end. Once the run is complete, the results are returned to the Execution Manager for storage.

### *Coordinated Multi-Machine Test Scenarios*

Most test execution tasks require just one machine, but some need more. For example, a test running against the ICCS GUIs on one machine may require manipulation of a hardware simulator on another in order to force an off-normal condition. The Execution Manager supports this by assigning *roles* to the machines specified in an execution task. In this example, the machine running the GUIs would be considered the *client* machine while the machine running the simulator is a *helper*. The

information passed by the Execution Manager to the Agents (and subsequently to the execution modules) includes a list of all the participating machines and what their roles are. With this, the test running on the client machine can, in the middle of the test, contact the helper on the other machine to make a request. Once the helper has taken the requested action (manipulating the simulator, in this case), it responds back to the client machine and test execution continues.

### *Status Reporting*

Status of all currently executing and previously completed automated tests is available through the web interface. While tests are executing, the Execution Manager receives updates from the remote Agents on a regular interval. These updates include a copy of the current log file from the execution module(s) running the tests. The live status information is immediately made available to the Status Manager. This allows the user to view the status of the run before it has completed. Users can also request to be notified by email when a particular testing task has finished. The email body includes an abbreviated status of the task along with links to retrieve more detailed information.

## TEST RESULTS SEARCH

Results for both automated and manual tests are stored in the test results database. The TestController web interface provides a search page for assembling custom queries using any combination of attributes (eg. test name, release ID, test result, executing user, test machine, etc.). Queries can be bookmarked for easy reuse.

The test results search mechanism also satisfies the demanding requirements of the automated status reports discussed in the following section.

## REAL-TIME STATUS REPORTING

Every ICCS release has a Release Testing Summary page that provides current status for every test area identified in the release test plan (see Figure 2). The report content is updated every time the page is visited. At the top of the page is a high-level summary of the status for each of our three test cycles: integration testing, formal testing and online testing. The user is able to drill down through the status summaries all the way to individual test logs, if desired (to view test failures, for example).

## SUMMARY

For any large software project, managing the testing for every product release is a difficult, time-consuming task. For the NIF ICCS team, the ICCS TestController helps plan, manage, execute and report status for the thousands of tests performed throughout the software lifecycle. As an integral part of a rigorous three stage testing approach, it helps ensure that the software controlling the National Ignition Facility is of excellent quality.

## REFERENCES

[1] L. Lagin, et al., "Status of the National Ignition Facility Integrated Computer Control System on the Path to Ignition," Fusion 2007, Quebec, July 2007.

[2] A. P. Ludwigsen, "Software Engineering Processes Used to Develop the National Ignition Facility Integrated Computer Control System," ICALEPCS 2007, Knoxville, October 2007.

[3] D. Casavant, et al., "Testing and Quality Assurance of the Control System During NIF Commissioning," ICALEPCS 2003, Gyeongju, October 2003.

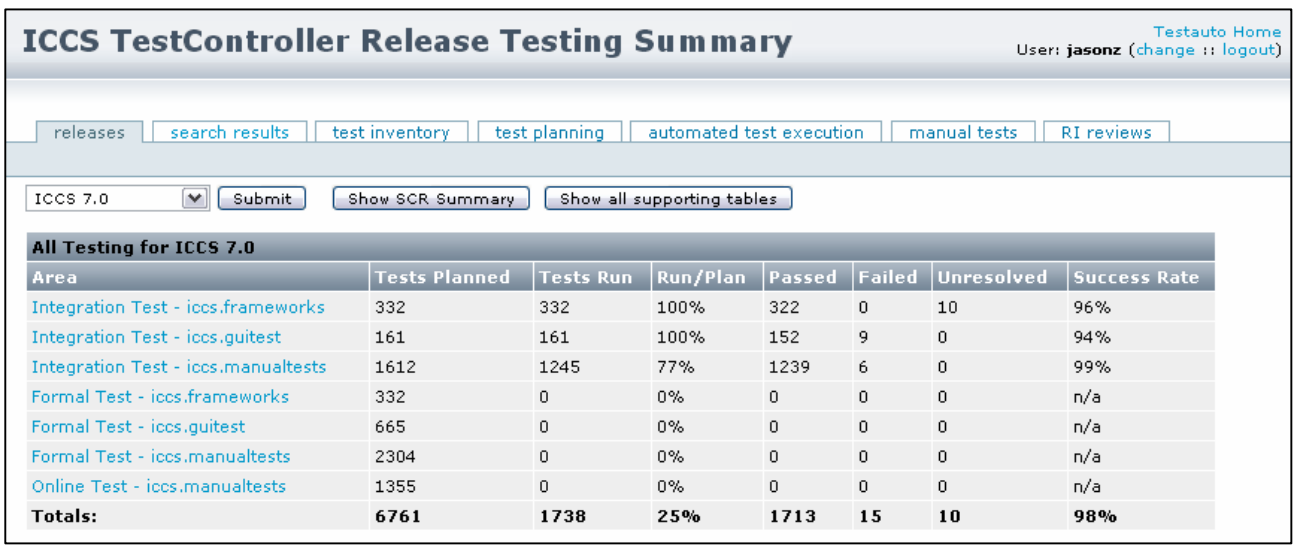


Figure 2: This is a screenshot of a release testing summary. Clicking on one of the high level areas in this table reveals more detailed information about what is being summarized. Subsequent drill-downs can lead all the way to the execution logs of the tests being counted.