ETALIQ
intelligent quality

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, "Etaliq process" on the following page represents the various components, processes, and sub-systems of the ETA application[1]. For further information on the planned availability of these sub-systems as well as other features, please contact your Etaliq Support representative.

1.1.1 Create/Import Test Plan

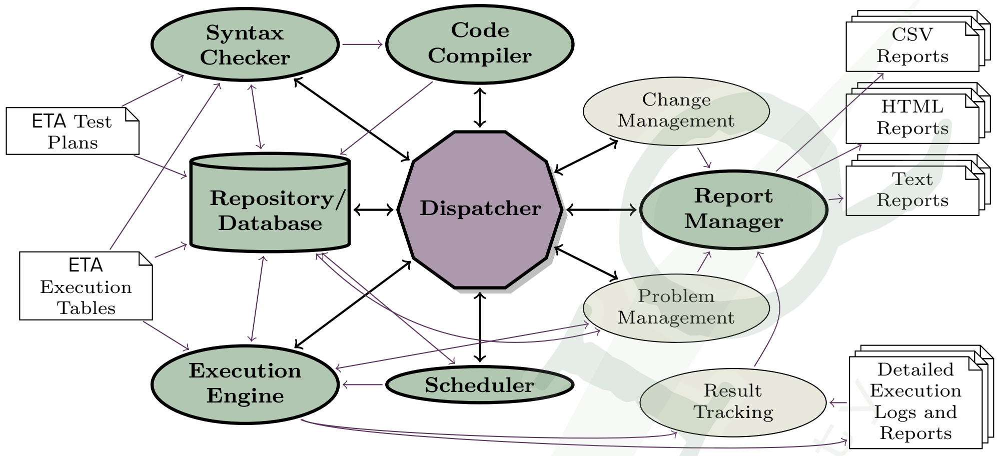
Create a test plan with any text editor and import into ETA database. This feature is discussed in more detail in chapter 12, "Creating a Test Plan" on page 85.

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.

[1]The *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 provides 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.

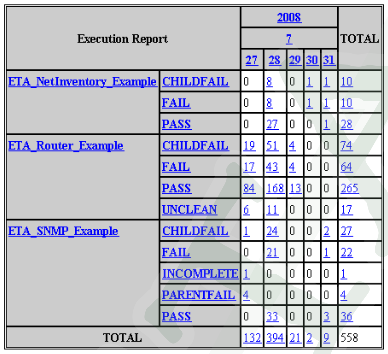
Custom summarized execution report examples

- 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.

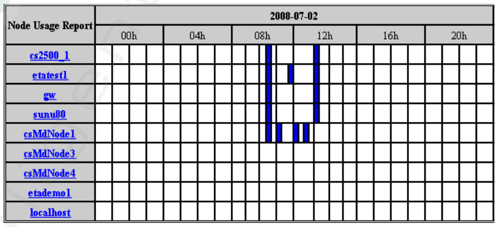
- Create a Summarized *Node Usage Report* for the past week, by quarter-hour (X-axis) and by node (Y-axis), showing when the nodes have been in use, and when they sat idle. This will help you locate under utilized resources.

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
	PARENTFAIL	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*



(b) A Summarized *Node Usage Report*

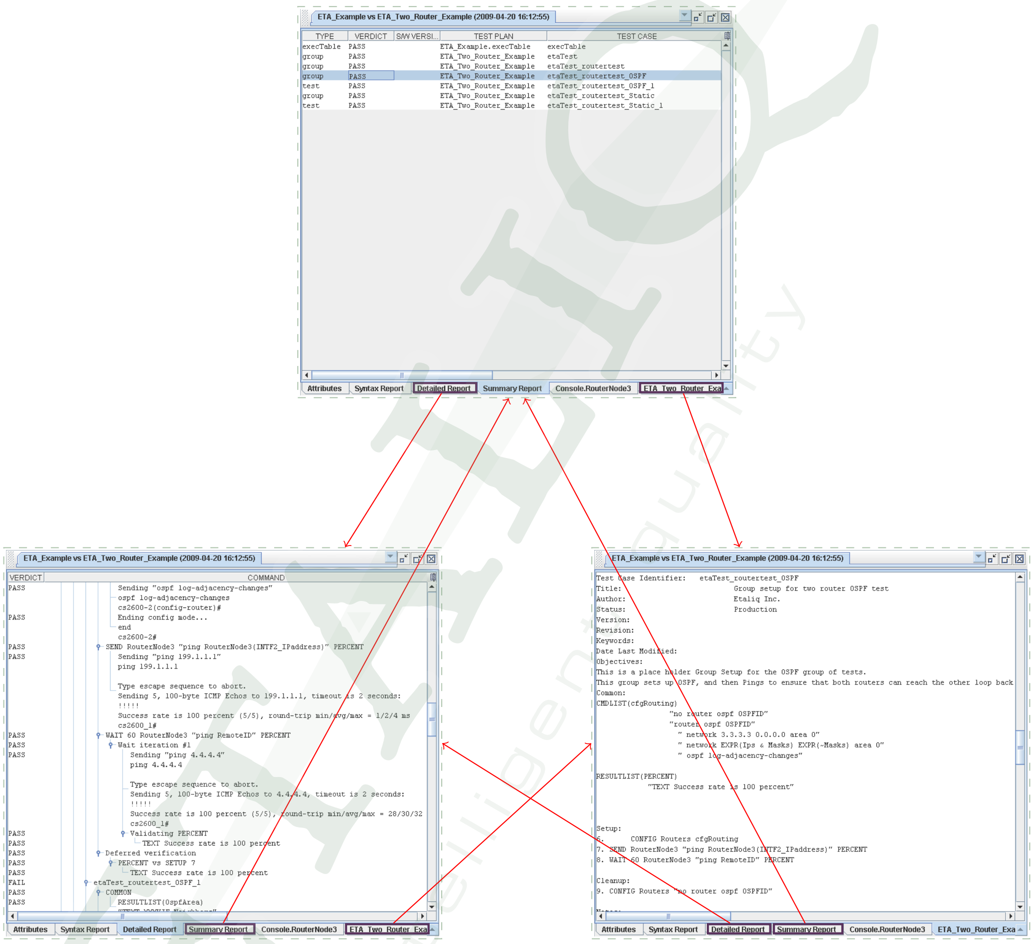
1.3.2 Test Automation Engineer

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 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.

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



TYPE	VERDICT	S/W VERSI...	TEST PLAN	TEST CASE
execTable	PASS		ETA_Example.execTable	execTable
group	PASS		ETA_Two_Router_Example	etaTest
group	PASS		ETA_Two_Router_Example	etaTest_routertest
group	PASS		ETA_Two_Router_Example	etaTest_routertest_OSPF
test	PASS		ETA_Two_Router_Example	etaTest_routertest_OSPF_1
group	PASS		ETA_Two_Router_Example	etaTest_routertest_Static
test	PASS		ETA_Two_Router_Example	etaTest_routertest_Static_1

```
VERDICT   COMMAND
PASS              Sending "ospf log-adjacency-changes"
                  ospf log-adjacency-changes
                  cs2600-2(config-router)#
PASS              Ending config mode...
                  end
                  cs2600-2#
PASS            SEND RouterNode3 "ping RouterNode3(INTF2_IPaddress)" PERCENT
PASS              Sending "ping 199.1.1.1"
                  ping 199.1.1.1

                  Type escape sequence to abort.
                  Sending 5, 100-byte ICMP Echos to 199.1.1.1, timeout is 2 seconds:
                  !!!!!
                  Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/4 ms
                  cs2600_1#
PASS            WAIT 60 RouterNode3 "ping RemoteID" PERCENT
PASS              Wait iteration #1
PASS                Sending "ping 4.4.4.4"
                    ping 4.4.4.4

                    Type escape sequence to abort.
                    Sending 5, 100-byte ICMP Echos to 4.4.4.4, timeout is 2 seconds:
                    !!!!!
                    Success rate is 100 percent (5/5), round-trip min/avg/max = 28/30/32
                    cs2600_1#
PASS                Validating PERCENT
PASS                  TEXT Success rate is 100 percent
PASS            Deferred verification
PASS              PERCENT vs SETUP 7
PASS                TEXT Success rate is 100 percent
FAIL          etaTest_routertest_OSPF_1
PASS            COMMON
PASS              RESULTLIST(OspfArea)
```

```
Test Case Identifier:   etaTest_routertest_OSPF
Title:                          Group setup for two router OSPF test
Author:                         Etaliq Inc.
Status:                         Production
Version:
Revision:
Keywords:
Date Last Modified:
Objectives:
This is a place holder Group Setup for the OSPF group of tests.
This group sets up OSPF, and then Pings to ensure that both routers can reach the other loop back
Common:
CMDLIST(cfgRouting)
                 "no router ospf OSPFID"
                 "router ospf OSPFID"
                   " network 3.3.3.3 0.0.0.0 area 0"
                   " network EXPR(Ips & Masks) EXPR(~Masks) area 0"
                   " ospf log-adjacency-changes"

RESULTLIST(PERCENT)
            "TEXT Success rate is 100 percent"



Setup:
6.      CONFIG Routers cfgRouting
7. SEND RouterNode3 "ping RouterNode3(INTF2_IPaddress)" PERCENT
8. WAIT 60 RouterNode3 "ping RemoteID" PERCENT

Cleanup:
9. CONFIG Routers "no router ospf OSPFID"
```

High-Productivity ETA Language Examples

- Send Receive and Verify

```
RESULTLIST(verifyIntState)
  "line status    = administratively down"
  "port status    = enabled"
  "input packets  = 0"
  "input errors   = 0"

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

```
RESULTCOMPLIST(verifyIntStatistics)
  "input packets  INCREASEBYATLEAST 5"
  "input errors   EQUAL"
  "output packets INCREASEBYATLEAST 5"
  "output errors  EQUAL"

SEND Node1 "show port statistics" verifyIntStatistics
SEND Node1 "ping 192.168.1.200 count 5 size 512"
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

```
RESULTLIST(verifyIntState)
  "line status = active"
  "port status = enabled"

WAIT 30sec Node1 "show port statistics"
                 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.

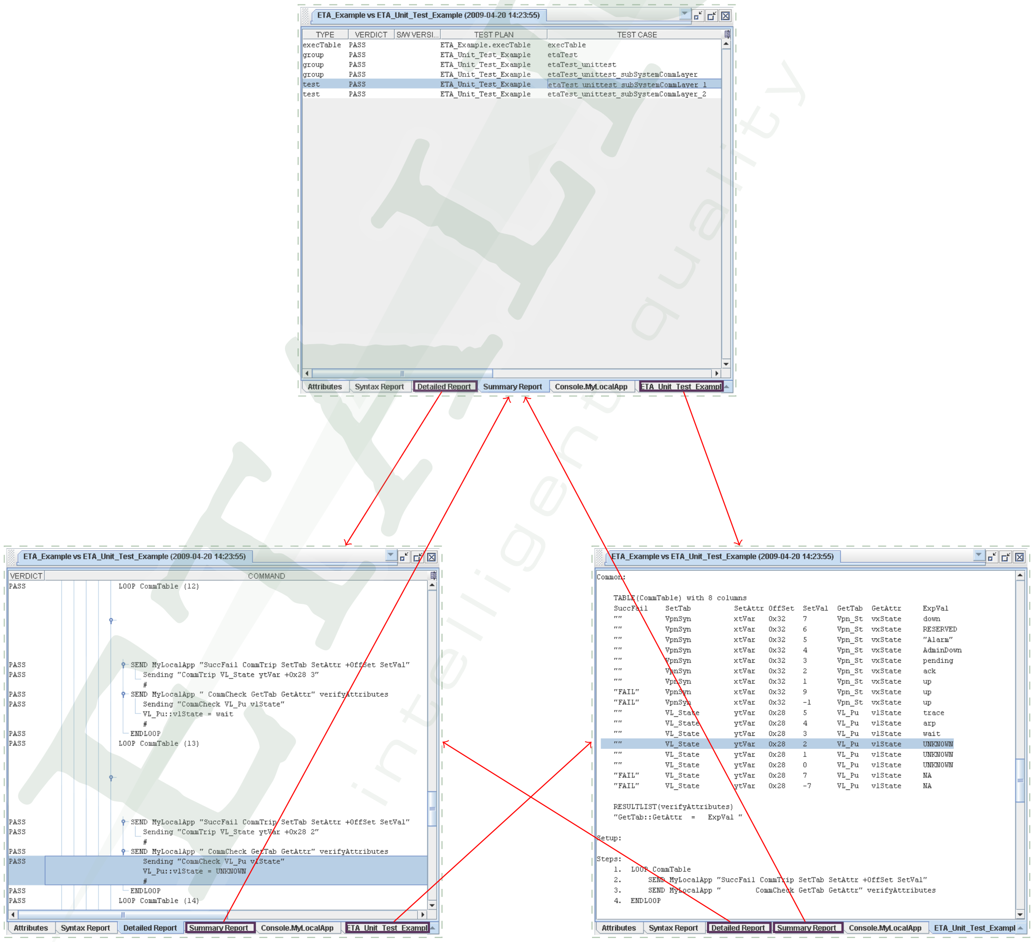
1.3.3 SUT Development Engineer

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



ETA_Example vs ETA_Unit_Test_Example (2009-04-20 14:23:55)

```
TYPE       VERDICT  S/W VERSI...  TEST PLAN                 TEST CASE
execTable  PASS                   ETA_Example.execTable     execTable
group      PASS                   ETA_Unit_Test_Example     etaTest
group      PASS                   ETA_Unit_Test_Example     etaTest_unittest
group      PASS                   ETA_Unit_Test_Example     etaTest_unittest_subSystemCommLayer
test       PASS                   ETA_Unit_Test_Example     etaTest_unittest_subSystemCommLayer_1
test       PASS                   ETA_Unit_Test_Example     etaTest_unittest_subSystemCommLayer_2
```

ETA_Example vs ETA_Unit_Test_Example (2009-04-20 14:23:55)

```
VERDICT   COMMAND
PASS        LOOP CommTable (12)


PASS          SEND MyLocalApp "SuccFail CommTrip SetTab SetAttr +OffSet SetVal"
PASS            Sending "CommTrip VL_State ytVar +0x28 3"
                #
PASS          SEND MyLocalApp " CommCheck GetTab GetAttr" verifyAttributes
PASS            Sending "CommCheck VL_Pu vlState"
                VL_Pu::vlState = wait
                #
PASS          ENDLOOP
PASS        LOOP CommTable (13)


PASS          SEND MyLocalApp "SuccFail CommTrip SetTab SetAttr +OffSet SetVal"
PASS            Sending "CommTrip VL_State ytVar +0x28 2"
                #
PASS          SEND MyLocalApp " CommCheck GetTab GetAttr" verifyAttributes
PASS            Sending "CommCheck VL_Pu vlState"
                VL_Pu::vlState = UNKNOWN
                #
PASS          ENDLOOP
PASS        LOOP CommTable (14)
```

ETA_Example vs ETA_Unit_Test_Example (2009-04-20 14:23:55)

```
Common:

   TABLE(CommTable) with 8 columns
   SuccFail    SetTab       SetAttr OffSet  SetVal  GetTab  GetAttr    ExpVal
   ""          VpnSyn       xtVar   0x32    7       Vpn_St  vxState    down
   ""          VpnSyn       xtVar   0x32    6       Vpn_St  vxState    RESERVED
   ""          VpnSyn       xtVar   0x32    5       Vpn_St  vxState    "Alarm"
   ""          VpnSyn       xtVar   0x32    4       Vpn_St  vxState    AdminDown
   ""          VpnSyn       xtVar   0x32    3       Vpn_St  vxState    pending
   ""          VpnSyn       xtVar   0x32    2       Vpn_St  vxState    ack
   ""          VpnSyn       xtVar   0x32    1       Vpn_St  vxState    up
   "FAIL"      VpnSyn       xtVar   0x32    9       Vpn_St  vxState    up
   "FAIL"      VpnSyn       xtVar   0x32    -1      Vpn_St  vxState    up
   ""          VL_State     ytVar   0x28    5       VL_Pu   vlState    trace
   ""          VL_State     ytVar   0x28    4       VL_Pu   vlState    arp
   ""          VL_State     ytVar   0x28    3       VL_Pu   vlState    wait
   ""          VL_State     ytVar   0x28    2       VL_Pu   vlState    UNKNOWN
   ""          VL_State     ytVar   0x28    1       VL_Pu   vlState    UNKNOWN
   ""          VL_State     ytVar   0x28    0       VL_Pu   vlState    UNKNOWN
   "FAIL"      VL_State     ytVar   0x28    7       VL_Pu   vlState    NA
   "FAIL"      VL_State     ytVar   0x28    -7      VL_Pu   vlState    NA

   RESULTLIST(verifyAttributes)
   "GetTab::GetAttr  =   ExpVal "

Setup:

Steps:
   1.  LOOP CommTable
   2.      SEND MyLocalApp "SuccFail CommTrip SetTab SetAttr +OffSet SetVal"
   3.      SEND MyLocalApp "        CommCheck GetTab GetAttr" verifyAttributes
   4.  ENDLOOP
```

High-Productivity ETA Language Examples

- Unit Test

```
TABLE(UtAttributesTable) with 4 columns
  AttrName SndValue ExpCmdStatus ExpOperState
  "reg1"   "0"      "OK"         "inited"
  "reg1"   "1"      "OK"         "learning"
  "reg1"   "-1"     "FAIL"       "learning"
  "reg1"   "2"      "FAIL"       "learning"
  "reg2"   "0x10"   "OK"         "passive"
  # ...

RESULTLIST(CheckCmdStatus)
  "command status = ExpCmdStatus"

RESULTLIST(CheckOperState)
  "state = ExpOperState"

LOOP UtAttributesTable
  SEND Node1 "compX-test-tool set AttrName SndValue" CheckCmdStatus
  SEND Node1 "compX-test-tool get state"             CheckOperState
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.

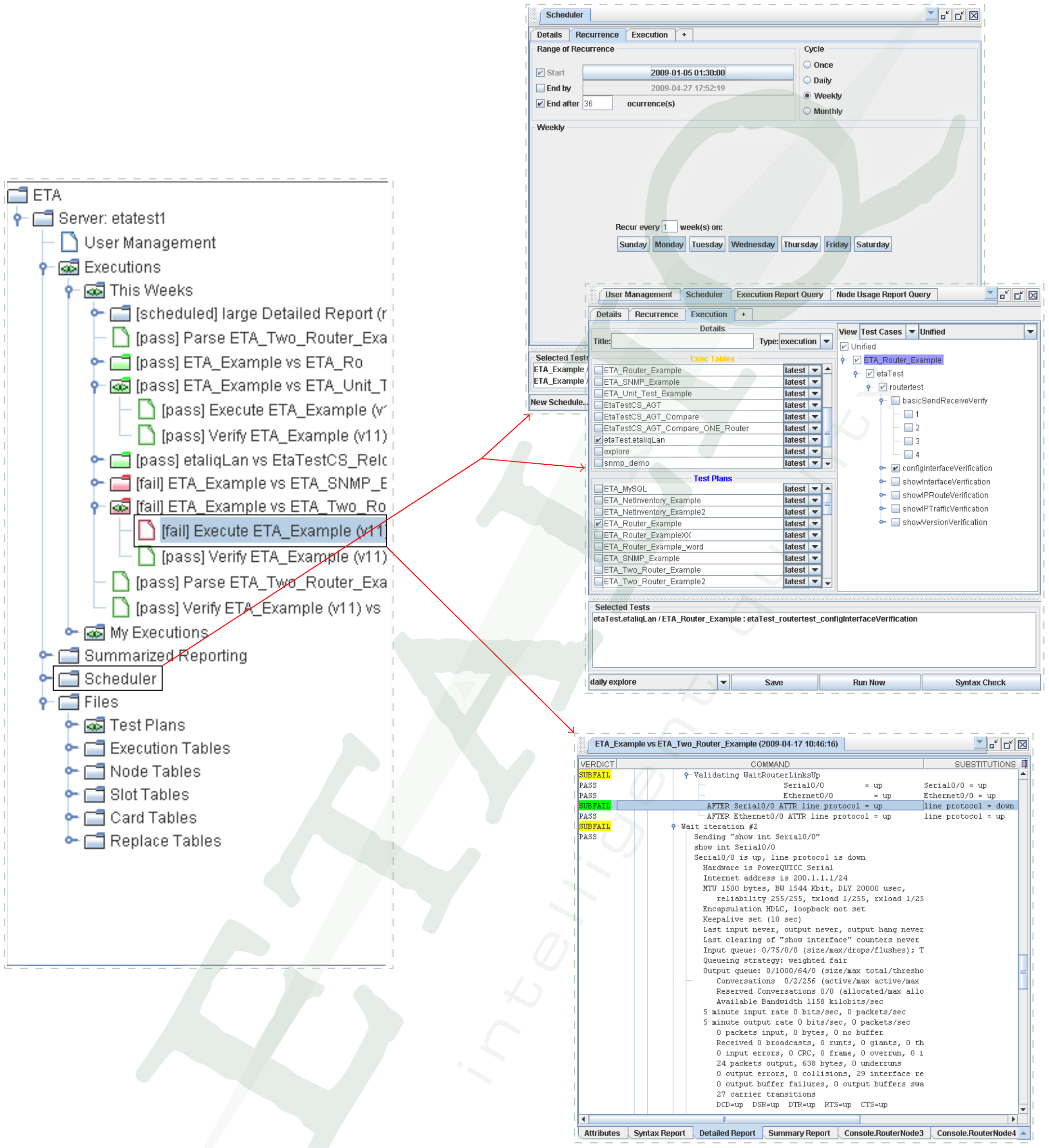
1.3.4 Operations Engineer

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 todays 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



ETA
Server: etatest1
User Management
Executions
This Weeks
[scheduled] large Detailed Report (r
[pass] Parse ETA_Two_Router_Exa
[pass] ETA_Example vs ETA_Ro
[pass] ETA_Example vs ETA_Unit_T
[pass] Execute ETA_Example (v
[pass] Verify ETA_Example (v11)
[pass] etaliqLan vs EtaTestCS_Relc
[fail] ETA_Example vs ETA_SNMP_E
[fail] ETA_Example vs ETA_Two_Ro
[fail] Execute ETA_Example (v11
[pass] Verify ETA_Example (v11)
[pass] Parse ETA_Two_Router_Exa
[pass] Verify ETA_Example (v11) vs
My Executions
Summarized Reporting
Scheduler
Files
Test Plans
Execution Tables
Node Tables
Slot Tables
Card Tables
Replace Tables

Scheduler

Details | Recurrence | Execution | +

Range of Recurrence
- [✓] Start: 2009-01-05 01:30:00
- [ ] End by: 2009-04-27 17:52:19
- [✓] End after: 36 ocurrence(s)

Cycle: Once / Daily / [✓ Weekly] / Monthly

Weekly

Recur every 1 week(s) on:

Sunday | Monday | Tuesday | Wednesday | Thursday | Friday | Saturday

Selected Tests
ETA_Example /
ETA_Example /
New Schedule...

User Management | Scheduler | Execution Report Query | Node Usage Report Query

Details | Recurrence | Execution | +

Details

Title: Type: execution

Exec Tables

[ ] ETA_Router_Example	latest
[ ] ETA_SNMP_Example	latest
[ ] ETA_Unit_Test_Example	latest
[ ] EtaTestCS_AGT	latest
[ ] EtaTestCS_AGT_Compare	latest
[ ] EtaTestCS_AGT_Compare_ONE_Router	latest
[✓] etaTest.etaliqLan	latest
[ ] explore	latest
[ ] snmp_demo	latest

Test Plans

[ ] ETA_MySQL	latest
[ ] ETA_NetInventory_Example	latest
[ ] ETA_NetInventory_Example2	latest
[✓] ETA_Router_Example	latest
[ ] ETA_Router_ExampleXX	latest
[ ] ETA_Router_Example_word	latest
[ ] ETA_SNMP_Example	latest
[ ] ETA_Two_Router_Example	latest
[ ] ETA_Two_Router_Example2	latest

View: Test Cases | Unified

- [✓] Unified
- [✓] ETA_Router_Example
  - [✓] etaTest
    - [✓] routertest
      - [ ] basicSendReceiveVerify
        - [ ] 1
        - [ ] 2
        - [ ] 3
        - [ ] 4
      - [✓] configInterfaceVerification
      - [ ] showInterfaceVerification
      - [ ] showIPRouteVerification
      - [ ] showIPTrafficVerification
      - [ ] showVersionVerification

Selected Tests
etaTest.etaliqLan / ETA_Router_Example : etaTest_routertest_configInterfaceVerification

daily explore | Save | Run Now | Syntax Check

ETA_Example vs ETA_Two_Router_Example (2009-04-17 10:46:16)

```
VERDICT                      COMMAND                                   SUBSTITUTIONS
SUBFAIL         Validating WaitRouterLinksUp
PASS                              Serial0/0        = up           Serial0/0 = up
PASS                              Ethernet0/0        = up         Ethernet0/0 = up
SUBFAIL            AFTER Serial0/0 ATTR line protocol = up        line protocol = down
PASS               AFTER Ethernet0/0 ATTR line protocol = up      line protocol = up
SUBFAIL       Wait iteration #2
PASS            Sending "show int Serial0/0"
                show int Serial0/0
                Serial0/0 is up, line protocol is down
                  Hardware is PowerQUICC Serial
                  Internet address is 200.1.1.1/24
                  MTU 1500 bytes, BW 1544 Kbit, DLY 20000 usec,
                     reliability 255/255, txload 1/255, rxload 1/25
                  Encapsulation HDLC, loopback not set
                  Keepalive set (10 sec)
                  Last input never, output never, output hang never
                  Last clearing of "show interface" counters never
                  Input queue: 0/75/0/0 (size/max/drops/flushes); T
                  Queueing strategy: weighted fair
                  Output queue: 0/1000/64/0 (size/max total/thresho
                     Conversations  0/2/256 (active/max active/max
                     Reserved Conversations 0/0 (allocated/max allo
                     Available Bandwidth 1158 kilobits/sec
                  5 minute input rate 0 bits/sec, 0 packets/sec
                  5 minute output rate 0 bits/sec, 0 packets/sec
                     0 packets input, 0 bytes, 0 no buffer
                     Received 0 broadcasts, 0 runts, 0 giants, 0 th
                     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 i
                     24 packets output, 638 bytes, 0 underruns
                     0 output errors, 0 collisions, 29 interface re
                     0 output buffer failures, 0 output buffers swa
                     27 carrier transitions
                     DCD=up  DSR=up  DTR=up  RTS=up  CTS=up
```

Attributes | Syntax Report | Detailed Report | Summary Report | Console.RouterNode3 | Console.RouterNode4

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

Copyright