Test Plan of the Test Manager for the ATLAS DAQ Prototype -1

Authors : R.Hart

Keywords : DAQ, Test Manager, test, prototype -1, functionality, reliability, performance.

Abstract

This note describes the plan to test the Test Manager (TM), concerning its functionality, its reliability and its performance.

NoteNumber : 140
Version : 1.0
Date : March 20 2000
Reference : http://atddoc.cern.ch/Atlas/Notes/140/Note140-1.html

1 Introduction

The TM is an ATLAS back-end component to use tests in an organized way. Its implementation is based on the high-level design **[1]** and described in the implementation report **[2]**. The user guide **[3]** explains how to use it. This note presents a test plan to check its functionality, to control its error recovery and to measure the performance. The TM uses several other back-end components. The most important ones are: the Process Manager (PMG) **[4]** and the Software database configuration **[5]**.

2 Features to be tested

All checks and tests are based on the *demo test-suite*, which is the default TM_Repository. It contains a couple of tests all pointing to the *tmgr_sleep* program. The synopsis of this program is as follows:

```
tmgr_sleep [-t delta] [-E exit]
```

Without arguments it sleeps for 30 seconds and exits with the test result `TmPass`, which means, according to the POSIX 1003.3 **[6]** standard, the test succeeded. With the `-t` argument the sleep duration can be changed and with the `-E` flag the exit status can be changed. The default arguments are used, unless stated otherwise.

All checks and tests listed below have to comply with the documentation of **[1]**, **[2]** and **[3]**. They have to run to completion to be valid. The results of the Test Plan will be documented in a Test Report.

2.1 Functionality

1. List available tests from *demo test-suite*. Check if list [`PinaColada, CubaLibre, PiscoSour, TripleSec`] is correct.
2. List all computers of *demo test-suite*. Check if list [`lnxatd01, rd13fe11, lynxdev, sunatdaq01, sunatdaq02, kuta, jura`] is correct.
3. Obtain static information of a particular test (for instance `PiscoSour`) and check it:

```
Description:    [Base test: no defaults, no environment]
Author:         [Judge Dee]
HelpLink:       [http://]
DefHost:        [sunatdaq01]
Programs:       [solaris,lynx,linux]
```

4. Obtain variable information of a particular test machine combination (for instance `PiscoSour, sunatdaq01`) and check it:

```
Executable:     [tmgr_sleep]
DefParams:      []
Environment:    []
Machine:        [sunatdaq01.cern.ch]
OS:             [solaris]
Type:           [Workstation]
```

5. Start a test asynchronously on the default machine and check the result. After 30 seconds the test should finish with result `Pass`, state `Finished`.
6. Start a test asynchronously on a particular machine and check the result. After 30 seconds the test should finish with result `Pass`, state `Finished`.
7. Perform previous test on different machines and platforms.
8. Perform step 5, 6 and 7 but now synchronously. Results and states have to be the same.
9. Start several tests (asynchronous) in parallel on several machines and check the results.
10. Start a test asynchronously on an arbitrary machine and stop it, while it is running. Check the result of the interrupted test: result `Unresolved`, state `Interrupted`.
11. Start a test asynchronously on an arbitrary machine with a timeout smaller than the sleep duration of the test. Check if the test was stopped after the timeout by expiration: result `Unresolved`, state `Expired`.
12. Repeat step 11, but now with a timeout larger than the sleep duration. Check if the test terminates normally: result `Pass`, state `Finished`.
13. Repeat step 11 and 12 for a synchronously started test. Results and states have to be the same.
14. Set the global timeout to a value smaller than the sleep duration of the test. Start several tests asynchronously and check if they are stopped by expiration: result `Pass`, state `Finished`.
15. Retrieve the `test_log` list of all launched tests and examine the results.

2.2 Recovery

16. Try to start a test on a machine without a *pmg_agent*. Check returned status and error: `Error: PMG_Error (No_Agent)`
17. Kill *pmg_agent* while test is running. The client will be unaware of the result of the test. It remains in the `Running` state. Terminating it explicitly by `StopTest` or implicitly by timeout yields the error: `PMG_Error` (no further detail).
18. Kill running test by hand and check the result: `Unresolved`, state: `Killed`.
19. Try to start a test of which no binary is available. Check returned error and status: `Error: PMG_Error (Creation_Failure)`

2.3 Performance & Reliability

20. Measure the average local and remote test creation time of a asynchronously started test.
21. Measure the average local and remote killing time, when stopping a test.
22. Check the reliability of the TM when starting 100 tests on the local machine at the same time. After completion (30 seconds) examine the `test_log` and check that all tests are `Finished` with result `Pass` and that they have ended at the same time (within 2 seconds accuracy).
23. Perform the previous step on a remote machine.

24. Perform step 22 but let the test expire with a timeout less than the sleep duration of the test (result `Unresolved`, state `Expired`).
25. Perform previous step on a remote machine.
26. Perform step 22 with argument `[-t 0]` for all tests. It implies that the tests exit immediately.
27. Perform previous step on a remote machine.
28. **Endurance** test. Run several tests asynchronously with a different timeout and different duration times (chosen randomly) on different machines and platforms in a loop for an extended period (several hours).

3 TM Testware

The TM under test should run in its own partition (for instance `tmtest`). It needs an *ipc_server* running in this partition. The PMG needs an IS server for its PMG_DynDB. The following list shows the syntax for the different servers:

```
bash# ipc_server & // default ipc server
bash# ipc_server -p tmtest &
bash# is_server -p tmtest -n PMG &
bash# pmg_agent -p tmtest &
```

The *ipc_server*'s and *is_server* have to run somewhere on an arbitrary host within the AFS framework. The default *ipc_server* may already run. The *pmg_agent* has to run on each machine you want to execute a test.

Almost all checks and tests from chapter 2 can be carried out with the *tmgr_try* program. Synopsis:

```
tmgr_try -p tmtest
```

The endurance test (no. 28) has to be developed yet and will probably be a combination of several scripts and programs.

4 References

[1]R.Hart, H.Boterenbrood, W.Heubers, **Test Manager design for the ATLAS DAQ Prototype-1,** Technical Note 066, V3.1, June 19 1998 (URL: http://atddoc.cern.ch/Atlas/Notes/066/Note066-1.html).

[2]R. Hart, **Implementation of the Test Manager for the ATLAS DAQ Prototype-1**, Technical Note 111, V2.0, Oct. 11 1999 (URL:http://atddoc.cern.ch/Atlas/Notes/111/Note111-1.html).

[3]R. Hart, **User Guide of the Test Manager for the ATLAS DAQ Prototype-1**, Technical Note 112, V2.0, Oct. 13 1999 (URL:http://atddoc.cern.ch/Atlas/Notes/112/Note112-1.html).

[4]P.Duval, L.Cohen, **Users guide for the Process Manager**, Technical Note 081, V1.0, 2 Feb. 1998 (URL: http://atddoc.cern.ch/Atlas/Notes/081/Note081-1.html).

[5]I.Soloviev, **Configuration Databases User's Guide**, Technical Note 135, V1.0, 6 Oct. 1999, (URL: http://atddoc.cern.ch/Atlas/Notes/135/Note135-1.html).

[6]Rob Savoye, **The DejaGnu Testing Framework** for DejaGnu Version 1.3, Jan. 1996 (URL: http://phantom.iweb.net:80/docs/gnu/dejagnu).