WaveAutomation™ Test Automation Framework

User Guide

Version 2.4

October 2007

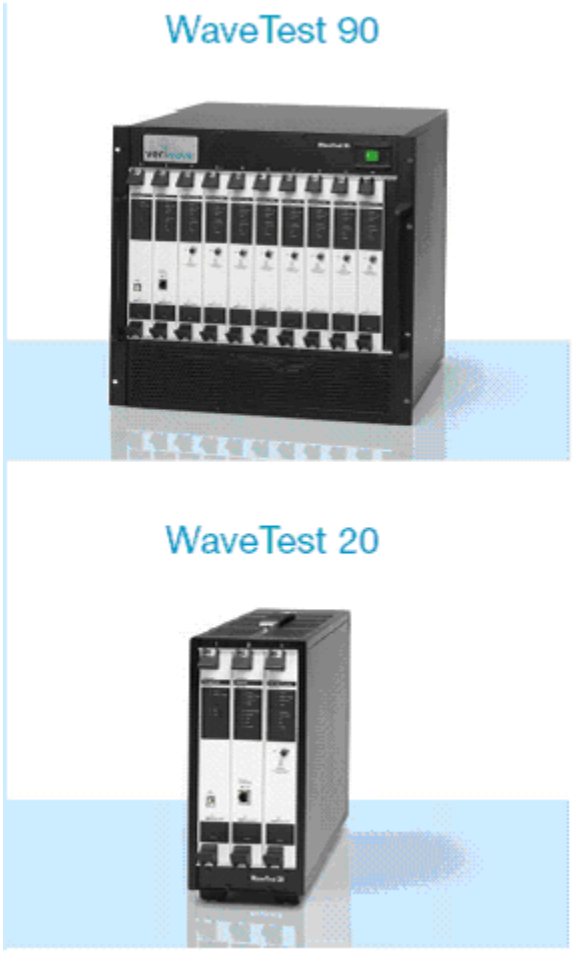
Preface

The preface introduces you to the VeriWave™ product line. You'll also find out how to contact Technical Support, Customer Support and what resources are available for you to get assistance. About this guide describes the conventions used in this manual.

VeriWave™ products

WaveTest™ 90 / WaveTest™ 20 Multi-Client Traffic Generator / Performance Analyzer is designed for equipment manufacturers to accurately analyze the performance of their WLAN infrastructure products, as well as for carriers and enterprise users that want to make the right choice when selecting WLAN equipment for deployment in their networks.

FIGURE 1. VeriWave products



WaveTest™ benefits

Scalability

- Scale WLAN infrastructure equipment testing to realistic scenarios comprising thousands of WLAN and Ethernet clients
- Per client security, power, rate, and payload settings
- Up to nine independent traffic generators / performance analyzers in each WaveTest™ 90 chassis

- Up to six WaveTest™ chassis daisy-chained in a single test

Accuracy

- Traffic generation and analysis at maximum air rate for all WLAN clients; Ethernet traffic at line rate
- Multiple flows per client emulate diverse traffic types including voice and data
- Frame by frame tracking of every client flow with 50 nSec time stamp accuracy
- Precision radio for roaming and rate adaptation measurements
- Individual client mobility control, providing precise and controllable roaming tests

Repeatability

- Sophisticated traffic scheduling emulates real-world scenarios of clients contending for the shared medium
- Regression testing using industry accepted test scripting languages
- WLAN client emulation independent of off-the-shelf MAC and radio constraints guarantees repeatable test sequences

Automation

- VeriWave Command Library (VCL) enables test automation using Tcl, Python and Perl
- Test and results configurations are interchangeable between VeriWave's automated test suites and user scripts

Large Scale Testing

Testing a large scale Wireless LAN has historically been an enormous challenge. With the WaveTest™ 90/WaveTest™ 20, network testing is brought to a new scale:

- Each WaveBlade traffic generator / performance analyzer emulates up to 500 WLAN or 1,000 Ethernet clients across single or multiple subnets
- Each WaveTest™ 90 supports up to nine WaveBlades with a combined capacity of up to 4,500 individual WLAN clients (STAs)
- Unprecedented network scale can be achieved by daisy-chaining up to six WaveTest™ 90 chassis for a total of 27,000 emulated clients

- Generate and analyze traffic between WLAN clients and Ethernet clients/servers, or mobile WLAN clients
- Gigabit Ethernet traffic generation and analysis at full wire-speed
- Generate multiple traffic flows per client, with each flow offering Layer 2, Layer 3 and Layer 4 traffic
- Real-time port statistics, per flow statistics, packet filters, triggers and capture capabilities for precise analysis
- Built-in client mobility allows precise roaming of each and every client between any access points at pre-determined times or power settings

Ease of Use

Responding to the unique demands of developers, Quality Assurance test engineers, pre-deployment labs, and competitive marketing labs, the WaveTest™ offers three fully integrated use-models:

WaveManager™. This user interface offers comprehensive and interactive control of multiple chassis and test ports: mobile-client emulation, Ethernet client emulation, traffic generation and analysis, all from a single, easy-to-use dashboard.

VeriWave Command Library (VCL). Programmatic control, test configuration, statistics, and reporting functions are supported via this common library that is accessible via TCL, Perl, and Python.

Automated tests. These tests address specific applications that require automation and regression, such as roaming, performance benchmarking, VoIP over WLAN, and security.

Configuration data, test control, and results are all seamlessly transferable between all use models. VCL command scripts can be generated automatically in Tcl, Perl, or Python from existing hardware or user configurations. Test results are collected and displayed using an extensive set of predefined counters, user-defined counters, triggers and filters, as well as a 100 MB capture buffer per WaveBlade.

Getting help

Technical support

If this or other available documentation does not resolve your problem, contact technical support at:

Internet: Customer section of www.veriwave.com

Email: support@veriwave.com

Phone: 503-473-8341 or 1-800-457-5919

Technical support hours are 8:30AM Pacific Time to 5:30PM Pacific Time.

For the quickest service, have the following information on hand when placing your call. If requesting help by email, include this information in the email:

- Your company's name.
- A contact person's name, phone number, and email address.
- A detailed description of the problem, including the point in the test when the problem occurred.
- The complete text of the error message, including the error name and number, if available.
- A software version number. To get the version number, click **About** on the Help menu.
- A screen shot of the user interface if the problem leaves it in an unusual state.

Note. *To make a screenshot of an error message or a user interface element, simultaneously press the Alt key and the Print Screen key. Paste the captured image into a document by moving the cursor into the document and simultaneously pressing the Ctrl key and the V key.*

Returning equipment

If you suspect that there is a hardware malfunction with WaveTest™, please contact VeriWave Technical Support. They will guide you through the process of diagnosis and equipment return, if required.

Do not return equipment directly to VeriWave until you have contacted VeriWave technical support to ensure that the return is handled properly.

Customer service

For help with ordering:

Phone: 503-473-8340

Email: customerservice@veriwave.com.

Address: VeriWave Inc.
8770 SW Nimbus Ave.
Beaverton, OR 97008

Other resources

Resource	Where to find it
Installation Guide	■ Printed version shipped with the equipment.
Installation and Maintenance Guide	■ On the CD shipped with the equipment. ■ The Customers section of www.veriwave.com.
WT90/WT20 WaveTest™ Traffic Generator and Performance Analyzer User Guide	■ On the CD shipped with the equipment. ■ The Customers section of www.veriwave.com.
WaveApps™ User Guide	■ The Customers section of www.veriwave.com.
Frequently Asked Questions	■ The Customers section of www.veriwave.com.

About this guide

Conventions used in this guide

Illustrations

Illustrations in this document showing the user interface are representative. They may not exactly match the screens on your system.

Typographical conventions

Warning! *A warning alerts you to a situation that could result in hazards to service personnel, damage to equipment or the loss of data.*

Note. *A note calls your attention to important information or useful tips*

Getting Started with WaveAutomation™

Overview

Many testing groups are caught in a cycle of having to manually test each new release or software patch that comes to them from engineering. The pressure to validate multiple releases simultaneously, with little or no time between releases, can severely strain the stamina of even the most capable testing group.

With WaveAutomation™ from VeriWave, testing groups can run fully automated tests using WaveTest WT-20 and WT-90 wireless test equipment. WaveAutomation makes it possible to replace many of the repetitive, time-consuming tasks associated with testing new software releases. A single test engineer can run hundreds or thousands of test cases with very minimal setup work.

In the WaveAutomation configuration file, you specify a set of pre-written VeriWave test applications to run; select the devices you wish to test; and choose the parameters to use for the tests (frame sizes, offered loads, security methods, channels, PHY rates, etc.). You can execute multiple test sequences with the ability to change parameters on the DUT (device under test) dynamically.

When you launch the test automation, WaveAutomation configures the DUTs as well as the VeriWave WT-90 and sequentially executes all of the tests that you've defined, until all tests have been run. Test results are stored in a user-defined location for easy viewing and analysis.

You can use WaveAutomation to quickly build up a set of sanity tests or regression tests to run against your new access point firmware and software loads – for example, a nightly sanity test to run against the engineering build-of-the-day, or a longer, more exhaustive set of tests to run on release candidate images.

Installation

Download the WaveAutomation software

To begin, log in at www.veriwave.com and download the WaveAutomation software. Choose a version of the automation software that matches the WaveTest chassis firmware and the WaveApps applications software that are currently installed.

For example, if you are currently using Version 2.4.2 of the WaveTest Firmware and version 2.4.2 of the WaveApps application software, you should download version 2.4.2 of the WaveAutomation software.

Installation Requirements

WaveAutomation runs under the following operating systems:

- Linux (2.6.x kernel)
- Windows Server 2003, XP, and Vista

  In order for WaveAutomation to run under Windows, you must have WaveApps and WaveTest installed on your system.

The following packages must be installed on your system:

- Python-2.4
- TCL 8.4
- tclx-8.4 (required for Linux; included with ActiveState TCL for Windows)
- expect (required for Linux; included with ActiveState TCL for Windows)

Install the WaveAutomation software

Installing on Linux systems

1. If the downloaded file has a .gz file extension, first uncompress the downloaded file, using the gunzip utility. For example, if the downloaded file is named `WaveAutomate-2.4.2.shar.gz`, then run this command:

   ```
   gunzip WaveAutomate-2.4.2.shar.gz
   ```

   This will uncompress the downloaded file and place the resulting uncompressed installation program in a file called `WaveAutomate-2.4.2.shar`.

2. Run the installation program. The installation program is a shell archive (a self-extracting shell script) that is executed as follows:

   ```
   sh WaveAutomate-x.x.x.shar
   ```

   where *x.x.x* is the version number of the automation software. For example:

   ```
   sh WaveAutomate-2.4.2.shar
   ```

Installing on Windows systems

- Double-click to extract the WaveAutomation zip archive onto your computer. Verify the destination folder where you want to extract the archive, then click **Install**.

Note. *In choosing a destination folder, choose a folder that is as close as possible to the top of a directory tree. You will need to type this path later when running WaveAutomation.*

- After installing WaveAutomation, you must perform one additional step: turning off Microsoft DEP (Data Execution Prevention).

Note. *This section is only applicable if you are using WaveAutomation to configure an access point. If the vendor type is "Generic", you can skip this section.*

With Windows XP SP2, Windows Server 2003 SP1, and Windows Vista, Microsoft added Data Execution Prevention to help prevent malicious programs from causing damage to your system. This feature has the unfortunate consequence of blocking WaveAutomation from being able to run telnet to configure APs. To run WaveAutomation, you must turn off DEP.

The following pages present two procedures for turning off Microsoft DEP. Perform the procedure that is appropriate for your operating system:

- Windows XP SP2 and Windows Server 2003 (page 14)
- Windows Vista (page 15)

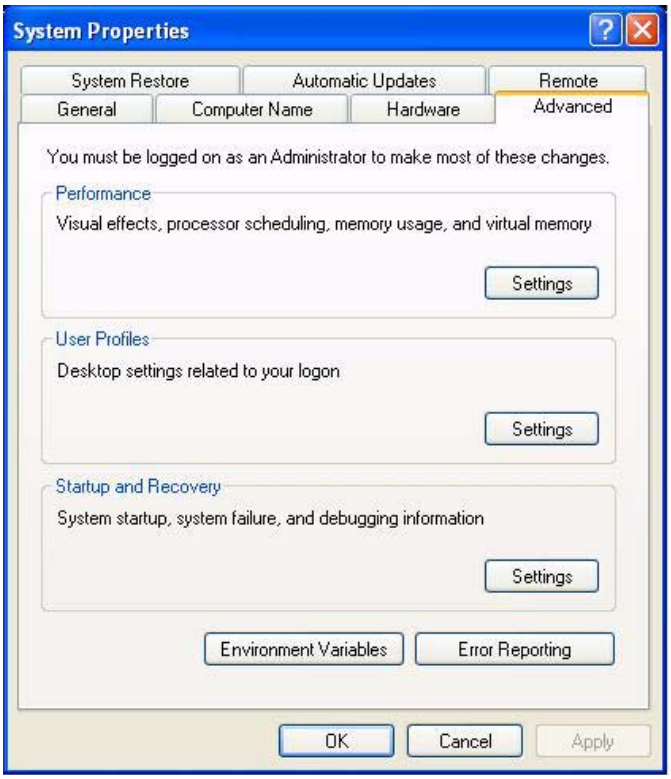
For Windows XP SP2 and Windows Server 2003

To turn off Microsoft DEP, perform the following steps:

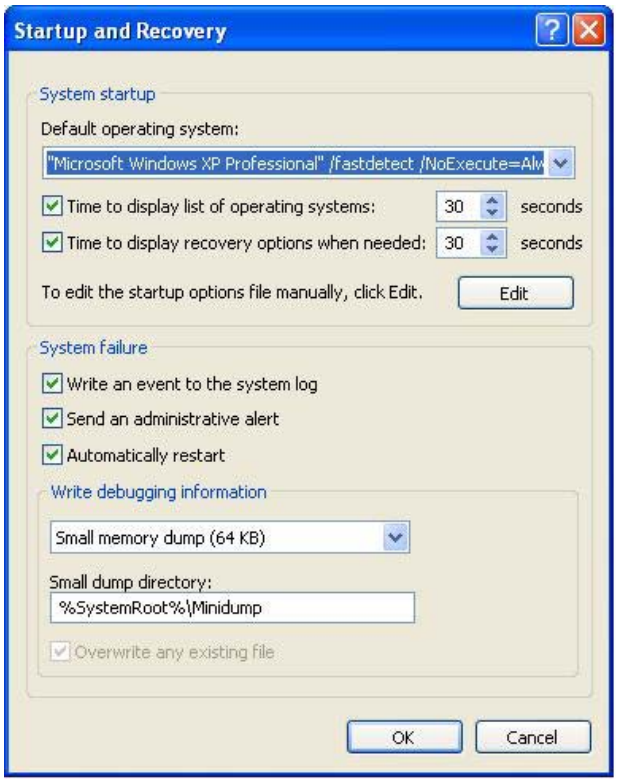
1. Click **Start > Run**, then type `sysdm.cpl` and press Return.

   or

   Right-click on **My Computer** and select **Properties**.
2. In the System Properties window, click the Advanced tab.
3. In the Startup and Recovery section, click **Settings**.



4. In the Startup and Recovery window, click **Edit**.



5. In response, Notepad opens the `boot.ini` file, with contents similar to the following:

```
[boot loader]
timeout=30
default=multi(0)disk(0)rdisk(0)partition(3)\WINDOWS
[operating systems]
multi(0)disk(0)rdisk(0)partition(3)\WINDOWS="Microsoft Windows
   XP Professional" /fastdetect /NoExecute=AlwaysOff
```

Warning! *Be extremely careful when editing this file. Careless mistakes can render your computer unbootable.*

6. Make sure that the `/NoExecute=` option is followed by the token `AlwaysOff`, as shown here. Save the file and exit Notepad.
7. In order for this change to take effect, reboot your computer.

For Windows Vista

To turn off Microsoft DEP, perform the following steps:

1. Click **Start > All Programs > Accessories**.
2. Right-click on Command Prompt.
3. Left-click on Run as Administrator.
4. If you are asked for permission, click Continue.

5. At the command prompt, enter the following text:

```
bcdedit.exe /set {current} nx AlwaysOff
```

6. Enter a Return. You should see this message:

```
The operation completed successfully.
```

In order for this change to take effect, you must reboot your computer.

The installation directory structure

The installation process creates a directory structure that contains the WaveAutomation software and all of the supporting libraries and programs needed for test automation.

Under Linux, the top-level directory is named `automation` and is created in your home directory.

Under Windows, the top-level directory is named `WaveAutomate-`*x.x.x* (where *x.x.x* is the version number) and is created wherever you extracted it.

The top-level directory contains these three subdirectories:

```
apps/
automation/
lib/
```

The `apps` and `lib` directories contain support files that enable the test automation, but are of little direct interest to you. All of the files that you will execute and modify are installed in the `automation` subdirectory.

Make sure you have the correct versions of Python and TCL

Testing compatibility on Linux systems

To verify that you have the correct versions of Python and TCL installed, execute the following script:

```
$ automation/bin/vw_auto.tcl -e
```

If you have the correct versions installed, you should see a display similar to this:

```
checking version compatibility
-----------------------------------------------
os is Linux
tcl version 8.4.9 is fine.
Python version 2.4 is fine.
Expect version 5.41.0 is fine.
Linux version 2.6.11.4-21.15-default is fine.
Invoked as /home/tim/veriwave//automation/bin/vw_auto.tcl
-----------------------------------------------
Compatibility check PASSED
```

Testing compatibility on Windows systems

Go to the installation directory (`C:\WaveAutomate-2.4.2`) and execute this script:

```
tclsh automation\bin\vw_auto.tcl -e
```

As this script executes, it indicates that you are running compatible versions of TCL and Python:

```
checking version compatibility
-----------------------------------------------
tcl version 8.4.15 is fine
Python version 2.4.3 is fine
Expect version 5.43 is fine
Tcl reports platform Windows NT version 6.0 aka:
Windows Vista
It appears you are running on a 32-bit windows box.
-----------------------------------------------
Compatibility check PASSED
```

If the compatibility test does not pass, you will need to check the output to determine which programs need to be installed or upgraded.

Configuring WaveAutomation

To configure WaveAutomation, follow these steps:

1. Change to the directory where the automation configuration files are stored:

   `cd $HOME/automation/automation/conf` (Linux)

   or

   `C:\WaveAutomate-2.4.2\automation\conf` (Windows)

   The `automation/conf` directory includes an example configuration file named `config-template.tcl`, with extensive comments to simplify the configuration process.

2. Make a copy of the VeriWave-supplied template configuration file (you'll subsequently edit the copy):

   `cp config-template.tcl config-`*`myname`*`.tcl`

   For example:

   `cp config-template.tcl config-tim.tcl`

Note. *Under Windows, copy the configuration template* installation_folder*automation\conf\config-template.tcl to config.tcl in the same directory.*

3. Edit your copy of the configuration file using a plain text editor such as vi or emacs.

   `vi config-tim.tcl`

   or

   `emacs config-tim.tcl`

   or

   `pico config-tim.tcl`

   or

   `notepad config-tim.tcl` (Windows)

The following pages provide detailed instructions for editing the configuration file.

The WaveAutomation Configuration File

The WaveAutomation configuration file has four major sections:

- **Global configuration** (page 19) – This section controls global configuration — values that are common to (or used by) more than one test: license keys, chassis address, and more.
- **Group configuration** (page 22) – In this section of the file, you can define source and destination groups. Tests can be run with multiple groups with different security types. Alternatively, you might set up multiple groups with a single test (e.g. roaming) to vary client roaming parameters (dwell time) for fast and slow roamers. Group parameters include SSIDs, number of clients, DHCP, client IP addresses, security methods, and others.
- **Test-specific configuration** (page 26) – This section of the file defines parameters that are specific to each benchmark test. For example, you might want to use one set of frame sizes for a latency test, and a different set of frame sizes for a forwarding rate test.
- **DUT configuration** (page 29) – This section of the file defines the devices (access points) to test, with details about each DUT: manufacturer, model, IP addresses, interfaces, radio types, power levels, WT-90 port to which the access point is connected, etc.

Note. *WaveAutomation configuration files use TCL (Tool Command Language) syntax. You do not need to have any prior knowledge of TCL to write or edit WaveAutomation configuration files. The configuration file is a plain text file that contains TCL variable assignment statements.*

These sections simply represent logical groupings within the configuration file. These groupings are strictly for convenience and do not affect the order in which configuration settings are established.

The following sections provide information to help you examine the configuration template file and adjust parameter values to achieve the type of testing you would like to perform. For details about any of the parameters listed in the file, refer to the *WaveApps User Guide*.

Global Configuration

In this section of the configuration file, you define values that are common to (or used by) more than one test.

You can override global values by setting more specific values in the test-specific section, to control specific test applications.

Chassis Name — Configure the IP address or DNS name of the WaveTest chassis. If you are using a multi-chassis set-up, specify the IP address or DNS name of the primary chassis.

To set the IP address of your WT-90 to 192.168.10.246, you would include the following line:

```
keylset global_config ChassisName  192.168.10.246
```

`keylset` performs a simple variable assignment, setting the value of a named parameter. In this example, the variable named ChassisName is being set to 192.168.10.246. The argument `global_config` indicates the section of the configuration file where the ChassisName parameter is set.

Benchmark List — Configure one or more benchmark tests to run. You can use WaveAutomation to run the same benchmark and applications that you can run using WaveApps.

The following table lists all benchmark tests available at the time of this release. This list is subject to change in future releases. For the most up-to-date list, examine the configuration template file `config-template.tcl`.

IEEE 802.11.2	unicast_latency
	unicast_max_client_capacity
	unicast_max_forwarding_rate
	unicast_packet_loss
	rate_vs_range
	unicast_unidirectional_throughput
WLAN Roaming	roaming_delay
VoIP QoS	qos_capacity
	qos_assurance
	qos_roam_quality
TCP	tcp_goodput
AAA	aaa_auth_rate
Wireless Mesh	mesh_latency_aggregate
	mesh_latency_per_hop
	mesh_max_forwarding_rate_per_hop
	mesh_throughput_aggregate
	mesh_throughput_per_hop

If you specify more than one test, the test names in the list should be enclosed in curly braces and separated by whitespace, to conform to TCL syntax. For example:

```
keylset global_config BenchmarkList {
  unicast_latency
  unicast_packet_loss
  unicast_unidirectional_throughput
}
```

Note that lists (enclosed within curly braces) are complete TCL definitions; you cannot comment in the middle of them. The following definition (which contains an invalid comment) would generate an error:

```
keylset global_config BenchmarkList {
   unicast_latency
   unicast_max_forwarding_rate
  # unicast_packet_loss
   unicast_unidirectional_throughput
}
```

License Keys — If you will be running tests that require license keys (for example, roaming and QoS tests), you must specify them in your configuration file. If you have two license keys, you would specify them like this:

```
keylset global_config LicenseKey {
   12345-67890-12345
   98765-43210-98765
}
```

Direction — Define whether the tests should run with Unidirectional or Bidirectional traffic. For example:

```
keylset global_config Direction { Unidirectional }
```

Client Groups — Define the clients to be created by the WaveTest chassis. WaveAutomation groups correspond to wireless client groups in WaveApps. The most basic test includes a single wireless and a single ethernet group:

```
keylset global_config Destination { ether_group_c }
keylset global_config Source      { wireless_group_a  }
```

The name of each group is a pointer to another TCL variable list; you'll define the groups shortly in the Group Configuration section (page 22).

Logs Directory — Set the directory under which to store test result logs. If you provide a full path, enclose it in double quotes:

```
keylset global_config LogsDir "/var/log/veriwave"
```

Alternatively, you can specify a path relative to where the automation code is installed:

```
keylset global_config LogsDir [file join $VW_TEST_ROOT
   results]
```

If $VW_TEST_ROOT is not specifically set in your environment, it is set to one directory higher than the running `vw_auto.tcl` program. For example, if you installed WaveAutomation in `$HOME/veriwave/automation`, and you specified a relative path (as shown above), your result files would be written in a set of directories under `$HOME/veriwave/automation/results`.

Number of Trials — Specify the number of trials to run for each unique test configuration. (You can override these values for individual groups or tests.)

```
keylset global_config NumTrials 1
```

Trial Duration — Specify the duration (in seconds) of each trial. You can override this setting for individual groups or tests.

```
keylset global_config TrialDuration 10
```

Loss Tolerance — Specify the percentage of packet loss that is acceptable in order for a test to be considered as passing. The default (0.0) means that in order for a test to be considered as passing, zero packet loss must be seen. This parameter is used by tests that perform a binary search, such as throughput. You can override this setting for individual groups or tests.

```
keylset global_config LossTolerance 0.0
```

PHY Rate — Define the maximum PHY encoding rate for data and management frame types. You can override this setting for individual groups or tests.

```
keylset global_config DataPhyRate 54
keylset global_config MgmtPhyRate 6
```

ARP Discovery Parameters — In general, you can leave the ARP discovery parameters commented out, so that the system uses default values. If you need to adjust the ARP parameters, simply un-comment the lines and set the values as needed.

```
#keylset global_config ArpNumRetries  5
#keylset global_config ArpRate        10
#keylset global_config ArpTimeout     30
```

Group Configuration

This portion of the file controls group configuration -- values that are common to all benchmark tests. You can override group values by setting more specific values in the DUT Configuration section (page 29).

Note. *The name of the group must match a name that you defined in the Global Configuration section (page 21).*

Any values defined specifically for a group take precedence and override the default values that are defined for a given test.

Defining a group

You can define a group with a single definition, using curly-brace (list) syntax:

```
set group_w {
    { GroupType           802.11abg }
    { NumClients          1 }
    { Dut                 sample-3com-thick-8760-ap   }
    { AssocRate           10 }
    { AssocRetries        20 }
    { AssocTimeout        20 }
    { DHCP                Enable }
    { Gateway             10.10.251.1 }
    { BaseIp              10.10.10.50 }
    { SubnetMask          255.255.0.0 }
    { IncrIp              0.0.0.1 }
}
```

Alternatively, you can set and override values by using the more verbose TCL "keylset" notation. For example:

```
keylset group_w GroupType     802.11abg
keylset group_w NumClients    1
keylset group_w Dut           sample-3com-thick-8760-ap
keylset group_w AssocRate     10
keylset group_w AssocRetries  20
keylset group_w AssocTimeout  20
keylset group_w DHCP          Enable
keylset group_w Gateway       10.10.251.1
keylset group_w BaseIp        10.10.10.50
keylset group_w SubnetMask    255.255.0.0
keylset group_w IncrIp        0.0.0.1
```

Although the keylset notation is a little more cumbersome to read, it can be easier to use. With keylset, you can disable individual lines by commenting them out. By comparison, TCL syntax does not permit you to comment out individual values within a group definition.

Here is an explanation of some of the values most commonly defined in the Group Configuration section. Not all of these values are shown in the example group definitions. For more about the values that you can define in this section, see the example configuration template file.

GroupType — Determines the type of client for this group: `802.11abg` (for wireless groups) or `802.3` (for ethernet groups).

SSID — The SSID to be used by the WaveTest clients when connecting with the DUTs. For example:

```
keylset wireless_group Ssid veriwave
```

DUT List — The names of the DUTs that this group is attached to. You should give each DUT a unique name that designates the access point or controller to which this group will send its traffic. This helps WaveAutomation know how to configure the DUT and the port to which it is connected on the WaveTest chassis.

Note. *For each DUT in this list, you must provide configuration information in the DUT Configuration section of this file. For example, if you specify a DUT named* `tims-ap`*, you must define a device named* `tims-ap` *in the DUT Configuration section (page 29). The name of each DUT in this list must match the corresponding name in the DUT Configuration section.*

Security Methods — You can specify a list of security methods to test for each group. For example:

```
keylset wireless_group_a Method WEP-Open-40
keylset wireless_group_b Method WPA-PSK
```

The configuration template file provides examples that show all of the possible security types that you can specify. To meet your specific testing needs, you can customize the configuration of any of the parameter lists in the configuration file.

You can choose to run one or more (or even all) of the supported security methods in a single automated test run.

If you specify more than one security method, be sure to use whitespace to separate the names in list. For example:

```
Method {
  None
  WPA2-Open-40
  WPA2-SharedKey-128
}
```

Depending on the list of security methods that you are using in your tests, you may need to add or change other settings in your group configuration — for example, WEP and pre-shared keys, the location of security certificates, etc. For further details, see the comments in the configuration template file.

Channels — There are several ways to define a group of radio channels to test. To specify a single channel:

```
keylset wireless_group_b { Channel { 3 } }
```

To iterate over a set of channel numbers, enclose the list of channel numbers in curly braces:

```
keylset wireless_group_b { Channel { 1 3 5 7 } }
keylset wireless_group_a { Channel { 36 40 44 48 52 56 } }
```

Alternatively, you can define the radio channels using range notation, which has the following syntax:

```
keylset ... [range <start> <end> <step>]
```

The following example cycles through all of the channels from 1 through 12 (inclusive), using a step value of 1:

```
keylset wireless_group_b Channel [range 1 12 1]
```

Note that range notation uses square (rather than curly) brackets. Take care not to confuse [ ] and { } when editing your configuration file. Remember to enclose lists in curly braces, and to use square brackets when invoking functions such as range. The sample configuration file includes extensive comments to reinforce this distinction.

To test the channels in a different order each time you execute this set of tests, you can specify a randomized list:

```
keylset wireless_group_b { Channel [randomize_list [range
   1 12 1]] }
```

NumClients — The number of clients that the WaveTest will create for the group.

DHCP — To use a DHCP server to assign client IP addresses, set this variable to Enable.

```
keylset group_x DHCP Enable
```

For static IP addresses, you must set the following parameters:

```
keylset group_w Gateway     10.10.251.1
keylset group_w BaseIp      10.10.10.50
keylset group_w SubnetMask  255.255.0.0
keylset group_w IncrIp      0.0.0.1
```

If both DHCP and static IP addresses are defined, DHCP is used.

Multiple access points — To configure WaveAutomate to use more than one access point in roaming and mesh tests, use the `AuxDut` variable. All access points listed here will be configured identically to those specified in the `Dut` variable.

```
keylset wireless_group AuxDut { tims-ap2 tims-ap3 }
```

You can override settings (for example, the radio channel to use on a specific auxiliary DUT) in the DUT configuration for each device.

Configuring an ethernet group

Thus far, we've looked at the process for configuring a wireless group. Configuring an ethernet group is much simpler; you only need to define a few variables. For example:

```
set ether_group {
{ GroupType    802.3 }
{ NumClients   32 }
{ Dut          tims-ap }
{ DHCP         Enable }
}
```

Configuring groups in external files

Because group definitions tend to be more stable than the rest of the configuration file, you might choose to manage group configurations as one or more external files, which you then include into this configuration file. This approach allows you to share definitions among multiple groups, with the additional benefit of making the configuration file more manageable and less intimidating. Any group definitions in the main configuration file override those in the external files.

After you have created a group definition file, you can load it by including a line of the form:

```
cfg_load_module group wireless_group_w
```

The `group` command understands the WaveAutomation directory structure, and loads the group definition file `wireless_group_w.tcl` from the .../`automation/conf` directory.

Alternatively, if you need to have a common group configuration across multiple installations, you can use the TCL source command to explicitly specify the path of the included file. For example:

```
source /usr/local/etc/wireless_group_w.tcl
```

or

```
source ~/wireless_w.tcl
```

Test-Specific Configuration

This portion of the file defines values that pertain to individual tests. You can override these values by setting more specific values in the DUT Configuration section of the file (page 29).

For each benchmark test, the configuration template file provides test-specific default settings that you can modify as needed. For example:

```
keylset myLatency Benchmark unicast_latency
keylset myLatency FrameSizeList {  511 1492 }
keylset myLatency IntendedLoadList {  100 500 }


keylset myPacket_loss Benchmark unicast_packet_loss
keylset myPacket_loss FrameSizeList { 511 1492 }
keylset myPacket_loss IntendedLoadList {  100 500 }
```

The test configuration name (`myLatency`) serves as a reference to the associated benchmark test (`unicast_latency`).

For a more complete list of settings that you can define in this section of the configuration file, see the comments in the configuration template file. You may also find it helpful to refer to the test descriptions in the *WaveApps User Guide*.

FrameSizeList — Defines the frame sizes to be used in each test. You can use list notation to explicitly list each frame size that you would like to test (like this):

```
keylset myLatency FrameSizeList { 64 128 256 }
```

Alternatively, you can set frame sizes by using range notation like this:

```
keylset myLatency FrameSizeList [range 64 256 64]
```

This range notation generates a list of values for a range that starts at 64, ends at 256, and has a step of 64. This is equivalent to:

```
keylset myLatency FrameSizeList {64 128 192 256}
```

To test all possible frame sizes between 64 and 1024, inclusive:

```
[range 64 1024]
```

To test all possible *even* frame sizes from 64 and 1024, inclusive, specify a step of 2:

```
[range 64 1024 2]
```

IntendedLoadList — Defines the list of intended loads. The intended load is expressed in frames/second and is applied at the port level. The port load is divided equally between all source clients on a port.

As with the frame size list, you can use list notation and range notation interchangeably.

Search Resolution — The search resolution (default = 0.1) determines how precise the search for the final test result needs to be. For instance, a value of 0.1 means that the search will stop if the current result is within 0.1% of the previous iteration result. Keep in mind that specifying a higher precision will increase run times.

Configuring multiple runs of the same test

To run two separate versions of a test, each with different settings, create two test configurations.

```
keylset latency1 Benchmark unicast_latency
keylset latency1 Frame Custom
keylset latency1 FrameSizeList { 1518 }
keylset latency1 ILoadList { 100.0 }
keylset latency1 ILoadMode Custom
...
keylset latency2 Benchmark unicast_latency
keylset latency2 Frame Custom
keylset latency2 FrameSizeList { 1518 }
keylset latency2 IntendedLoadList { 200.0 }
keylset latency2 IntendedLoadMode Custom
...
```

In the Global Configuration section (page 20) of your file, you'll need to also configure the BenchmarkList variable to run the multiple test runs:

```
keylset global_config BenchmarkList {
   latency1
   latency2
}
```

Configuring options for roaming tests

Unlike most of the other tests, roaming-specific test options are unique for each wireless group. These options are listed in the configuration template file along with the test parameters. You must define the options separately for each group.

Configuring QoS tests

In order for the QoS tests to run successfully, you must define both voice and background groups in the Group Configuration section (page 22). Here is an example of how you might do this:

```
# Wireless_voice Configuration
keylset wireless_voice GroupType 802.11abg
keylset wireless_voice BssidIndex 0
keylset wireless_voice Ssid "qos"
keylset wireless_voice Dut generic
keylset wireless_voice Method { None }
keylset wireless_voice Channel { 1 }
keylset wireless_voice NumClients 2
keylset wireless_voice TrafficClass Voice
keylset wireless_voice DHCP Disable
keylset wireless_voice BaseIp 192.168.1.50
keylset wireless_voice IncrIp 0.0.0.1
keylset wireless_voice SubnetMask 255.255.255.0
keylset wireless_voice Gateway 192.168.1.1

# Wireless_background Configuration
set wireless_background $wireless_voice
keylset wireless_background TrafficClass Background

# Ether_voice Configuration
keylset ether_voice GroupType 802.3
keylset ether_voice NumClients 2
keylset ether_voice Dut generic
keylset ether_voice DHCP Disable
keylset ether_voice BaseIp 192.168.1.70
keylset ether_voice IncrIp 0.0.0.1
keylset ether_voice TrafficClass Voice
keylset ether_voice SubnetMask 255.255.255.0

# Ether_background Configuration
set ether_background $ether_voice
keylset ether_background TrafficClass Background
```

Configuring mesh tests for BLOG mode

In BLOG mode, you can set one or more wireless cards to generate interference while running a mesh test. This allows you to see how the access point reacts to interference and packet drops in real-world conditions.

To enable BLOG mode for a mesh test, you need to perform two steps.

1. Define your BLOG settings. For example:

```
keylset blog1 Card                        192.168.16.246:7
keylset blog1 Band.1.BinLow                40
keylset blog1 Band.1.BinHigh               614
keylset blog1 Band.1.BinStrikeProbability 25
keylset blog1 Band.2.BinLow                615
keylset blog1 Band.2.BinHigh               1189
keylset blog1 Band.2.BinStrikeProbability 25
keylset blog1 Band.3.BinLow                1190
keylset blog1 Band.3.BinHigh               1764
keylset blog1 Band.3.BinStrikeProbability 25
keylset blog1 Band.4.BinLow                1765
keylset blog1 Band.4.BinHigh               2340
keylset blog1 Band.4.BinStrikeProbability 25
```

2. Reference these settings for each test in which you wish to use them. For example:

```
keylset mesh_latency_aggregate Blog blog1
```

DUT Configuration

After you have automated WaveTest, you must specify the configuration of each DUT that you wish to test. The DUT Configuration section must provide configuration information for every DUT listed in the Group Configuration section (page 23).

By default, DUT configuration information is inherited from global settings and client group settings. For example, you don't generally need to redefine security methods, SSIDs, channels, PHY rates, or WEP keys for the DUT — those values are defined in the global and group configuration sections.

Any DUT-specific parameters override inherited parameters. For example, in the case of roaming, you may need to override settings (for example, the radio channel to use on a specific auxiliary DUT) in the DUT configuration for each device (see page 25).

WaveAutomation supports the following access points:

- Cisco IOS
- Foundry IronPoint-200
- Cisco LWAPP
- Symbol
- 3Com
- Trapeze

The .../`automation/conf/dut` directory provides examples of how to define vendor-specific settings.

1. Find an existing sample entry for the vendor and model of the access point that you would like to configure. Let's say you want to add a new Cisco AP to your list of available DUTs.

2. You can use the cisco-1232 DUT definition (in the .../automation/conf/dut directory) as a template:

```
set cisco-1232 {
    { HardwareType        ap                  }
    { Vendor              cisco               }
    { APModel             cisco-1232          }
    { APSWVersion         IOS-12.3(11)JX      }
    { WLANSwitchModel     n/a                 }
    { WLANSwitchSWVersion n/a                 }
    { ConsoleAddr         192.168.10.245      }
    { ConsolePort         2035                }
    { ApUsername          Cisco               }
    { ApPassword          Cisco               }
    { AuthUsername        ""                  }
    { AuthPassword        Cisco               }
    { Ssid                veriwave            }
    { SsidBroadcast       true                }


    { RadiusServerAddr      10.10.250.1   }
    { RadiusServerSecret    nothing       }
    { RadiusServerAuthPort  1812          }
    { RadiusServerAcctPort  1813          }

    {
        Interface     {
            { Dot11Radio0 {
                { InterfaceType 802.11bg            }
                { DHCP          Enable              }
                { Power         1                   }
                { WavetestPort  192.168.10.246:9    }
                { AntennaRx     DIVERSITY           }
                { AntennaTx     DIVERSITY           }
            }}
            { BVI1 {
     { InterfaceType 802.3              }
                { DHCP          Enable              }
                { IpAddr        10.10.250.36        }
                { IpMask        255.255.255.0       }
                { Gateway       10.10.250.1         }
                { WavetestPort  192.168.10.246:1    }
            }}
        }
    }
}
```

3. Copy that file and rename it.

```
cp cisco-1232.tcl myname-cisco-ap.tcl
```

4. Make the appropriate changes in the new file. Here's what your copy might look like:

```
set myname-cisco-ap {
    { HardwareType          ap                  }
    { Vendor                cisco               }
    { APModel               cisco-1231          }
    { APSWVersion           IOS-12.3(11)JX      }
    { WLANSwitchModel       n/a                 }
    { WLANSwitchSWVersion   n/a                 }
    { ConsoleAddr           10.10.200.10        }
    { ConsolePort           2044                }
    { ApUsername            lab                 }
    { ApPassword            lab                 }
```

```
    { AuthUsername          ""              }
    { AuthPassword          lab             }
    { Ssid                  my_test         }
    { SsidBroadcast         true            }


    { RadiusServerAddr      10.10.200.1     }
    { RadiusServerSecret    supersecret     }
    { RadiusServerAuthPort  1812            }
    { RadiusServerAcctPort  1813            }
.
.
.
```

5. After creating the DUT definition, you reference it in your configuration file. If your DUTs are managed in the dut directory, include a line like this:

   `cfg_load_module dut myname-cisco-ap`

6. Alternatively, if you do not wish to manage multiple files, you can copy and paste the DUT definition into your configuration file and modify it there.

- **APModel**, **APSwVersion** — Not directly used by WaveAutomation; only used in the PDF reports.
- **InterfaceType** — Determines which radio or ethernet interface to use in the test.
- **WavetestPort** — Defines the linkage between the interface on the access point and the port on your WaveTest chassis. As shown above, you must use a colon to separate the access point's IP address (in the example, `192.168.10.246`) or DNS name and the port number (`1`).
- For roaming or mesh tests, you will need to define DUT definitions for each access point you specified in the AuxDut variable (page 25). Since the WaveAutomation configuration file is actually a TCL script, it is easy to quickly create multiple DUT definitions.

  For example, you can make a copy of the DUT called `my-ap` and name the DUT `my-ap2`. The WavetestPort information on `my-ap2` is changed to match how it is attached to the WaveTest chassis:

```
set my-ap2 $my-ap
keylset my-ap2 Interface.802_11b.WavetestPort
  192.168.10.246:3
keylset my-ap2 Interface.802_11a.WavetestPort
  192.168.10.246:2
keylset my-ap2 Channel 9
```

The "generic" vendor type

When the DUT is already configured, or if another part of the test system will be doing the configuration, you can use the "generic" vendor type in the DUT configuration section. For example:

```
keylset ap12 Vendor                          generic
keylset ap12 APModel                         unspecified
keylset ap12 APSwVersion                     unspecified
keylset ap12 Interface.802_11b.InterfaceType 802.11bg
keylset ap12 Interface.802_11b.WavetestPort  192.168.10.249:1
keylset ap12 Interface.802_3.InterfaceType   802.3
keylset ap12 Interface.802_3.WavetestPort    192.168.10.249:7
keylset ap12 Interface.802_11b.Bssid         00:15:70:01:c0:e0
```

- **Bssid** — This example attaches a specific BSSID to the b radio. If you don't specify a valid BSSID, this defaults to 0 (scan for the BSSID on the specified channels).

Binding a DUT to a group

To bind the DUT to a specific group, include a line of the following form:

```
keylset wireless_group_a Dut myname-cisco-ap
```

Note. *The DUT name that you set in the group configuration must match the name of a DUT definition. This is how the automation software figures out which DUTs you are testing and how to reach those DUTs so they can be configured.*

Configuring DUTs in external files

The Group Configuration section (page 26) discusses the possibility of managing group definitions in a set of one or more external files. This option also exists for DUTs.

Because DUT configurations are often static after they are defined and working, you might choose to manage DUT configurations as one or more external files, which you then include into this configuration file. This approach allows you to share definitions among multiple DUTs, with the additional benefit of making the configuration file more compact and easier to maintain. The DUT configurations can be reused and shared by multiple users who may choose to reference those common/shared DUT configurations from their own separate and more frequently changed personal automation configuration files.

After you have created a DUT definition file, you can load it by including a line similar to this:

`cfg_load_module dut` *dutNameAndModel*

For example:

```
cfg_load_module dut 3Com-2750
```

The `dut` command understands the WaveAutomation directory structure, and loads the DUT definition file `3Com-2750.tcl` from the .../`automation/conf/dut` directory.

Alternatively, if you need to have a common hardware configuration across multiple installations, you can use the TCL source command to explicitly specify the path of the included file. For example:

```
source /usr/local/etc/lab_hardware.tcl
```

or

```
source ~/hardware.tcl
```

For examples of DUT configuration files, examine the existing files in the .../`automation/conf/dut` directory.

Advanced features

Command-line variable substitution

WaveAutomation allows you to pass variables on the command line. This allows you to use a single configuration file for multiple purposes. For example, if you create a configuration file containing this line:

```
keylset wireless_group Dut $my_dut
```

you can control the DUT to test from the command line:

```
$ automation/bin/vw_auto.tcl --var my_dut tims_ap
```

Under certain circumstances, you might also find it useful to define a list of channels, frame sizes, or security methods from the command line.

If you are using list (curly-brace) notation in your configuration file, TCL syntax requires that you define any variables (such as `$my_dut`) on separate keylset lines, rather than in a group definition.

Running WaveAutomation

After you have set up the configuration file, running WaveAutomation is a straightforward process. The following procedures apply to Linux and Windows, respectively.

Run the tests under Linux

1. Change to the directory where WaveAutomation is installed:

   ```
   cd $HOME/automation/automation/bin
   ```

2. Run WaveAutomation. Specify your configuration file (in this example, config-tim.tcl).

   ```
   $./vw_auto.tcl -f ../conf/config-tim.tcl
   ```

   To find out more about what is happening while the tests execute, you can use the `--debug` option:

   ```
   $./vw_auto.tcl -f ../conf/config-tim.tcl --debug 3
   ```

3. Examine the test results.

As the tests execute, they send status information to your screen. Detailed test logs and results files are written to the directory you specified in the global configuration section (by default, `…/automation/results`). That directory captures all test output in a file called `output.log`.

WaveAutomation generates the same logs that are generated by running the WaveApps tests interactively. For details about interpreting these tests, see the *WaveApps User Guide*.

Run the tests under Windows

1. Open a command window:

- Click Start>Run.
- Type `cmd.exe`, then click **OK**.

2. In the command window, change to the directory where you extracted the archive (for example, `C:\WaveAutomate-2.4.2\automation`).

3. Run WaveAutomation. Specify your configuration file (in this example, config-tim.tcl).

   ```
   tclsh automation\bin\vw_auto.tcl -f
   ../conf/config-tim.tcl
   ```

   To find out more about what is happening while the tests execute, you can use the `--debug` option:

   ```
   $./vw_auto.tcl -f ../conf/config-tim.tcl --debug 3
   ```

4. Examine the test results.

Test execution

When you execute the test automation script `vw_auto.tcl`, it executes `vwConfig.py`, a tool that generates a WML configuration file that the individual tests understand. The specific tests are then executed, using the derived WML file to produce results that you can examine.

The test results are identical to what you would get from running WaveApps interactively.

Note. *The first time you run the automation, it may take a significant amount of time (5 minutes or more) for the first test case to generate the first PDF report showing the test results. This is a one-time delay. Subsequent test cases will not experience this delay. The delay is caused because some large graphic images in the reports are generated at run time when the first report is generated. Once generated, those graphics are reused without the need to generate them again when new reports are created.*

Examine the results

For each test that you run, the output results are stored in a separate time-stamped subdirectory, created under `automation/results/`.

```
$ cd automation/results
$ ls
20070614-154936 20070507-135726 20070613-105730 20070620-090412
20070427-085138 20070507-140838 20070613-112703
$ cd 20070614-154936
$ ls -l
total 96
-rw-r--r-- 1 tim users 11159 2007-06-14 15:49 gen_code.tcl
-rw------- 1 tim users 84030 2007-06-14 15:54 output.log
drwxr-xr-x 4 tim users 112 2007-06-14 15:52 unicast_packet_loss
```

The top level of the results tree contains these files and directories:

- gen_code.tcl — Used internally by WaveAutomate.
- output.log — Contains the full test run output.
- A directory for each test that you configured under the BenchmarkList variable (here, `unicast_packet_loss`).

Each test directory can contain a large number of directories. WaveAutomation creates a subdirectory for each variable it ran a test for. For example, if your configuration file contained this line:

```
keylset wireless_group Channel { 1 6 }
```

your results directory would include a separate subdirectory for each channel:

```
$ cd unicast_packet_loss
$ ls -l
total 0
drwxr-xr-x 4 tim users 120 2007-06-14 15:50 Channel=1
drwxr-xr-x 4 tim users 120 2007-06-14 15:53 Channel=6
```

Similarly, if you directed WaveAutomation to test two different security methods:

```
keylset wireless_group Method { None WPA2-EAP-TTLS-GTC }
```

then both the Channel=1 and Channel=6 directories would contain separate subdirectories for each security method:

```
$ cd Channel=1
$ ls -l
total 0
drwxr-xr-x 2 tim users 376 2007-06-14 15:50 Method=None
drwxr-xr-x 2 tim users 376 2007-06-14 15:52
  Method=WPA2-EAP-TTLS-GTC
```

The actual test results are at the very bottom of the subdirectory tree:

```
$ cd Method=WPA2-EAP-TTLS-GTC
$ ls -l
total 264
```

```
-rw-r--r-- 1 tim users 9536 2007-06-14 15:52
  Console_unicast_packet_loss.html
-rw-r--r-- 1 tim users 787 2007-06-14 15:52
  Detailed_unicast_packet_loss.csv
-rw-r--r-- 1 tim users 214804 2007-06-14 15:52
  Report_unicast_packet_loss.pdf
-rw-r--r-- 1 tim users 514 2007-06-14 15:52
  Results_unicast_packet_loss.csv
-rw-r--r-- 1 tim users 106 2007-06-14 15:51
  RSSI_unicast_packet_loss.csv
-rw-r--r-- 1 tim users 16572 2007-06-14 15:52
  Timelog_unicast_packet_loss.txt
-rw-r--r-- 1 tim users 7600 2007-06-14 15:51
  unicast_packet_loss.wml
```

Although the actual names of the files will vary depending on the test, the naming is similar for all tests.

- Console_*testName*.html — The text output while the actual test ran.
- Detailed_*testName*.csv — Detailed test results in CSV format.
- Report_*testName*.pdf — A graphical report of the test results.
- Results_*testName*.csv — Test results summary in CSV format.
- Timelog_*testName*.txt — VCL debugging information.
- *testName*.wml — Configuration file for this particular test.
- *.cap, *.vwr — Packet capture files. Produced when a test aborts or when turned on via the command line.

Special forms of testing

Most users won't need to use the techniques specified in this section.

If you are testing new or unsupported types of devices under test (DUTs), you are responsible for properly configuring each DUT before launching the automated tests. The following command line includes options that tell WaveAutomation to skip the step of configuring the DUTs and only configure the WaveTest system prior to each test:

```
$./vw_auto -f ../conf/config-tim.tcl --debug 5 --noclean
  --noconfig --noping
```

APPENDIX A

vw_auto.tcl Command Syntax

This appendix provides command-line syntax for the WaveAutomation vw_auto.tcl testing script. Some users may find it useful to use the command line to override values in the WaveAutomation configuration file.

The vw_auto.tcl script loops through a list of DUTs (access points) as well as a list of security types and runs the VeriWave throughput test on each DUT for each security type. DUTs and security types are defined in the WaveAutomation configuration template file $VW_TEST_ROOT/conf/config.tcl.

Example

```
./vw_auto.tcl --debug 3 -f /tmp/my_config.tcl
```

Usage

```
vw_auto.tcl
```

When you execute the test automation script `vw_auto.tcl`, it executes `vwconfig.py`, a tool that generates a WML configuration file that the individual tests understand. The specific tests are then executed, using the derived WML file to produce results that you can examine.

The test results are identical to what you would get from running WaveApps interactively.

vw_auto.tcl Command-Line Arguments

`-h` or `--help`	Display a help message with full syntax.
`-c` <*ipAddr*|*host*>	Override the WaveTest chassis IP defined in the configuration file (page 19).
`-e` or `--environ`	Verify that the required TCL/Expect/Python versions are present (see page 16).
`-f` *configFile*	Use the specified configuration file instead of the default $VW_TEST_ROOT/conf/config.tcl.
`-w` *benchmark WMLfile*	Run using the specified benchmark using the specified WML file (see page 39).
`-i` *id*	Use the specified ID to tag the result output.
`-o` *outputFile*	Write a log of the test run to the specified file. The default file is in the results directory.
`--license` *key*	Use the specified license key to enable certain tests (page 21).
`--debug` *level*	Print out debugging output. Useful values are: 1 – errors only 2 – warnings and errors 3 – warnings, errors and info
`--duts` *dutList*	Override the DUT_LIST in the configuration file (page 23).
`--nodut`	Do not try to access the DUT. Equivalent to --noclean --noconfig and --noping.
`--noclean`	Do not attempt to erase configuration on the access point before and between test runs.
`--noconfig`	Do not configure the access point for the current security method before running the test.
`--noping`	Do not check network connectivity between the access point and WaveTest chassis.
`--nopause`	Do not pause between configuring the access point and running the benchmark.
`--pause` *seconds*	After changing the configuration on the DUT, wait the specified number of seconds before scanning for BSSIDs. This is to allow time for the DUT to reconfigure and re-initialize its radio interfaces and begin sending beacons before scanning for BSSID's. The default is 15 seconds.
`--notest`	Do not run the benchmark.
`--nowml`	Do not generate a WML file.

`--srcgroups` *groupList*	Override the list of destination source groups in the configuration file (page 21).
`--destgroups` *groupList*	Override the list of destination client groups in the configuration file (page 21).
--tests *testList*	Override the BenchmarkList in the configuration file (page 20).
`--savepcaps`	Save PCAP files from test runs (regardless whether tests PASS or FAIL). By default, PCAP files are only saved when tests FAIL or ABORT
`--var` *name value*	Set the variable *name* to the specified *value* before sourcing the configuration file. This allows you to use the variable `$name` in the configuration file (page 33). Values may be singular values or quoted lists. For more details, see `.../automation/doc/advanced.txt`.