ETALIQ
intelligent quality

Send, Receive, and Verify!

Part II: Automation is That Easy with ETA.

A White Paper
by Etaliq Inc.

www.etaliq.com

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

Phone: (613) 241–1385
Fax: (613) 241–1523
http://www.etaliq.com

Synopsis

Send, Receive, and Verify! Automation of test and operations is that simple with Etaliq's Easy Test Automation (ETA) infrastructure tool.

This document is the second in a series of white papers that describe the problems with, and solutions to, the many challenges in telecommunications test and operations. Part I describes many of the problems and inefficiencies in the systems that are in use today. It is highly recommended that you take the time to read "Send, Receive, and Verify! – Part I: Why Isn't Automation That Simple", to get a complete understanding of the inefficiencies and challenges that exist in today's telecommunications automation systems. This document, Part II, describes in detail how Etaliq, with its ETA product, has implemented solutions to all of the challenges presented in Part I. While Part I and Part II are written specifically for the Telecommunications industry, the same solutions apply to other engineering environments including Aerospace, Defense, Medical and Security.

The solution is Simple: Simplify and integrate the various components required to implement and operate an efficient and effective automation infrastructure. Simplify the functions to create, debug and maintain automated tests, and integrate the subsystems required to execute tests, review logs, and summarize their pass/fail and resource usage statistics. Simplifying these processes will allow more staff within your organization to quickly and easily participate in the automation processes. It would be no longer necessary to rely solely on automation and regression engineers.

Analysis of several production implementations shows that the Return On Investment (ROI) opportunities are huge when viewed from a micro economic perspective. Improving the efficiency of an individual user by as little as 10 percent reduces costs by, on average, 10,000 USD per year. Our detailed work study analysis has shown that the efficiency improvement for test automation and regression resources was in the order of 30 to 40 percent. Additionally, the elimination of in-house test tool development, maintenance and support staff provided even more ROI. Even more efficiencies, again in the order of 10 to 30 percent, were realized by both development and manual test engineers during Unit, Feature, Interoperability, and Al-

pha product test phases. A single ETA written utility reduced Lab Setup time from 4–5 weeks to just 3 days, providing a huge resource ROI. All phases of the product creation cycle were positively affected.

Analysis also shows that increasing the efficiency of automation infrastructure initialization and per-command execution time reduces lab resource costs dramatically. Our experience has shown that the initialization and run time efficiency improvements have freed up as many as 1 in 5 test lab setups, such that 1,000,000 USD and more, for each repositioned lab, was made available to other operational tasks.

Other potential returns, although less tangible, are also available when improving the overall operation of the Test and Automation Engineering disciplines. These returns are realized when quality objectives are attained sooner, time-to-market is reduced, and customer defect reports are reduced. Furthermore, improved product quality translates into strong customer satisfaction and corporate reputation.

Improving the efficiency of Testing in general within an organization also provides huge opportunities for ROI from a macro perspective. Testing is a very large percentage of the overall costs related to product engineering. Most estimates put it between 30 and 50% of product engineering costs. While the total dollar costs for each product are specific to each company, improving the efficiency of test has the potential to improve output and provide savings throughout the organization. Efficiency improvements in Testing will positively affect Development, Support and Maintenance Engineers, Test, Automation and Regression Engineers, and others, throughout the company.

To reiterate: Improving the efficiency of initialization and command execution times saves lab and human resources; and simplifying the functions to create, debug and maintain automated tests saves development, support and maintenance staff resources as well as lab hardware. These are only two of the many productivity enhancements built into the ETA automation infrastructure tool that contribute directly to real dollar ROI. Details on these two and many others are contained in the remaining sections of this document.

The opportunity for efficiency improvement has become very apparent in recent years. Several large multi-discipline corporations have recognized this opportunity through the acquisition of all of the largest players in the test and automation infrastructure prod-

uct space. The list is impressive. Acquiring companies include HP (Mercury Interactive), IBM (Rational) and Borland (Segue), to name a few.

Today, the perception is that automation is easy, or at least it should be. With ETA it is now a reality. For more information on the most innovative test automation infrastructure to hit the market in 30 years, please contact sales@etaliq.com.

Detailed Solutions

Test Plan and Case Productivity Enhancements

Test Plan documents contain individual Test Cases which define attributes that describe their contents. Test Plan chapters are arranged to categorize Test Cases into groups. In these groups, the Test Cases often have common attributes, setup requirements and general objectives. This provides an opportunity to create two types of automated tests where chapters are Test Groups that have attributes, setups and cleanups that apply to all Test Cases, and Test Cases have their own attributes, objectives, setups, steps and cleanups. Using a hierarchical arrangement of Test Groups and Cases within the automation presents optimization possibilities in both execute time and lab resources usage. Common setups and cleanups can be executed once for a series of Test Cases instead of repeating for each, saving execute time on valuable lab resources.

Within ETA, Test Plans are hierarchies of groups, sub-groups and Test Cases that are arranged to efficiently use Lab Resources and optimize execution time.

Test Case attributes, sections, and verdicts, within the ETA automation tool, each provide their own specific productivity enhancements

to the overall automation infrastructure. These include:

Test Case (TC) Name or ID:

- Uniquely identifies each individual Test Case within product testing,
- Used for execution status and result tracking,
- Structured to include a hierarchy or categorization within its name in order to provide a representation of groupings of test cases.
  For example, TC ID "`L2VPN_VLAN_ErrorStatistics_1`" represents the membership in L2VPN macro group of tests, in the VLAN-related sub-group of tests, and finally in the Error Statistics minor group of tests of which it is the first "1" test. These groupings often match the chapters within a test plan.

TC Title:

- Briefly represents the test title in written form.
  For example, "L2VPN VLAN error statistics verification during gigabit Ethernet error injection"

Author:

- Name and/or User ID of the Test Engineer Subject Matter Expert (SME)

Date:

- Date when Test Case was originally written or most recently modified

Keywords:

- A list of technology words that further represent categories of what is performed within the Test Case.
- These keywords can be used to filter specific result summaries and reports from the database.

Attributes:

- A list of Unit Test Plan ID's that provides a cross-reference to the Developer written test plans and test cases. This is

useful in cross-reference to sub-system related code and test cases.

- These attributes can be used to filter specific result summaries and reports from the database.

Objectives:

- A detailed description of the requirement that is being addressed in the TC. It often includes a brief description of the setup upon which the test must be executed as well as a summary of the steps and the verifications to be performed.

Common:

- This section contains ETA verb language code that describes any common definitions that are used throughout the Test Case. Conditions may also be included to, for example, ensure that the test case does not execute against setups that do not meet its resource requirements. Another example condition is one that sets the link speed, scalability or other aspects of the test to desired levels for the hardware found to be present during execution.
- Common Verdicts:

  NA: This verdict, not applicable, is set for this section and Test Case when any Prerequisite Condition is not met. For Example, if the Interfaces to be tested are *not* VLAN, then assign the **NA** verdict. When this verdict is set, *do not* continue with the execution of this TC, thereby saving lab resource time.

  PASS: The Test Case prerequisites and common objectives have all Passed.

  FAIL: One or more of the common objective definitions have failed. When this verdict is set, *do not* continue with the execution of this TC, again saving lab resource usage time.

Setup:

- This section contains comments and ETA verb language code describing the set of steps necessary to put the test lab into a state where the detailed steps can verify the objective.
- A failure to properly achieve the lab setup state specified means that any further execution of the TC would not be useful. In this case do not execute the steps section and proceed directly to the TC cleanup section. This again saves valuable execute time on lab resources.
- Section Verdicts:

  PASS: The test lab has achieved proper prerequisite state for the execution of the TC.

  INCOMPLETE: If this occurs, *do not* execute the TC detailed Steps, again saving valuable lab resources. This Verdict should be noted upon failure to achieve a positive setup state. Test Cases with this verdict should not be used in reports showing the overall project Pass/Fail ratio reporting. In actual fact, the test case should still appear in the list of Test Cases that have not yet executed when reporting on the percentage of test cases currently executed.

Steps:

- It is important to note that this is the only section that verifies the objectives of the test. The common, setup and cleanup sections are used to prepare the lab for a test or to return the testbed to a pre-test state.
- This section contains comments and ETA verb language code that describes the detailed steps to be followed in order to verify the objectives of the TC.
- Section Verdict:

  PASS: This section verdict is applied when all steps in this section have passed, have verified the test objectives' conditions, and test results are as expected.

FAIL: This section verdict must be applied if any step in this section fails.

Both the **PASS** and **FAIL** verdicts are results that must be used with the summarized reporting feature to show pass/-fail results.

Cleanup:

- This section contains comments and code that describe the detailed clean up steps to be followed in order to return the test lab to its original condition prior to the execution of this TC. This is done so that any test can be executed independent of any other test in the test plan.
- Section Verdicts:

  PASS: The test lab has achieved proper clean up conditions where it has been returned to original pre-test state.

  UNCLEAN: This section verdict must be applied if any step in this section fails. Upon failure of a cleanup section it must be noted that any subsequently executed test case is suspect as to its pass or fail result. A failure to return the testbed to a clean state may negatively affect any subsequent TC.

Automation that is created within a simple written Test Plan and Test Case structure, that includes the original SME written Setups, Steps, Cleanups and Attributes as well as easy to use Automation Code itself, provides several advantages. This document can be written in any standard off-the-shelf word processor. Just save the document in standard ASCII text format, import it to ETA, and that is the automation. It's as simple as that! Advantages include:

- Reduced possibility of mismatch between the written TC definition and the code;
- Extending the ISO product development processes into the automation world;
- Consistent documentation rules for all test plan documents and the automation;

- Product Development Engineers review the automation code and the test plan document;
- Being available as training material for Junior Test Engineers with real commands, responses and verifications included; and
- A unique ability to reduce the replication of test cases within the various test plan documents throughout the product document database.
  Note: Today's Test Engineers often copy test cases from previously released test plans into their own, in order that their test document be self contained. This results in mass duplication of tests throughout an organization, contributing to a whole different set of inefficiencies.

ETA's ability to integrate automation into the Test Plan document provides a full set of advantages over other architectural choices. Organizations can choose to use all or just a portion of this optional feature.

Advantages in using ETA's optional integrated Test Plan and automation feature to its fullest means: Test Automation that consistently continues to match the Test Plan written objectives, Development and Test Engineers review the Test Plan document (and thus the automation) thereby extending the ISO process to cover automation. Additionally, Test Plans can be written to support multiple product versions in order to reduce the need to copy/paste and duplicate test cases amongst many documents and databases.

When the Test Plan and Automation, as specified above, are loaded into an integrated automation application, the advantage of database classification is added. Classification of plans, groups and cases, including all of their related attributes provides significant advantages when reporting product and project status to management. Meaningful data abounds. Creation of additional execution-related attributes during or post-execution provides even more valuable data when reporting on the status and/or quality of a project or product.

Many Test Plan, Group and Case attributes are available during execution pass/fail reporting for filtering and summarization purposes within ETA. Additional runtime attributes can also be added for tracking purposes during execution. All of this provides an unparalleled flexibility and accuracy of reporting when selected by management to track product status and quality, or to report on lab resource usage.

"Send, Receive, and Verify!" Simple, As it Should Be

Simplified Send, Receive, and Verify in the most efficient manner possible, is the objective. ETA provides a unique set of verb language verbs and nouns that are designed to meet that objective, perfectly. They pinpoint the action and definition requirements used while testing, whether it's automated or manual. ETA's simplified verb language functions combine to read very much like the traditional Test Engineer steps, as written in the Test Case steps.

Take the following very simple Test Plan written steps for example:

1. An enabled connection is defined as interface state up, line protocol up, and connection state enabled.
2. "show interface status" and verify that the connection is fully enabled.

Using traditional Tcl and regular expression parsing tools, you would need to call a procedure to send the command, parse the returned response, and use a series of switches and conditions for each individual expected result element. In addition, you need to verify that no errors exist on each of the above steps as well as provide associated positive and negative logging statements. In total, you would need more than 50 lines of code using the traditional approach.

Now, look at the same steps of actual automated code using ETA verb language commands:

```
RESULTLIST(EnabledConnection)
  "interface state = up"
  "line protocol = up"
  "connection state = enabled"

SEND NodePE1 "show interface status" EnabledConnection
```

These steps use two of the high level commands that are central to ETA's success and ease of use.

Command 1: defines of an *expected result list*, "`EnabledConnection`", containing three attributes and their associated expected values. This list is available for use anywhere in subsequent commands.

Command 2: performs a **`SEND`**, to a specific device ("`NodePE1`"), of a user-defined command (between the quotes), and associates the resulting output for verification against the previously-defined expected result list "`EnabledConnection`".

In a nutshell, the "**`SEND`**" above, is "Send, Receive, and Verify!", all integrated into one. It performs the send of the user-defined command to "`NodePE1`", retrieves the response and parses it, using powerful proprietary attribute recognition technology, and verifies all attributes against the expected values listed in "`EnabledConnection`". In addition, it performs a series of additional features on behalf of the user. **`SEND`**, like all verb language commands in ETA, includes; writing to all associated Logs and Reports, updating of step, section and test verdicts, checking for error and warning conditions, and reporting Pass/Fail not only on the **`SEND`** command itself, but on each element of the expected result list as well. The errors and warnings conditions that are checked include those returned from or on behalf of "`NodePE1`" (e.g. timeout, no response, unavailable, unauthorized, rejected command, alarms, etc.), as well as non-existent attribute, invalid mathematical or conditional expressions, and many more.

ETA's unique generic parsing is a hyper-productive feature. The left side of an expected result element is an attribute that is expected to exist in the device response. The right side of each element is its expected value. Between the attribute and the expected value is an operator. Traditional in-house built tools as well as many of the off-the-shelf variety require that Automation Engineers build com-

plex, highly syntax-intensive algorithmic expressions to parse the attributes and values from a command output, and then, in separate functions, create code to error check, verify and report on each step and element.

Automation that reads near identical to Test Case written steps is invaluable at any time in the test cycle. Automation infrastructure that performs all logging, error checking and reporting functions on behalf of the automator enhances productivity tremendously.

ETA is full of advanced, integrated, easy-to-use verb language commands that completely, and perfectly, simplify the automation process.

Integrated Code Reduction Features

Several ETA verb language features have been implemented specifically to reduce the amount of code necessary to accomplish any given task. As you've seen above, these include fully integrated command-level error checking as well as Reporting functions for all detailed, console, error, and summary logs. This section describes a couple of the other most popular features in ETA that reduce code size and complexity.

Generic Parsing and Verification

ETA's unique generic parsing and verification functionality is a hyper-productivity feature that enhances readability, reduces code length, and dramatically simplifies code syntax and structure.

Traditional in-house built tools, as well as many of the off-the-shelf variety, require that Automators to build complex, highly syntax-intensive algorithmic expressions to parse the attributes and values from a command result. In addition, for each Send, Receive and

Verify function, they need to write additional code for error checking, attribute existence and expected value verification, as well as all of the related logging and verdict control. Traditional infrastructures, using Tcl/Perl/Expect/RegExp utilities, would require 10 to 20 times more lines of code to perform functionalities similar to ETA. Even with that amount of code, many of the productivity enhancement features would not be included, such as node usage reservation and tracking, alarm detection and parsing, and the highly productive log review features, etc.

There is no longer a need to allocate large blocks of time to Automation Engineers to create, debug and support, highly syntax-intensive parsing and verification code. ETA provides a fully integrated "Send, Receive, and Verify!" functionality.

ETA Command: Result List

In order to emphasize the benefits of the previous **RESULTLIST** example, consider the expected result element "connection state = enabled". The left side of the expected result element is an attribute that is expected to exist in the device response, which in this case is the two-word attribute "connection state". The right side of each element is its expected value, which in this case is "enabled". Again from the example, between the attribute and the expected value is the equality operator, "=".

ETA, in one easy to read element, now has all of the information necessary to parse, error-check, and ultimately verify an attribute's existence and its expected value. No additional code is required for error checking, logging and verdict control. Readability is dramatically improved.

ETA Command: Wait

Test step definitions often require that the tester "wait" for a specific condition to be true prior to proceeding with the next step in the Test Case. A specific test case example is, "wait for a Processor—or Card, or an entire Node—to reload and to fully recover to an active state, and then verify all connections are again in an enabled state". This type of step is frequently required during failure and recovery testing. Another specific Test Case step example is, "wait for an interface to return to an enabled state". It is specifically for these testing related patterns that ETA includes the **WAIT** command.

> **ETA includes a set of easy-to-use functions that focus on what a tester needs to do. WAIT is just one of those commands. In one simple command, the test will wait for a condition to be true, prior to proceeding.**

WAIT is effectively a **SEND** command on steroids. It includes a looping element that repeats the full **SEND** with verification according to an interval, and a full timeout capability to stop waiting after a specific period if the expected result is still not met. Look at this example of the same send command, using the ETA **WAIT** command:

```
WAIT 30sec NodePE1 "show interface status" EnabledConnection
    WaitInterval=2sec
```

This command again makes use of the same 3 elements included in the expected result list definition "`EnabledConnection`". It will wait for up to a maximum of 30 seconds for all 3 elements of the expected result to be true. It will repeat the same "Send, Receive, and Verify!" functionality of the **SEND** command every 2 seconds according to the wait interval shown. If at any interval the expected results are verified to be met, the **WAIT** command will pass and the subsequent command will be initiated. This means that if the condition is met after only 10 seconds of elapsed time, then the automation will continue without waiting an additional 20 seconds. If after the 30 second timeout period the interface has not achieved the "`EnabledConnection`" state, the step will fail and the verdicts will be updated accordingly. Adding

this functionality to an already existing set of procedures in Tcl or Perl would require 20 or more lines of additional code. This means that 70 or more lines of code would be required to perform a simple wait for an interface to reach a connection enabled state.

In ETA, it's simple to optimize execute time, reduce code size, and perform complex testing procedures using an expected result definition `RESULTLIST` and a `WAIT` command. Send, Receive and Verify!, in two lines of easy to read code.

In a nutshell, the "**`WAIT`**" above, is "Send, Receive, and Verify! performed within a Loop for a repeating Interval, including a failure Timeout!", all integrated into one. It performs the send of the user-defined command to "`NodePE1`", retrieves the response, parses it, recognizes the individual elements or attributes and verifies these against the expected result values in "`EnabledConnection`". It repeats this until either the expected result is true or a timeout occurs. Like **`SEND`**, it also performs a series of additional features on behalf of the user including; writing to all associated Logs and Reports, setting of section and step verdicts, checking for errors returned from or on behalf of "`NodePE1`" (e.g. timeout, no response, empty, rejected command, etc.), and reports Pass/Fail on each element of the expected result list for each iteration of the **`WAIT`** loop itself.

ETA verb language commands reduce code size and complexity more than 10-fold. They also save time during execution and provide an incredible boost to productivity for all automation tasks whether during creation, execution or maintenance.

ETA Command: Result Compare List

Another of the most popular code reduction features of ETA is again created based on what a tester has to do. It is the integrated Result

Compare List command. A frequently required test step definition demands that a tester ensures that, after error recovery, error statistics stop increasing while normal send/receive counters begin increasing again. It is specifically for this test pattern that ETA includes a **RESULTCOMPLIST** command. It is used to easily verify the difference in attribute values between two **SEND** commands. This command in combination with the **SEND** command (used twice, or in a loop) is implemented with a special version of the Verify capability. Look at this example, using the same **SEND** command and a Result Compare List:

```
RESULTCOMPLIST(UpConnectionStats)
  "collisions      NOTINCREASING"
  "giants          NOTINCREASING"
  "runts           NOTINCREASING"
  "input frames    INCREASEDBYATLEAST   10000"
  "input octets    INCREASEDBYATLEAST   80000"
  "output frames   INCREASEDBYATLEAST   10000"
  "output octets   INCREASEDBYATLEAST   80000"

SEND NodePE1 "show interface status" UpConnectionStats
SLEEP 2 secs
SEND NodePE1 "show interface status" UpConnectionStats
```

This command similarly makes use of a simple, yet highly effective, expected result definition of 7 elements. The difference here is that each attribute value is not compared with the value contained in the node response, but rather to the difference between the first and second node responses. The process is: On the first use of the **RESULTCOMPLIST** in any **SEND**, retain its response for later processing; Upon reuse of the **RESULTCOMPLIST** in a second **SEND** command, parse both responses and verify the expected results against the differences in those responses.

This is yet another example of a **SEND** command on steroids. It focuses on delivering functionality that testers write into the steps of Test Cases. Again to reiterate, the first use of the **SEND** command with a **RESULTCOMPLIST** parses and retains the values of the required attributes. Then, the second **SEND** command parses its attribute values, computes the differences, and verifies the defined conditions.

Again, as with the other examples, coding requirements in Tcl or Perl would be in the order of 100 lines of code.

ETA's unique tester-oriented command language reduces code length from 100 lines to just 4. An incredible 96% reduction in the total number of lines of code. ETA means less code; Less code means less automation time and less errors; Less automation time and fewer errors means significant savings.

Therefore, ETA *means savings!*

This is a simple example of the Result Compare feature within ETA's powerful command language. There are more than 30 specialty conditional operators that are defined to deal with all known cases of attribute comparison.

ETA verb language commands are designed with day-to-day testing tasks in mind, implementing unique time-saving features that target specific tasks that testers do, without requiring complex code for parsing, comparison and error checking code. They provide an incredible boost to productivity for all automation tasks whether during creation, execution or maintenance.

Simplicity: ETA Verb Language

As demonstrated above, ETA's verb language is absolutely easy. Engineers and managers with very limited automation experience can now automate. ETA's total instruction set is 25 commands, less than half of which are frequently used. Manual Test and Product Development Engineers no longer need to be teamed up with Automation Engineers. They can write code themselves, without guidance or assistance with complex syntax, environment setup, coding, or anything else.

The vast majority of automated code written with ETA uses 10 commands. This is in direct contrast to existing in-house and off-the-shelf systems that create 100's or even 1,000's of functions within their systems.

Integrated Environment Setup and Definition

ETA's infrastructure is fully integrated, providing simple, easy-to-use table definitions that allow any user to get up and running quickly. New users are up and operating within minutes. No special environment setup variables, no complex multi-level path statements, no Unix shell scripts or package locating and loading processes. ETA has it all in one; Node definitions, execution parameters, Test Plans and Test Cases, fully integrated. Nothing else is required. Just schedule and run.

Integrated Time-Saving Features

ETA's infrastructure provides several time-saving features that reduce the time to create, debug, execute and review automated TC's. Reductions in time to create and debug not only saves time for Automation and Test Engineers, it also reduces the amount of time test lab resources are needed for unproductive use. Both of these result in significant ROI and saving, but there are more. . .

Pre-execution Code Verification

ETA provides an integrated syntax checking feature. This feature is a pre-execution failure detector, highlighting coding errors such as; undefined variables and nodes, invalid use of defined verb language commands, flow control errors (e.g. loop, condition or scope without end), misaligned or missing quotes, and many, many more. All of

these checks are performed during pre-execution without using test lab resources. Syntax errors are noted in an easy to use integrated logging system. Automation and Test Engineers need not wait for executions to complete to find syntax-related errors.

Simulated Node Code Verification

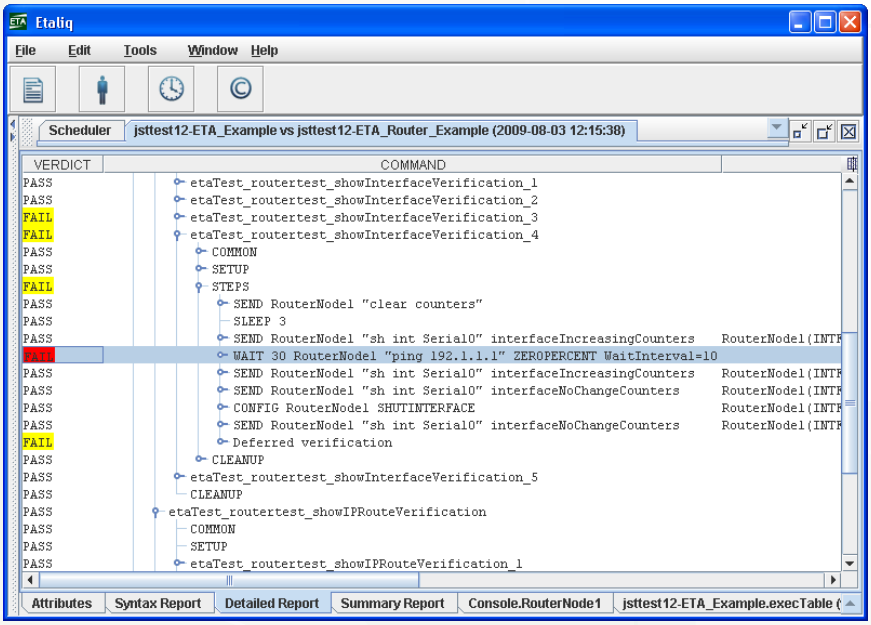
ETA's unique integrated node simulation feature allows code to be verified without using valuable lab resources. This is yet another feature that reduces the amount of unproductive lab resources usage. Another benefit of this very powerful feature is that automation code can now be created and verified much earlier in the Test Cycle. This means that a just-in-time inventory of working automated test cases can be available, ready to run, at the beginning of a test cycle instead of just at the end or during regression.

Integrated Error Verification

ETA's unique integrated error checking feature ensures that a comprehensive consistent approach is implemented for each and every command in all automated test cases. ETA protects against errors during communications to and from each and every node. Errors like timeout, no response, empty, rejected command, session failure and disconnect no longer cause automated scripts to abort. Each individual ETA command is protected during execution against abort and failure conditions. Failures like undefined or empty variables, invalid expressions, or invalid parameters no longer cause script aborts.

Logging Reporting Features

ETA's integrated logging and reporting mechanism provides extensive detailed logging, console logging, and error logging from within the infrastructure; thereby relieving the Automation Engineer of this responsibility. This feature has, in addition to many other benefits, consistency of log detail across all automated executions. The following is a screenshot of the detailed log output from an ETA execution.



As you can see in the figure above, the reviewer can selectively open and close each line in the detailed log to reveal or hide more detailed information. Each individual line contains its verdict which is crucial to quickly determining failure points and causes. This report is the cornerstone of the highly productive Log and Report review feature. It provides additional columns of information, some of which are off-screen above. All ETA execution logs and files are created consistently, using the same features, with the same logging detail, regardless of what test plan is run.

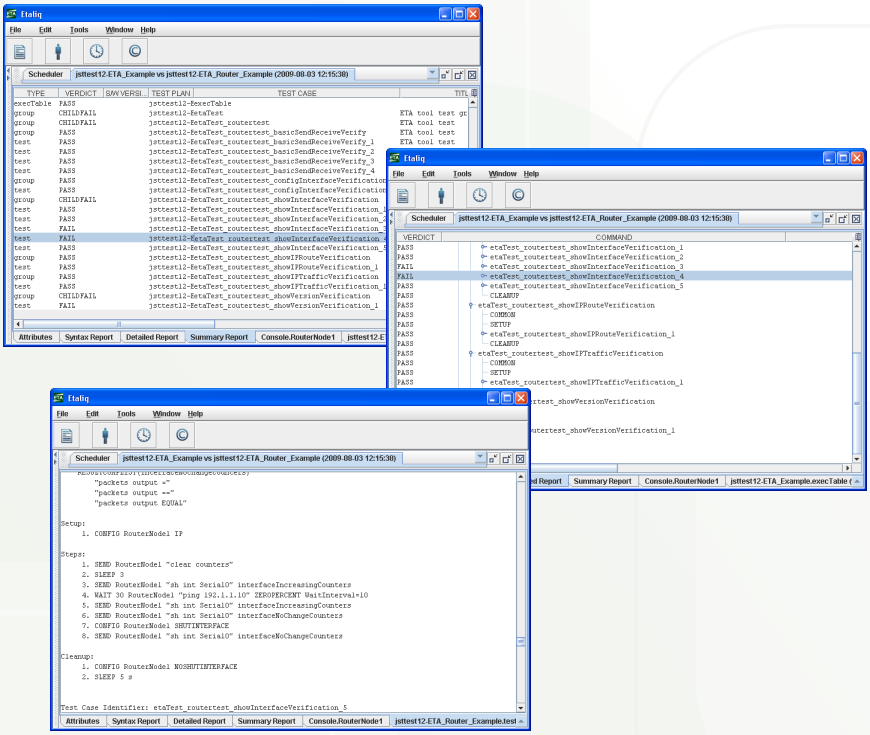
Logging and reporting consistency contributes to enhanced productivity all over the organization. Regression Engineers need to spend much less training and familiarization time when adopting responsibility for new regression tests. Automation Engineers spend no time adding extra debug logging into old style scripts to determine what is going on. Developers, Manual Testers, and even Field Engineers quickly become familiar with the consistent log output which means they will not need assistance from automation experts when reviewing log files and recreating problems.

ETA's integrated logging and reporting mechanism provides real-time updates, while executing. Test Engineers often use this feature to gather more information and do additional problem determination

for test cases that consistently fail. They manually connect to the device that exhibits the failure, start the automation and watch as it progresses to the point where a failure occurs.

ETA's unique fully indexed reports and logs provide the ability to jump from one log or report to another, to the same relative position in the target file. This is a huge time-saving feature when building new automation or reviewing executions. By "Jumping" to the same relative position in a console log, the tester can easily determine if the failing device provided any additional unsolicited information during the failure. Devices will often send unsolicited, failure related, error or alarm messages to the active session. Testers may do a search in a console log for errors or alarms that should not occur. When finding these, they can highlight them and "Jump" to the detailed log, directly to the Test Case code that was being executed when the error occurred.

ETA's unique fully indexed reports and logs are also fully linked to the source code of the Test Cases in the Test Plan. Again, productivity is dramatically enhanced for many within the organization. Automation Engineers can jump directly to a test case to modify the test code when a failure is related to the code within the Test Case.



The relative position jumping feature is represented in the above screenshots, and described here. In the above example, a tester opens an execution from the left tree reviews its logs and reports in full screen. The bottom row of each of the screenshots shows individual tabs for each of the files that are included in this execution. Drawing your attention now to the top-left screenshot where the *Summary Report* is opened, the user clicks to select one of the Test Cases. From that selection point, the tester can select to "jump" to the same relative position in any other file, including the test plan. In this case the tester selects to "jump" to the *Detailed Log*. As you can see from the middle-right screenshot above, what appears is the *Detailed Log*, opened to the same relative position as selected in the Summary Report. During investigation of this failure, the user has opened the tree tab in the *Detailed Log* to show a step in the Test Case and then selects to open the Test Plan while doing further problem determination. The third screenshot, bottom-left, shows that the tester can also jump to the same relative position within the source file where the Test Plan has been opened to the exact Test Case.

ETA provides yet another totally unique productivity tool, where logs and reports are linked to each other and to the origin source code in the Test Plan.

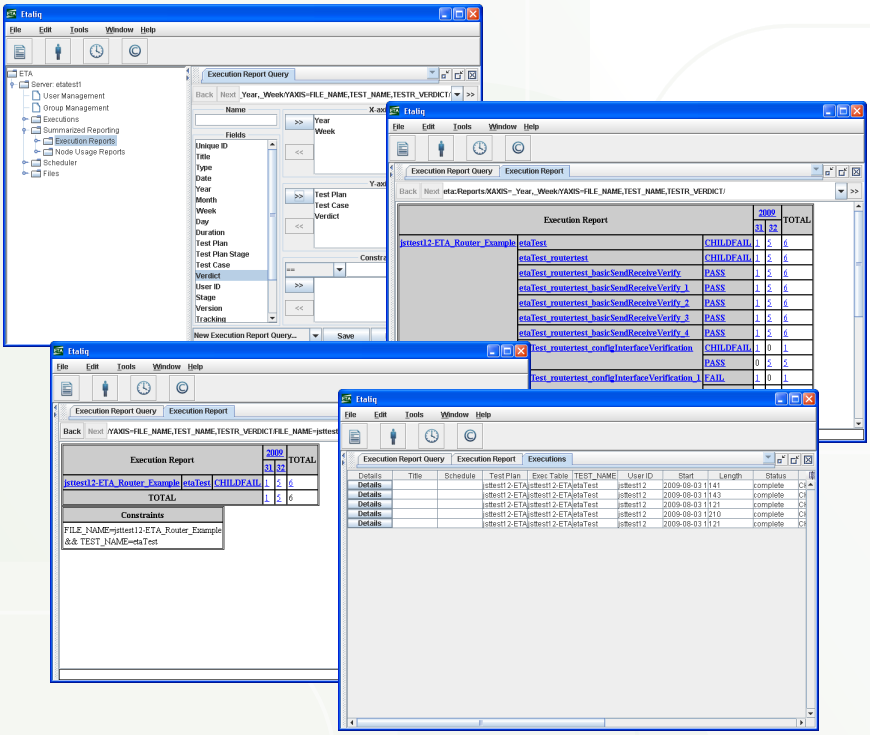
Managers, Test Engineers and Product Development Engineers can easily review the objectives and steps of the actual Test Case, while inspecting a failure. They can quickly determine if the problem is with the automation code itself or with the device being tested. They can also jump directly from log to log to further investigate device failures while doing additional problem determination.

Summarized Results Reporting

Summarized reporting for individual and merged project branches, and product-specific regression results often require a significant amount of time to generate. Creating summarized pass/fail results daily, weekly and monthly can be a daunting task. Engineers updating spreadsheets for management so that these are quickly available is a waste of valuable time. What if you could fill out a request form specifying exactly what you want to see, and how you want to see it? Here are some questions that are easily transformed into summarized result tables with ETA:

- What feature tests were run to verify a specific bug fix?
- What is the current pass/fail ratio for all tests executed for a specific project or development branch?
- During what execution against what image, did a specific test begin failing?

The sheer number of unique summarized pass/fail reports that can be requested is endless. ETA provides a fully customizable report generation facility to summarize test case results. You can customize your query, get your results in table form, and drill down through these results, arriving at the detailed log within the execution where a failure was detected.



From the left tree, the Manager or Tester selects the Execution Reports tool from within the Summarized Reporting tree. What appears in the first screenshot (top-left) above is the request form that allows the user to completely customize the report by assigning fields to the *X*- and *Y-axis*'s and applying filters/constraints to the request. The resulting report is shown in the second screenshot (top-right) above. The report is in table form showing the verdict results for all selected test groups and cases. This tool supports the concept of drill-down where sub-reports can be selected by clicking on any of the table cells that are underlined in blue. The third screenshot (bottom-left) shows a sub-report that contains the associated results of only the etaTest Test Group. The fourth screenshot (bottom-right) that is shown is a table of all of the executions that were summarized in one of the numeric pass/fail table cells. These executions are also drill-down capable, where selecting a *Details* button in any of the rows results in opening the actual Execution in the Log and Report review tool. You've already seen how reports, logs and source files are all easily selectable through their unique linkages.

> ***Remarkable!*** **From very high level summarized reports, to very detailed failure logs in a matter of seconds. With only a few clicks.**

Further emphasizing the usefulness of this feature, lets put this into the context of a customer-reported defect. When determining why a customer has found a defect, the question is often asked: "What Test Case was supposed to verify this, is it automated, where are the logs, and why was it not detected?" ETA allows you to search back as far as you want through test case results, in order to determine everything you ever wanted to know about what happened. All logs and reports are available through drill-down reporting. Simply double-click on a result and proceed.

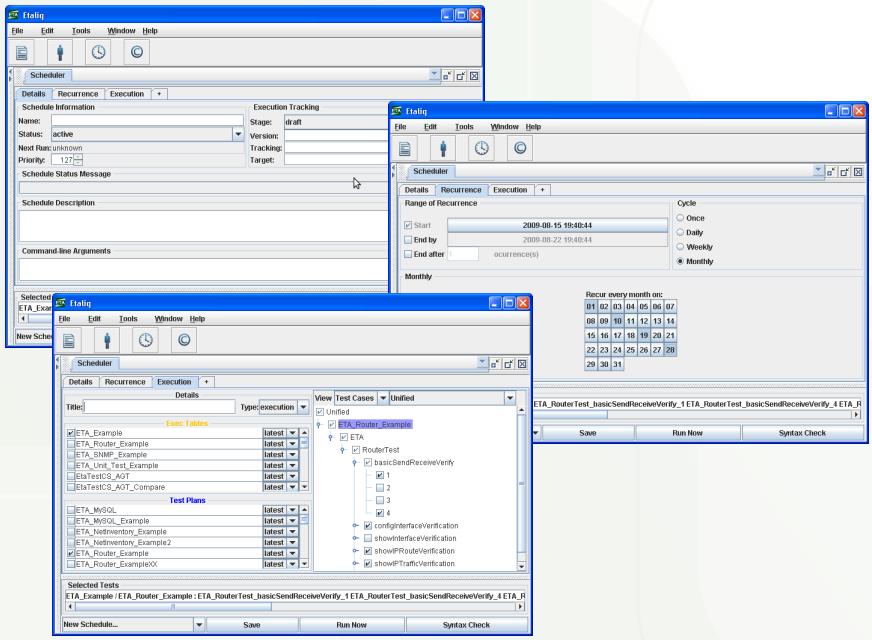
Smart Scheduler

ETA provides a fully customizable smart scheduler to assist you in operating the most efficient automation operation ever.

For example, on a project where new test images are available every day, there is seldom enough time or resources to execute all tests against every image. When this is the case, an effective way to operate is to rotate the tests that run during a project so that all tests run as frequently as possible. Running some of the tests on Monday, Wednesday and Friday, and the rest on Tuesday, Thursday, and Saturday, is easily done with ETA. Additionally, you might choose to run the performance and stress tests on a large scale configuration Sunday or twice monthly.

Another example is where a set of Test Cases for a specific topology or technology, written to be hardware independent, must be run multiple times against various cards and topologies to complete the required testing. Running all tests, against all hardware types, every day or week is likely not possible. Again, with ETA's easy to use scheduler you can set up a complete set of execution schedules

and optimize the use of all of your hardware. ETA makes all of this possible and easy.



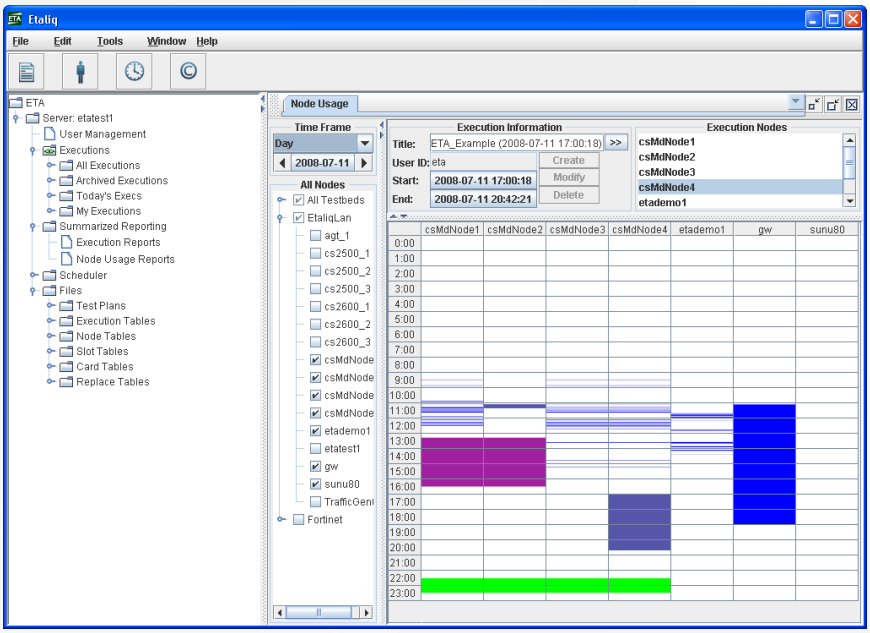
The three screenshots above show that there are 3 components of any schedule. The Details of the schedule itself (top-left), the Recurrence definitions (middle-right), and the selection of the Test Plans and Cases to be run (bottom-left). When selecting those Test Cases to be run, the user also selects the execution parameters files which include the lab hardware information. The bottom line of each screenshot shows that schedules can be saved for scheduled use, run now, or a simple syntax verification only can be selected during code creation and maintenance. This completes the necessary components of a smart scheduler.

ETA allows you to save schedules with detailed descriptions, attributes, and tracking variables. You define the recurrence as required, and select what to execute. A matrix of testbeds with specific execution parameters can each be scheduled to operate a set of unique Test Plans or an individual one. Inside each Test Plan you can uniquely select individual or a combination of Test Cases and Test Groups for execution.

Node Usage Reservation and Reporting

Device usage statistics enable Managers, Testers, and Automation personnel to more effectively use the resources under their control.

Individual nodes or entire testbeds can be locked out of automation and reserved for a specific user within the ETA Node Reservation and Reporting system.



The report shown above is for daily usage tracking of executions and manual lockouts that are complete. Using this same report format, you can scroll forward through the coming days and weeks to determine when activities are scheduled for any of the selected nodes and testbeds. If needed you can adjust your automation timelines within the scheduler to most effectively use your resources.

An additional Node Usage report format is available to view historical node usage statistics for weeks or months at a time, rather than the daily view above. The graphic representation of this second report allows you to quickly identify when testbeds and devices are locked out for manual testing, used for automation, or most importantly, sitting idle.

ETA's Testbed resource tracking feature for reporting and forecasting node usage is another tool, valuable in getting the most out of your test lab resources.

Some clients use the reservation tool to lockout a testbed for use for manual testing during the day, and make it available for automation at night. Setting up ETA and the organization to operate this way is painless, while it makes many more resources available for test automation.

Conclusion

Etaliq's ETA is a truly innovative test automation infrastructure solution that provides the ability for SME Test Engineers to automate themselves, without the need to team up with Automation Engineers. This virtually eliminates miscommunication between Test and Automation Engineers, resulting in more automated tests that do meet test objectives and many less *false pass* automated Test Cases. ETA provides the optional ability to merge the Test Plan with the automation code using its easy to use infrastructure; again, dramatically reducing mismatches and *false passes*. Its' simple to use system based on the principle of "Send, Receive, and Verify!" contributes to increased accuracy, reduced time to create and execute, and, ultimately, to a highly efficient automation operation. Development Engineers review the automation at the same time as they review the Test Plan, thereby extending ISO processes into the automation realm, again contributing to increased accuracy.

The simulator feature enables verification of Test Code validity prior to product availability. This dramatically increases productivity of all automators and lab resources.

The integrated compiler and syntax checker eliminates much of the problem determination time during new test development and existing test maintenance. Again, this dramatically increases productivity of automators and lab resources.

The simplified command structure using verb language and nouns reduces code length while increasing readability. Training, maintenance, support, and code creation times are all dramatically reduced.

The fully integrated smart scheduler feature optimizes the use of lab resources and makes it possible to run more tests, more frequently. The node usage portion of this scheduling system documents resource usage and assists in identifying and rectifying inefficiencies. These features, as well, dramatically increase the productivity of automators and lab resources.

The fully integrated environment setup, device definition and inventory, tool setup and initialization, library and procedure files dramatically reduces the time to create, maintain, and run automated tests. This, in combination with the optimized automation infrastructure, means that more tests can be run, more frequently.

The integrated structured logging and reporting feature with all files linked to each other as well as to the origin source files, dramatically reduces resource effort during problem determination; making all resources more productive. This also provides a side benefit where customer-reported failures can be traced to actual log and source files that should have detected the failure, making it a simple task to update the automation to cover the requirement.

The integrated, customizable, summary reporting mechanism means that more detailed and accurate statistics about product, feature, or individual project quality are available at the touch of a button. No more is it necessary to waste valuable engineering time assigning them to update execution profile spreadsheets and databases.

To date, tools like IBM Rational, HP Mercury, Borland Segue, and others, have been unable to break into this market where complexity rules. At last count, more than 100 companies claim to provide, meet or exceed some or all of the requirements to operate an efficient automation infrastructure for this environment. Additional evidence can be found in the number of patent submissions related to test automation being received by both International and U.S. Patent Offices. This market still searches for a viable solution even after 30 years of evolution in the development, test and operation tools field.

When looking for a new automation infrastructure tool, ETA is the one. Integrated productivity enhancement features for coders, re-

viewers, developers, testers and management; Dramatically reduced code size; Increased test lab productivity; Significantly reduced execution time; More tests per hour; Syntax checking; Integrated linked logs, reports and source files; Highly flexible scheduling controls; Integrated and flexible summary reporting and node usage statistics; All in one—There is no comparison.

Etaliq provides a test automation infrastructure that addresses all of the problems described in Part I, "Send, Receive, and Verify! – Part I: Why Isn't Automation that Simple", as well as more hyper-productivity features. Traditional tools suppliers cannot compete with the efficiencies of ETA and recently established Telecom niche suppliers have missed the point. Simple tests are only a small part of the equation. The point is that, unlike these other suppliers, ETA provides an easy to use infrastructure that applies directly to simple tests as well as scaled performance and stress tests; while positively affecting all resources that have anything to do with testing. Efficiencies and benefits are realized across the entire organization, including hardware and software development engineering, test engineering, including testers, automators, alpha and beta verification engineers, as well as customer support.

With Etaliq's ETA, all product development, test and support organizations can now dramatically increase test lab and resource efficiency, reduce time-to-market, achieve higher quality targets faster and easier, reduce customer-found defects, and, ultimately, bring positive contributions to their corporate reputation. All of this while providing huge ROI savings.

For more information or to request a trial on your site, send us an e-mail at sales@etaliq.com. To try the ETA demo tool over the web using our simple lab, visit `http://www.etaliq.com`, and follow the links to register for the live ETA demo.