



ETA v2.1

OVERVIEW

EXCERPTS FROM

USER GUIDE

&

COMMAND REFERENCE

Etaliq Inc.
4B-2548 Sheffield Road
Ottawa, ON K1B 3V7

Phone: (613) 241-1385
Fax: (613) 241-1523
<http://www.etalIQ.com>

Chapter 1

Introduction to ETA

1.1 Process Overview

The ETA application supports all facets of test automation and reporting:

- The creation and retention of test plans and the automation of test cases.
- The creation and retention of execution tables defining devices or testbeds where test plans will be executed.
- The scheduling and execution of these test plans against one or more testbeds or devices defined in execution tables.
- The detailed reporting and logging of executions for easy review.
- Summarized reporting of execution results and resource usage.

Figure 1.1, “Etalq process” on the following page represents the various components, processes, and sub-systems of the ETA application¹. For further information on the planned availability of these sub-systems as well as other features, please contact your Etalq Support representative.

1.1.1 Create/Import Test Plan

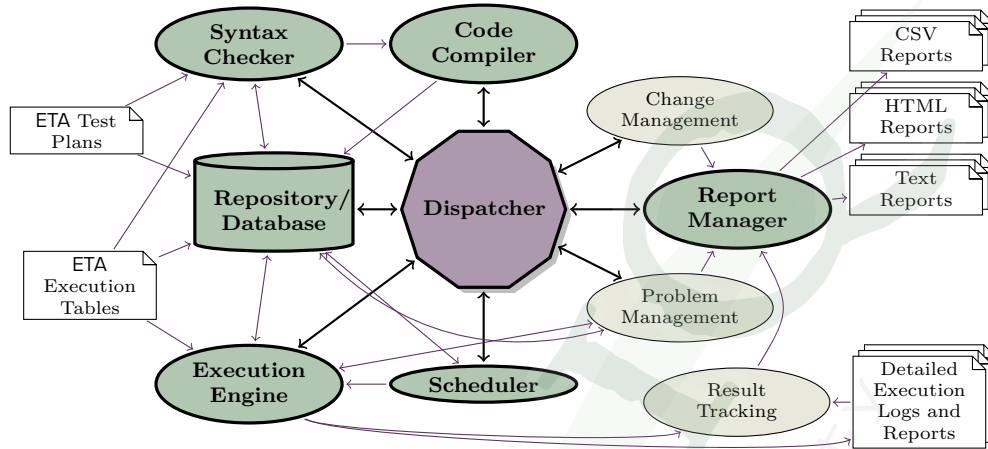
Create a test plan with any text editor and import into ETA database. This feature is discussed in more detail in chapter 12, “Creating a Test Plan” on page 85.

1.1.2 Execute Test Plan

Execute a test plan to occur immediately or set up a schedule all from within the ETA Client GUI. This feature is explained in more detail in chapter 6, “Execution Scheduling” on page 41.

¹The *Change Management* and *Problem Management* sub-systems are not implemented in the current version of ETA.

Figure 1.1: Etaliq process



1.1.3 Review Results

Review the detailed results or logs associated with any execution, which can be archived (baselines) or downloaded. This feature is explained in more detail in chapter 9, “Reviewing Executions” on page 59.

1.1.4 Summarized Reports

Create executions or node usage summary reports dynamically for forecasting and analysis. This useful tool aids in resource usage within the testing cycles and schedules. This feature is explained in more detail in chapter 11, “Summarized Reporting” on page 73.

1.1.5 Archive or Download Reports

Download text versions of detailed execution test plans, execution tables, reports, and logs for off database storage or move entire execution reports to archive. This feature is explained in more detail in chapter 1, “Introduction to ETA” on the previous page.

1.2 ETA System Architecture

ETA (Easy Test Automation) is a test automation tool for testing any system or software supporting a text based command-line interface. Text based testing protocols that are used to communicate with these devices includes Telnet, TL1, SSH, SNMP, and various socket based communications protocols. In addition, ETA pro-

vides the ability to use various shell environments (Sh, Tcsh, Tclsh, etc.) and direct sockets (MySQL®, etc.) to communicate with the devices used for testing.

The ETA Automation Language is an efficient and flexible method used to automate Systems Under Test (SUTs) and other vendor products used to test these systems. Examples of devices used in testing include traffic generators, session replicators, and capture/decode software/hardware.

ETA comprises the following sub-system elements:

- ETA Automation Language
- ETA Code Compiler
- ETA Syntax Checker
- ETA Execution Engine
- ETA Repository/Database
- ETA Scheduler
- ETA Report Manager

1.2.1 ETA Automation Language

The ETA Automation language is a 4th generation language, which is syntax checked and pre-compiled. ETA's reduced instruction set is written to make it quick and easy for anyone to automate tests. Test plans are written using any standard text editor or any off-the-shelf word processor that is capable of saving in a raw ASCII text format. ETA uses this version of test plan file as its source code.

1.2.2 ETA Syntax Checker

The first time a specific test plan/execution table combination is scheduled for execution, a Syntax Check is automatically performed. If desired, this check can be performed (or repeated) through the *Scheduler* interface.

The syntax checker is responsible for verifying the following:

- That resources required in the test plan are appropriately defined in the execution table
- That the test plan contains all mandatory markers denoting test plan, group and case sections, attributes, IDs, and definitions
- That the command syntax used for each command is according to syntax rules
- That declared lists, variables, flags, and named commands are correctly used in conjunction with other commands in the test plan

During the compilation step, hardware abstraction is applied throughout the test plan using the **REPLACE** commands contained therein. Hardware abstraction allows the same set of tests to execute against various nodes and hardware types without having to create entirely different test plans. An example of this is to **REPLACE** 'Serial0/1' with 'Ethernet2/2', throughout the test plan. By applying this **REPLACE** command to a test plan prior to execution; the test executable will operate against the **Ethernet2/2** port instead of the **Serial0/1** port. This is one example of how ETA supports hardware abstraction.

1.2.3 **ETA Execution Engine**

The ETA Execution Engine uses the test plan file to run tests and completes the hardware abstraction routines to modify tests for hardware platform differences. Tests can execute against any node, slot, or port without having to be rewritten.

1.2.4 **ETA Repository/Database**

The ETA Repository/Database contains the following:

- All user information
- Test plans and their associated groups, cases, and attribute definitions, including previous versions
- All tables (i.e. **exec**, **node**, **slot**, **card**, and **replace**), including all of their versions
- All execution result files with all of their associated reports and logs
- All node usage statistics and execution results

1.2.5 **ETA Scheduler**

The ETA Scheduler starts executions as requested by the users. It also keeps track of what is currently running, to make sure that multiple executions do not attempt to use the same nodes at the same time. When executions are requested, it ensures that they have been pre-verified so that users will know, in advance, whether their executions will be able to run at the allotted time. In addition, it calculates future run times of recurring schedules.

1.2.6 **ETA Report Manager**

ETA Report Manager is responsible for the creation, the maintenance, and the archiving of the logs and reports associated with *Executions*. It also rolls up the results of multiple executions to provide summarized *Execution Reports* and tracks the usage of nodes by executions and manual lockouts to provide summarized *Node Usage Reports*.

Under *Executions* are reports for individual executions. Each references all the files involved in each execution. Further, each contains reports on all the details of the execution. The non-input-file reports are as follows:

Attributes the execution metadata

Syntax Report all the output of the parsing and compilation steps

Detailed Report the complete output of the execution run

Summary Report a short report with one line per test

Console Log(s) the complete output from each node

Summarized Reporting is used to track results across many executions or to track resource usage over many days, weeks, or months. *Summarized Execution Reports* show how many test cases match certain constraints, broken down arbitrarily. (e.g. how many test cases **PASS**ed each day this past week). *Summarized Node Usage Reports* show when nodes were used for manual or automated testing, and it can also be constrained or broken down by *Node*, *User*, or both.

1.2.7 ETA Server Directory

ETA directories are set up to store and maintain the various files used by the system. The default directories for all ETA Server application file data are:

- /usr/share/eta
- /var/lib/eta
- /var/log/eta

1.3 Role Appropriate Features

1.3.1 Manager

ETA assists in simplifying the gathering and reporting of information that managers need to determine the current state of SUT software/hardware. The *Summarized Reporting* of **PASS/FAIL** verdicts is used to show the current state of any project, sub-project, image, platform, target. At test scheduling time, the user defines a series of classifications for the entire set of executed tests. These classifications, in addition to several default attributes, are selected and filtered in order to create customized **PASS/FAIL** reports. See section §11.3, “*Summarized Execution Reporting*” on page 74.

In addition, ETA provides the ability to report on node usage statistics. Nodes are documented as “used for automation”, “locked-out for manual testing or reproduction”, or “unused”. This information is available for both retrospective analysis and future planning and enables managers to determine and optimize hardware usage within their labs. section §11.4, “*Summarized Node Usage Reporting*” on page 79.

Custom summarized execution report examples

- Create a summarized execution report for the current month, by unique execution ID (X-axis) and individual test case ID (Y-axis), which shows the verdicts for all tests executed within the month. This is useful in identifying test cases that changed status from **PASS** to **FAIL** or vice-versa. The report includes the drill-down links to the individual executions where the **PASS/FAIL** occurs.
- Create a Summarized *Execution Report* for an entire target/project by week (X-axis) and summarized totals for verdicts for each test plan (Y-axis) that has been executed. This is useful in identifying the general quality of SUT software on a weekly basis through a test or verification cycle.
- Create a Summarized *Node Usage Report* for the past week, by quarter-hour (X-axis) and by node (Y-axis), showing when the nodes have been in use, and when they sat idle. This will help you locate under utilized resources.

Figure 1.2: Common Manager Views

Execution Report		2008				TOTAL	
		7					
		27	28	29	30		31
ETA_NetInventory_Example	CHILDFAIL	0	8	0	1	1	10
	FAIL	0	8	0	1	1	10
	PASS	0	27	0	0	1	28
ETA_Router_Example	CHILDFAIL	19	51	4	0	0	74
	FAIL	17	43	4	0	0	64
	PASS	84	168	13	0	0	265
	UNCLEAN	6	11	0	0	0	17
ETA_SNMP_Example	CHILDFAIL	1	24	0	0	2	27
	FAIL	0	21	0	0	1	22
	INCOMPLETE	1	0	0	0	0	1
	PARENTEFAIL	4	0	0	0	0	4
	PASS	0	33	0	0	3	36
TOTAL		132	394	21	2	9	558

(a) A Summarized *Execution Report*

Node Usage Report	2008-07-02					
	00h	04h	08h	12h	16h	20h
cs2500_1						
etatest1						
gw						
csnx08						
csM1/Node1						
csM1/Node1						
csM1/Node4						
etademo1						
localhost						

(b) A Summarized *Node Usage Report*

1.3.2 Test Automation Engineer

ETA assists test automation engineers by reducing the amount of code necessary to accomplish any given task tenfold. Etaliq's execution engine sub-system provides a syntax checker that minimizes the time required to detect and correct typos and logic errors within the ETA language code. The ETA language is comprised of a few basic instructions, many of which combine to dramatically reduce code size and errors.

Etaliq's generic parsing capabilities save a tremendous amount of time by enabling any test automation engineer to define a set of expected results without the need for complex parsing utilities or techniques.

Additional high-productivity features that apply to test automation engineers are shown under section §1.3.4, “Operations Engineer” on page 12.

Figure 1.3: Common Test Automation Views

The figure illustrates the common test automation views in the ETA interface. It consists of three screenshots:

- Top Screenshot:** A table listing test cases. The columns are TYPE, VERICT, SANVERS, TEST PLAN, and TEST CASE. The rows show various test cases like 'execTable', 'group', 'test', and 'test' with their respective statuses and identifiers.
- Bottom-Left Screenshot:** A detailed view of a test execution. It shows the command being executed, the result (PASS), and the success rate. For example, 'SEND RouterNode3 "ping RouterNode3(INTF_Ifaddress)" PERCENT' results in a success rate of 100 percent (5/5).
- Bottom-Right Screenshot:** A summary report for a test case. It includes the test case identifier, title, authors, status, version, keywords, and a detailed result list. The result list shows the success rate and the specific test results.

High-Productivity ETA Language Examples

- Send Receive and Verify

```

1 RESULTLIST(verifyIntState)
2 "line status = administratively down"
3 "port status = enabled"
4 "input packets = 0"
5 "input errors = 0"
6
7 SEND Node1 "show port status" verifyIntState

```


The above line sends, to “Node1”, the command “show port status”, then verify the expected result definitions from the list named “verifyIntState” against the response received to the command. Each individual item in the result list is verified individually and assigned a verdict.

- Verify Statistics Changes

```

1 RESULTCOMPLIST(verifyIntStatistics)
2   "input packets INCREASEBYATLEAST 5"
3   "input errors EQUAL"
4   "output packets INCREASEBYATLEAST 5"
5   "output errors EQUAL"
6
7 SEND Node1 "show port statistics" verifyIntStatistics
8 SEND Node1 "ping 192.168.1.200 count 5 size 512"
9 SEND Node1 "show port statistics" verifyIntStatistics

```

The above set of instructions, sends the “show port statistics” command to “Node1”, then pings a destination, and again sends the same command, “show port statistics”, to the node. The response from the first show command are compared against the response from the second show, based on the expected results in the result compare list named “verifyIntStatistics”

- Wait Until an Event before proceeding

```

1 RESULTLIST(verifyIntState)
2   "line status = active"
3   "port status = enabled"
4
5 WAIT 30sec Node1 "show port statistics"
6                   verifyIntState interval=2sec

```

The above command will Wait up to 30 seconds for all of the expected results defined within “verifyIntState” to be true, prior to proceeding. The command will be sent and the response verified every 2 seconds, to verify whether the results are true. If results are true after any given interval, the automation proceeds without wasting additional time.

- Further Reading
 - For examples of complete test plans, see appendix C on page 293.
 - For the ETA language details, see chapter 14 on page 107.
 - For ETA’s unique IP Expression syntax, see section §16.1.1 on page 230.

1.3.3 SUT Development Engineer

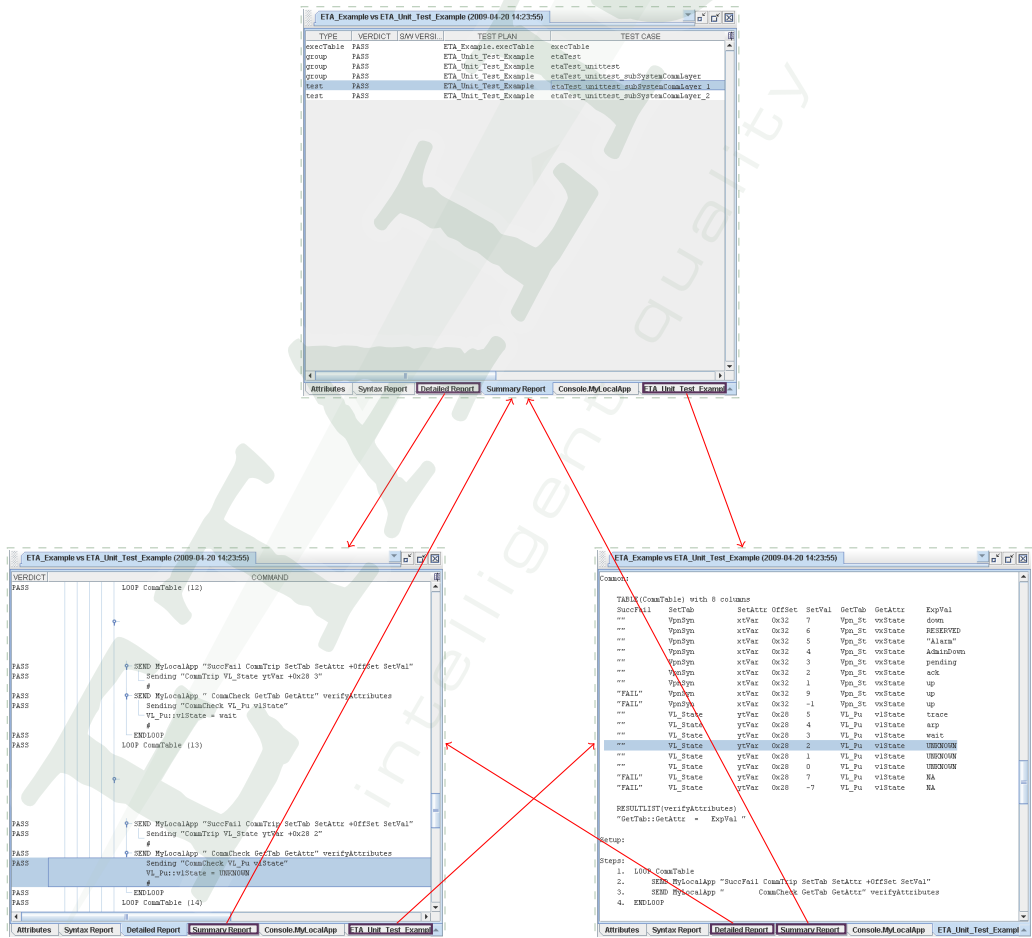
ETA assists SUT development engineers with any daily repetitive tasks required to test their development code on any device available to them. With a click of a

button, entire testbeds can be reset to a desired/expected state, making them ready for manual testing or automated sanity testing of latest compiled image.

Etaliq's execution engine sub-system provides a syntax checker that minimizes the time required to detect and correct typo's and logic errors within the ETA language code. The ETA language is comprised of a few basic instructions, many of which combine to dramatically reduce code size and errors.

The ETA language, comprised of a few basic instructions, and Etaliq's generic parsing capabilities help SUT development engineers focus on the work at hand, not on writing of automated tests.

Figure 1.4: Common SUT Developer Views



High-Productivity ETA Language Examples

- Unit Test

```

1 TABLE(UtAttributesTable) with 4 columns
2   AttrName SndValue ExpCmdStatus ExpOperState
3   "reg1"   "0"       "OK"         "initiated"
4   "reg1"   "1"       "OK"         "learning"
5   "reg1"   "-1"      "FAIL"       "learning"
6   "reg1"   "2"       "FAIL"       "learning"
7   "reg2"   "0x10"   "OK"         "passive"
8   # ...
9
10 RESULTLIST(CheckCmdStatus)
11   "command status = ExpCmdStatus"
12
13 RESULTLIST(CheckOperState)
14   "state = ExpOperState"
15
16 LOOP UtAttributesTable
17   SEND Node1 "compX-test-tool set AttrName SndValue" CheckCmdStatus
18   SEND Node1 "compX-test-tool get state"          CheckOperState
19 ENDLOOP

```

The above example defines a set of testable attributes, values and expected outcome in an easy to manage table named “UtAttributesTable”.

Each attribute and value is fed to the SUT developer’s ‘compX-test-tool’ Unit Test (UT) test tool for component “compX”. The returned command status of each operation and the resulting operational state of the component are verified for each line of the table.

With the simplicity of ETA’s verb language and Etaliq’s generic parsing capabilities, writing reusable and one-off UT scripts is now quick and easy and at the reach of any SUT development engineer.

1.3.4 Operations Engineer

ETA assists operations engineers by reducing the amount of time necessary to review execution logs and reports by providing a highly structured *Detailed Report* output format in tree format so that the operations engineer can quickly drill down to the location of a failure. This *Detailed Report* is relative position linked to all other *Console Log(s)* and files used or created during the execution including the source code in the test plan itself. This allows an operations engineer to quickly identify a failure condition and determine the actual cause, both within the *Console Log(s)* of the devices in the testbed and within the test plan source code itself.

ETA also provides operations engineers with the ability to maintain a set of repeating schedules in order to optimize both the use of the existing testbed resources while ensuring the broadest coverage by executing all tests against a series of images. Often in today’s automation labs, there is not enough time and/or resources, to execute all automated regression tests. This often results in the same tests being scheduled to run against images repeatedly, while other tests of executed

infrequently or not at all during a test cycle. ETA schedules can be set up to execute tests 1, 3, 5, and 7 every Monday, Wednesday and Friday, and tests 2, 4, and 6 to execute every Tuesday, Thursday and Saturday, and finally all tests 1 to 11 to be run every Sunday.

- ETA's full function scheduler allows operations engineers to schedule many different test plans and test cases to execute repeatedly on a schedule without user intervention. Specific test plans or portions thereof can be scheduled to execute daily, weekly, monthly or in any combination in order to achieve the maximum coverage of SUT images.

Figure 1.5: Common Operations Engineer Views

The figure illustrates three key views in the ETA software:

- Left Panel (Tree View):** Shows the project hierarchy under 'ETA'. Key folders include 'Server: etatest1', 'User Management', 'Executions', 'My Executions', 'Summarized Reporting', and 'Scheduler'. Under 'Executions', there are sub-folders for 'This Weeks' and 'My Executions', containing various test reports with status indicators like '[pass]', '[fail]', and '[scheduled]'. The 'Scheduler' folder is highlighted with a red box, and a red arrow points from it to the Scheduler window.
- Top Right Window (Scheduler):** Displays the configuration for a scheduled task. It includes tabs for 'Details', 'Recurrence', and 'Execution'. The 'Recurrence' tab is active, showing a start time of '2009-01-05 01:30:00', an end time of '2009-04-27 17:52:19', and a frequency of '36 occurrences(s)'. The 'Weekly' tab is selected, showing a recurrence of 'every 1 week(s) on' with days 'Sunday' through 'Saturday' listed below. A red arrow points from the Scheduler window to the Execution Report Query window.
- Middle Right Window (Execution Report Query):** Shows a list of test cases and plans. It has tabs for 'Details', 'Recurrence', and 'Execution'. The 'View' tab is active, displaying a tree of test cases under 'etaTest' and 'routerTest'. A red arrow points from the Execution Report Query window to the console window.
- Bottom Right Window (Console):** Displays the output of a test execution. The title bar reads 'ETA_Example vs ETA_Two_Router_Example (2009-04-17 10:46:16)'. The output shows a series of commands and their results, including 'show int Serial0/0', 'show int Serial0/0', and 'show int Serial0/0'. The output is mostly 'PASS' but includes a 'SUB FAIL' for 'Validating WaitRouteLinkUp' and another 'SUB FAIL' for 'Wait reception 40'. A red arrow points from the console window back to the tree view.

See section §11.3, “*Summarized Execution Reporting*” on page 74 for more information.



Copyright

ETA® Copyright © 2000–2009 by Etaliq Inc.

All rights reserved.

Contact information for Etaliq Inc. is at the Etaliq website at <http://www.etaliq.com>.

This document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this document may be reproduced in any form by any means without the prior written authorization of Etaliq Inc. Documentation is provided “as is” without warranty of any kind, either expressed or implied, including any kind of expressed or implied warranty of non-infringement or the implied warranties of merchantability or fitness for a particular purpose.

Etaliq Inc. reserves the right to change any products described herein without notice. Etaliq Inc. assumes no responsibility or liability arising from products described herein, except as expressly agreed to in writing by Etaliq Inc. The use and purchase of this product does not convey a license under any patent rights, trademark rights, or any other intellectual property rights of Etaliq Inc.

All other trademarks mentioned in this document or website are the property of their respective owners. The use of the word “partner” does not imply a partnership relationship between Etaliq Inc. and any other company.

Printed in Canada.