Software User Manual

NetApp

Network Application (NetApp) User Guide

NetApp-SWUM101-R

5300 California Avenue • Irvine, CA 92617 • Phone: 949-926-5000 • Fax: 949-926-5203

July 27, 2012

Revision History

Revision	*Date*	*Change Description*
NetApp-SWUM101-R	07/27/12	**Updated:** • "Supported Features" on page 10 • Build variables added to "Build from an AppLibs Release" on page 25 • "Set the Time and Date" on page 24 **Removed:** • "CallBacks" section (refer to the NetApp programmer's guide for this information)
NetApp-SWUM100-R	04/27/12	Initial release

Broadcom Corporation
5300 California Avenue
Irvine, CA 92617

Table of Contents

About This Document 7

Purpose and Audience 7

Acronyms and Abbreviations 7

Document Conventions 7

Reference Documents 8

Technical Support 8

Section 1: Overview 9

High-Level Requirements 9

Supported Features 10

Architecture 11

Directory Structure 11

Section 2: NetApp API Design 13

Overview 13

CallBacks 13

IP Settings 16

Link Monitoring 17

Wi-Fi 18

Wi-Fi Protected Setup 18

Wi-Fi Scan 20

Wake-On-Wireless LAN 21

Additional Wi-Fi Information 21

Zero Configuration Networking 22

Bluetooth 22

Location Aware Service 22

Additional Networking Features 23

Section 3: Using NetApp 24

Makefile 24

Using netapp.h 24

Set the Time and Date 24

Build NetApp 25

Build from an AppLibs Release 25

NetApp-Only Tarball 26

Testing 27

Run NetAppTester 27

Test Example...27
Appendix A: Readme File 29

List of Figures

Figure 1: NetApp Architecture 11
Figure 2: Directory Structure 12
Figure 3: Handshaking Between Application and Kernel 16
Figure 4: Handshaking to Determine Link State 17
Figure 5: Handshake to Acquire Wi-Fi Connectivity 19
Figure 6: Obtaining Wi-Fi Scan Results 20
Figure 7: Location Aware Service 23

List of Tables

Table 1: General Callbacks 13
Table 2: Wi-Fi Callbacks 14
Table 3: Bluetooth Callbacks 14
Table 4: Link State Descriptions 17
Table 5: Wake-On-Wireless-LAN Events..21
Table 6: Make Build Rules 26
Table 7: Variables 26

About This Document

Purpose and Audience

This document discusses the design goals and principles used in the development of the Network Application (NetApp) API. The usability and testing of the NetApp API are also included in this document.

The intended audience includes:

- NetApp API users who want to know how it works.
- NetApp API users who want to make customizations.
- NetApp API developers who need to know how to extend it.

This does not document internal APIs or internal implementation, unless it directly affects the public API.

Where possible, actual NetApp API code is used to illustrate the principles. Be aware that the NetApp API may change slightly over time; all changes are reported with every release (refer to ChangeLog.html).

Acronyms and Abbreviations

In most cases, acronyms and abbreviations are defined on first use. For a comprehensive list of acronyms and other terms used in Broadcom documents, go to http://www.broadcom.com/press/glossary.php.

Document Conventions

The following conventions may be used in this document:

Convention	***Description***
Bold	User input and actions: for example, type **exit**, click OK, press Alt+C
`Monospace`	Code: `#include <iostream>` HTML: `<td rowspan = 3>` Command line commands and parameters: `wl [-l] <command>`
< >	Placeholders for *required* elements: enter your <username> or `wl <command>`
[ ]	Indicates *optional* command-line parameters: `wl [-l]` Indicates bit and byte ranges (inclusive): [0:3] or [7:0]

Reference Documents

The references in this section may be used in conjunction with this document.

Broadcom provides customer access to technical documentation and software through its Customer Support Portal (CSP) and Downloads and Support site (see Technical Support). For Broadcom documents, replace the “xx” in the document number with the largest number available in the repository to ensure that you have the most current version of the document.

Document Title	*Reference*	*Source*
Broadcom Documents		
[1] Wake-on-Wireless-LAN	TBD	TBD

Technical Support

Broadcom provides customer access to a wide range of information, including technical documentation, schematic diagrams, product bill of materials, PCB layout information, and software updates through its customer support portal (https://support.broadcom.com). For a CSP account, contact your Sales or Engineering support representative.

In addition, Broadcom provides other product support through its Downloads and Support site (http://www.broadcom.com/support/).

Section 1: Overview

High-Level Requirements

NetApp API is a complete API to configure and control wired and wireless interfaces and Bluetooth devices. It is a simple and lightweight API, with a single header file (netapp.h) and a collection of shared libraries (libnetall.so + other dependences).

NetApp autogenerates netapp.inc, which can be included in your application for obtaining NETAPP_CFLAGS and NETAPPLDFLAGS.

NetApp supports most Broadcom STB chipsets, including the BCM7425, BCM7346, BCM724X, BCM723X, and so on.

NetApp runs on many different Kernel versions, such as 2.6.37, 3.3, and so on.

Contact Broadcom with queries about support for a specific STB device or Kernel version.

NetApp supports the following hardware:

- Bluetooth: BCM20702
- Wi-Fi

NetApp supports the following chips and drivers:

- USB: BCM43236B (BCM43234, BCM43235, BCM43237, and BCM43238)
- Drivers:
  - Kirin: 5.102.98.42
  - Falcon: 5.90.188.51
  - Aardvark: 6.30.75

Features of NetApp are:

- Decoupling of the Wi-Fi networking and Bluetooth code from the main application.
- Ability to control just Wi-Fi or Wi-Fi along with other interfaces.
- Simplified configuration and control of the Network Interface Devices. NetApp does not manipulate the data path. It was originally designed for an STA. Simplified AP support is planned for 2012.

Supported Features

Features supported in release version 7.0.1 are:

- Networking (TCP/IP):
  - Get/Set IP settings
  - DHCP Client Daemons
  - DHCP Server (for Wi-Fi Direct™)
  - Bonjour®/Zero Configurations (Service Discovery)
  - IPv4LL, Multicast-DNS, DNS-Service Discovery
  - Multi-Threaded Single Process Support
  - Multiple clients connect and interact with NetApp (VUDU®, DLNA®, GUI/System Settings)
- Wi-Fi:
  - Connection Manager
  - Wi-Fi Protected Setup (version 1.0 and 2.0)
  - Wi-Fi Invite
  - Wake-on-Wireless-LAN (WoWL)
  - Wi-Fi Direct
  - MiraCast™ (Wi-Fi Display)
- Bluetooth:
  - Pairing/Bonding of HID remote control devices
  - Device and service discovery
  - HID Voice: Using FLAC and Google® Voice recognition search
  - Voice Recognition (Audio HID)
  - A2DP (AV Sink)
  - AVRCP
- USB Hotplug
- Database Back End:
  - SQLite
  - Automatically get/set settings to reconnect to previous access points, Bluetooth devices, etc.
- iperf
- flac (used for PCM->flac conversion for voice recognition)

Architecture

Figure 1 on page 11 is a representation of the NetApp architecture in terms of kernel and user space.

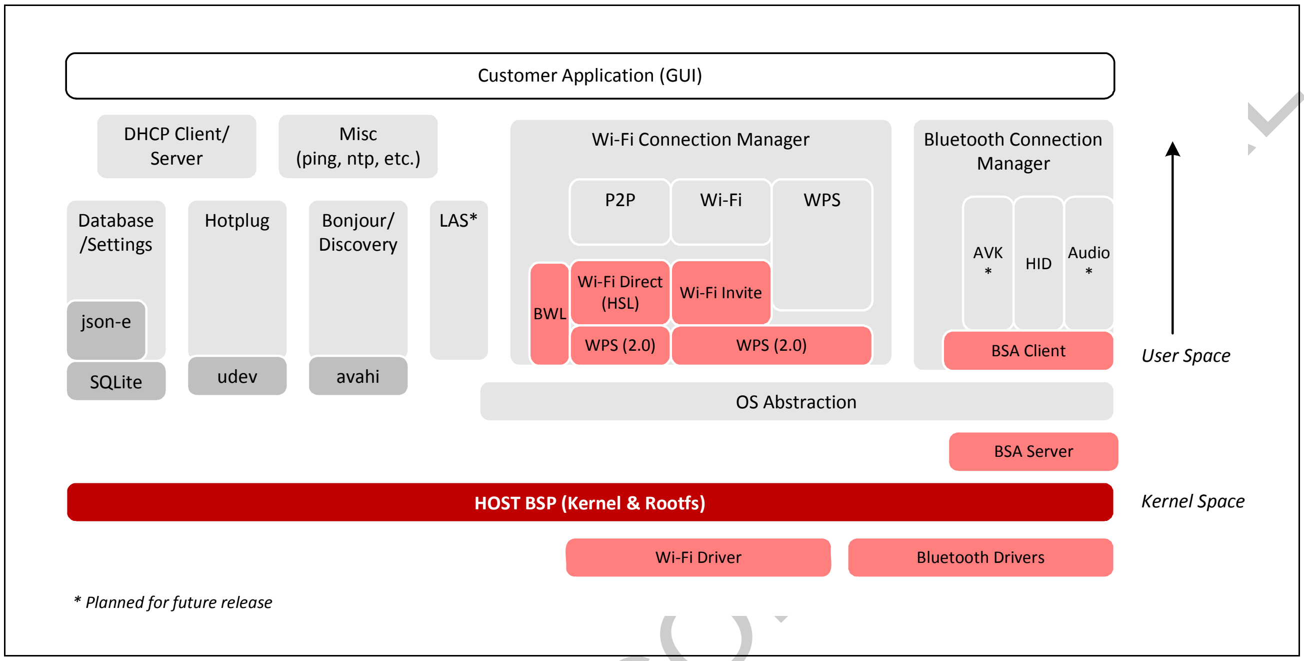
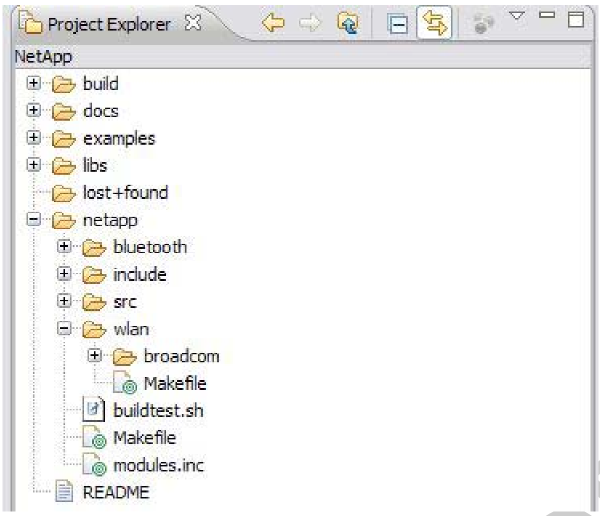


Figure 1: NetApp Architecture

Directory Structure

A screen shot of the directory structure is shown in Figure 2 on page 12. The top level directories are:

- build, which includes the main makefile to build NetApp and all the dependant library files. File common.inc is a head file used internally by NetApp modules.
- examples, which includes simple API examples on how to build, run, and use NetApp.
- docs, which contains the API documentation (auto-built).
- libs, which contains the dependant libraries used by NetApp. If any of the libraries are not built or included then NetApp will disable the library support.
- netapp, with following subdirectories:
  - Bluetooth, containing the current integrated BSA software release.
  - Include, containing the NetApp header file.
  - src, containing the NetApp source code.
  - Wlan, containing the WLAN code.
- README, which provides detailed instructions on how to build and link with NetApp.



```
NetApp
  build
  docs
  examples
  libs
  lost+found
  netapp
    bluetooth
    include
    src
    wlan
      broadcom
      Makefile
    buildtest.sh
    Makefile
    modules.inc
  README
```

Figure 2: Directory Structure

Section 2: NetApp API Design

Overview

All synchronous APIs return immediately. Any API that takes longer than 50 ms is asynchronous and returns the status in a callback defined in NETAPP_CB_TYPE. An API can be opened multiple times within a single process. Each open instance can register its own callback instance. The back end framework (threads, timers, semaphores, etc.) are instantiated the first time NetAppOpen() is called and torn down during the last NetAppCLose() call.

For efficient power management, each NetApp API should be torn down when entering standby and then re-opened after leaving standby.

The API is protected with a mutex (binary semaphore). This is to ensure that only one context is controlling vital components of the API at a time.

Debug output is available with you run `Export NETAPP_DEBUG = y`.

CallBacks

Table 1 lists all the general callbacks available in the API.

Table 1: General Callbacks

NETAPP_CB Name	*Description*	*pvBuffer*	*ulData0*
LINK	Change in the link status, link states are: • DOWN: The Ethernet link is down (disconnected). • ACQUIRING: The Link is up but NetApp is in the process of acquiring or configuring the IP address. • UP: The Ethernet link is up and the interface has an IP address.	Not Used	NETAPP_LINK_ STATE
PING	Results from a Ping request.	Ping request addres/name (char*)	Not used
DNSLOOKUP	Results from a DNS lookup.	DNS lookup request name (char*)	Not used
NTPDATE	The call to NetAppNtpSetDate() has completed in the background and the callback will simply send the results.	Not used	Not used

Table 1: General Callbacks (Cont.)

NETAPP_CB Name	Description	pvBuffer	ulData0
SETSETTINGS	This callback is a result of a user calling NetAppSetNetworkSettings() and we have the results from trying to apply the settings.	Not used	Not used

Table 2 lists all of the Wi-Fi callbacks in the NetApp API.

Table 2: Wi-Fi Callbacks

NETAPP_CB Name	Description	pvBuffer	ulData0
SCAN_RESULTS	Wi-Fi Scan has been completed and the results are available to be fetched by calling NetAppWiFiGetScanResults(). The scan interval is the only information passed to this callback.	Not used	Scan count (incremented after each successful scan)
CONNECT	Bluetooth or Wi-Fi connection attempt results	NETAPP_WIFI_ AP_INFO structure or NETAPP_BT_DEV_INFO	Not used
INVITE	The client device has received an invite request.	AP's SSID	Not used
FETCH_APINFO	We have the results from the NetAppWiFiGetApInfo() call and the access point information is passed as port of the callback.	NETAPP_WIFI_ AP_INFO structure	Not used
HOTPLUG	The Wi-Fi interface was inserted or removed.	Interface Name (char*)	NETAPP_LINK_ STATE
RSSI_EVENT	There was a change in the signal strength from one discreet value to the next.	Not used	NETAPP_WIFI_ RSSI
ZEROCONF	NetApp has found a browse/ discovery request service or the service is removed.	Service Name (char*)	NETAPP_ZEROCONF_ SERVICE_STATE
P2P_PEER	Wi-Fi Direct Peer information	NETAPP_P2P_PEER_INFO	Number of discovered peers
P2P_CONNECT	Wi-Fi Direct Connection is established	NETAPP_WIFI_AP_INFO	Not used

Table 3 lists all of the Bluetooth-enabled callbacks in the API.

Table 3: Bluetooth Callbacks

NETAPP_CB Name	Description	pvBuffer	ulData0
CONNECT	Bluetooth or Wi-Fi connection attempt results	NETAPP_WIFI_ AP_INFO structure or NETAPP_BT_DEV_INFO	Not used

Table 3: Bluetooth Callbacks (Cont.)

NETAPP_CB Name	*Description*	*pvBuffer*	*ulData0*
BT_DISCOVERY_ RESULTS	Bluetooth discovery is complete and results are available.	Not used	Number of discovered devices
BT_SP_CONFIRM	Simple pairing confirm request	NETAPP_BT_DEV_INFO	Not used
BT_SP_NOTIFY	Notify the application that the value passed in ulData0 of the callback must be entered in the Bluetooth device we are trying to connect too.	NETAPP_BT_DEV_INFO	Value to enter

IP Settings

The API is valid for any network interface (wired, wireless, etc.). To fetch IP settings, use the callback NetAppGetNetworkSettings(). This is a list of the settings provided:

- IP mode (static, dynamic, off)
- MAC address
- IP address
- Subnet mask
- Gateway
- Primary DNS
- Secondary DNS

Figure 3 shows the behavior of the handshake between the application and kernel.

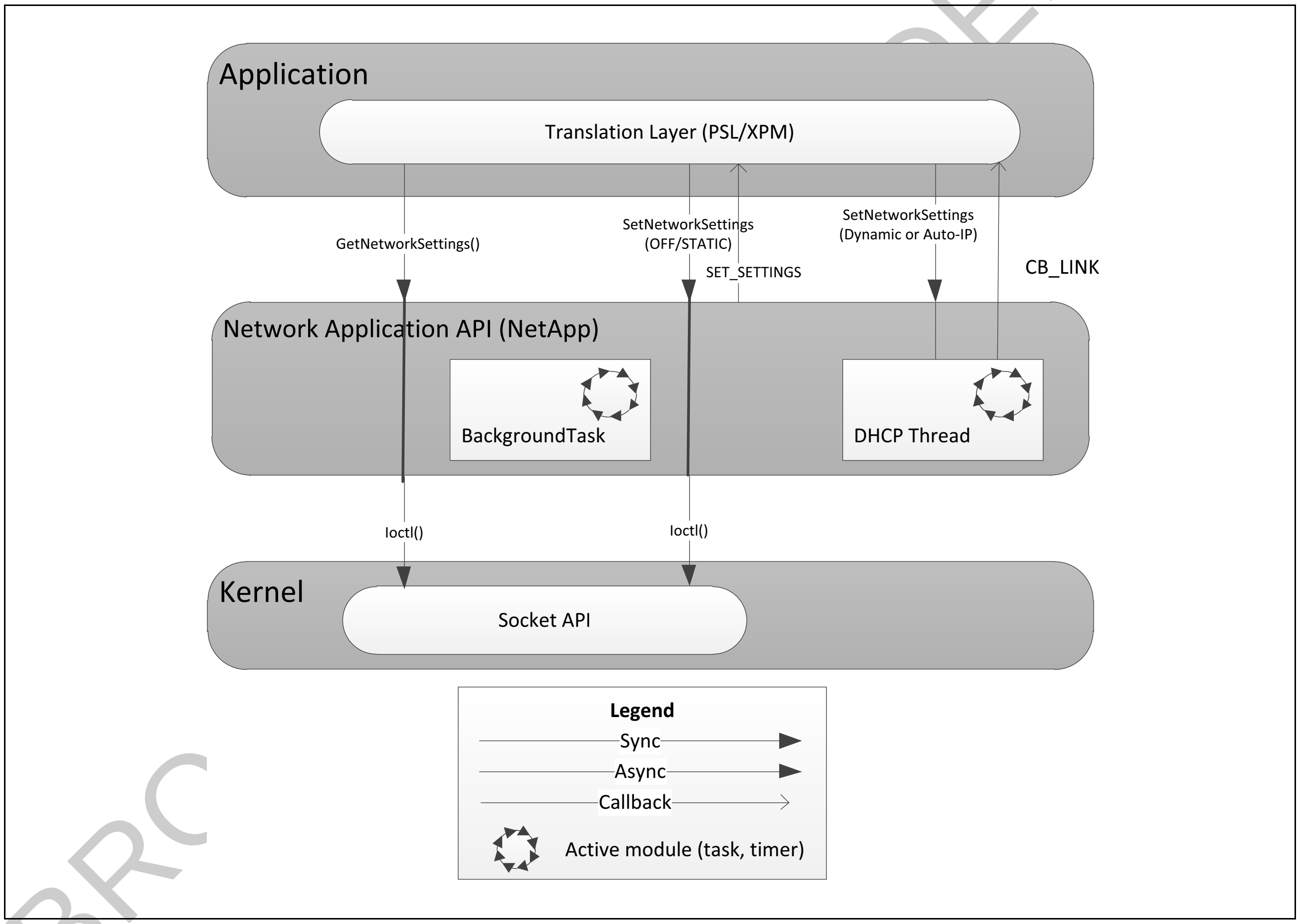

Application
Translation Layer (PSL/XPM)
GetNetworkSettings()
SetNetworkSettings (OFF/STATIC)
SET_SETTINGS
SetNetworkSettings (Dynamic or Auto-IP)
CB_LINK
Network Application API (NetApp)
BackgroundTask
DHCP Thread
Ioctl()
Ioctl()
Kernel
Socket API
Legend
Sync
Async
Callback
Active module (task, timer)

Figure 3: Handshaking Between Application and Kernel

Link Monitoring

Table 4 describes the link states. Figure 4 shows the behavior of the handshake in determining the link state.

Table 4: Link State Descriptions

Link State	*Description*
DOWN	The data link is down. This means that the wired cable is unplugged or the wireless interface is not connected to an Access Point (or the wireless interface is not plugged in).
ACQUIRING	The Data Link is up (the cable is plugged in or connected to a Wireless Access Point) and we do not have an IP address.
UP	The data link is up and we have an IP address.

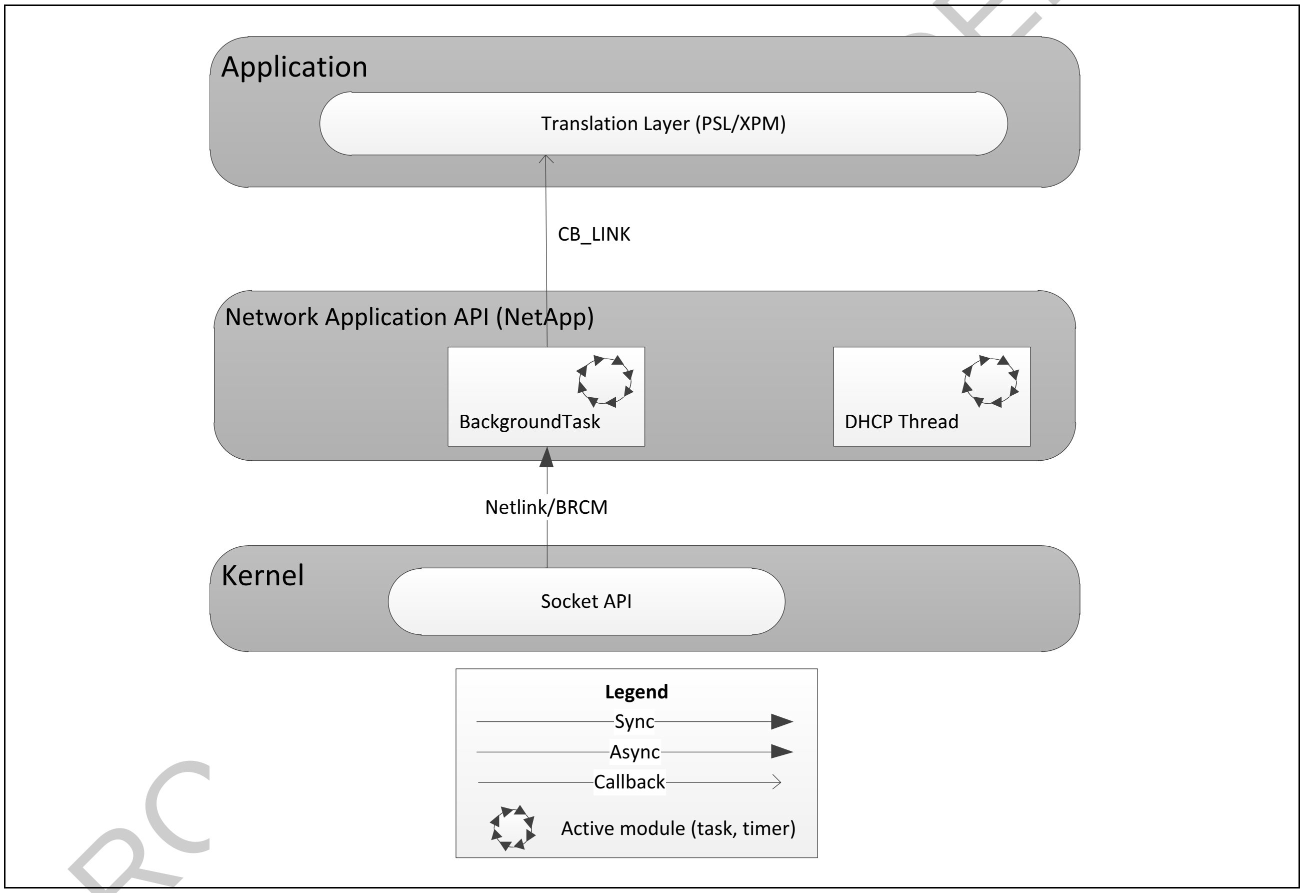


Figure 4: Handshaking to Determine Link State

Wi-Fi

The API uses Wi-Fi Protected Setup™ (WPS) version 1.0 and 2.0. This supports Open, WEP, WPA™ PSK, and WPA2™ PSK (AES and TKIP). It contains a complete built-in connection manager. Wi-Fi will connect with as little as an SSID and a password. NetApp will automatically determine the security settings of the AP, even for hidden access points. Concurrent request issues, such as connection and scanning, are triaged.

Wi-Fi Protected Setup

Wi-Fi Protected Setup is built in the registration loop. It contains fully-configurable device identifiers such as manufacturer, model number, and product name. There are two simple APIs:

- The push button API, NetAppWiFiConnectByPb(). See WPS Test Plan sections §5.1.2, §5.1.6, §5.1.7, and §5.1.8.
- The pin API, NetAppWiFiConnectByPin(). See WPS Test Plan §5.1.1, §5.1.3, §5.1.4, §5.1.5, and §5.2.x.

Wi-Fi Protected Setup definitions are as follows:

- Registrar, which controls the network. The wireless device must make itself known to the registrar in order to get the necessary credentials.
- Enrollee, which is part of the wireless devices performing the registration protocol with the registrar. At the end of the process, the enrollee will get the SSID and the credentials of the network.
- AP, which gets its authentication and encryption settings from the registrar. The registrar can be embedded in the AP.

The Wi-Fi Alliance® *WPS Test Plan 1.10* defines the required and optional tests.

- §5.1: Add to AP as an Enrollee.
  - Mandatory tests.
  - WPS Pin (using the BD/DTV generated WPS pin from NetAppWiFiGenerateWPSPin()).
- §5.2: Act as Registrar and Configure AP.
  - Tests are not mandatory.
  - NetApp support for these tests using routers pin (registrar pin) by setting bEnrollee== FALSE when calling NetAppWiFiConnectByPin().
- §5.3 Act as Registrar and add devices.
  - Test are not mandatory.
  - No support in NetApp.

For a normal connection, use API NetAppWiFiConnect(). This requires a SSID and, if the AP has a password, a password. The rest of the NETAPP_WIFI_AP_INFO parameters are optional.

Figure 5 shows the handshake process to acquire Wi-Fi connectivity.

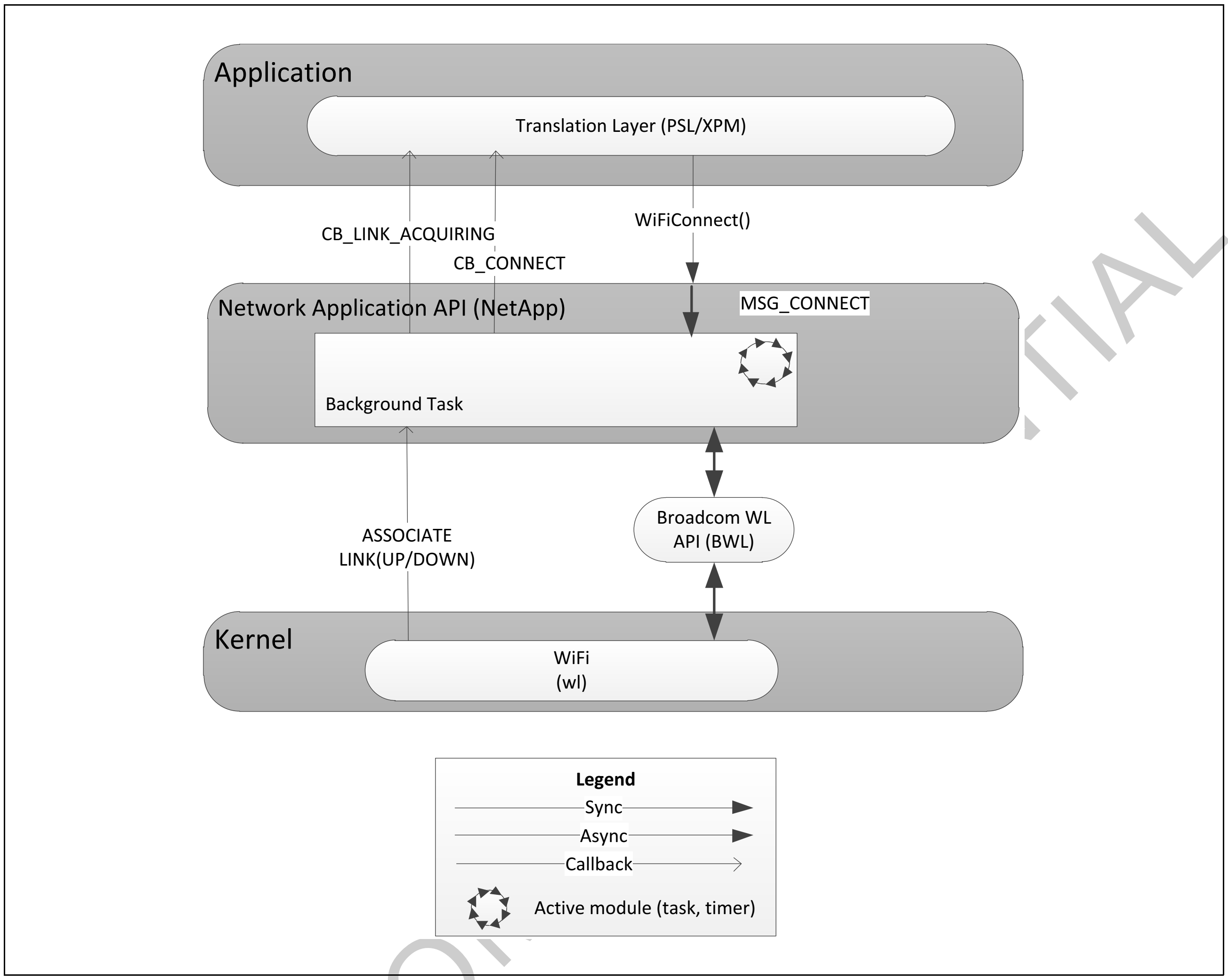


Figure 5: Handshake to Acquire Wi-Fi Connectivity

Wi-Fi Scan

The scan manages concurrent scan requests such as Wi-Fi Invite, P2P. The scan results are cached inside of NetApp and can be fetched inside any context. Results are fetched from NetAppWiFiGetScanResults() and the returned list must be freed (returned a copy). The scan parameters can be tweaked to return results faster with less accuracy or longer with more accuracy.

Figure 5 shows the process to obtain Wi-Fi scan results.

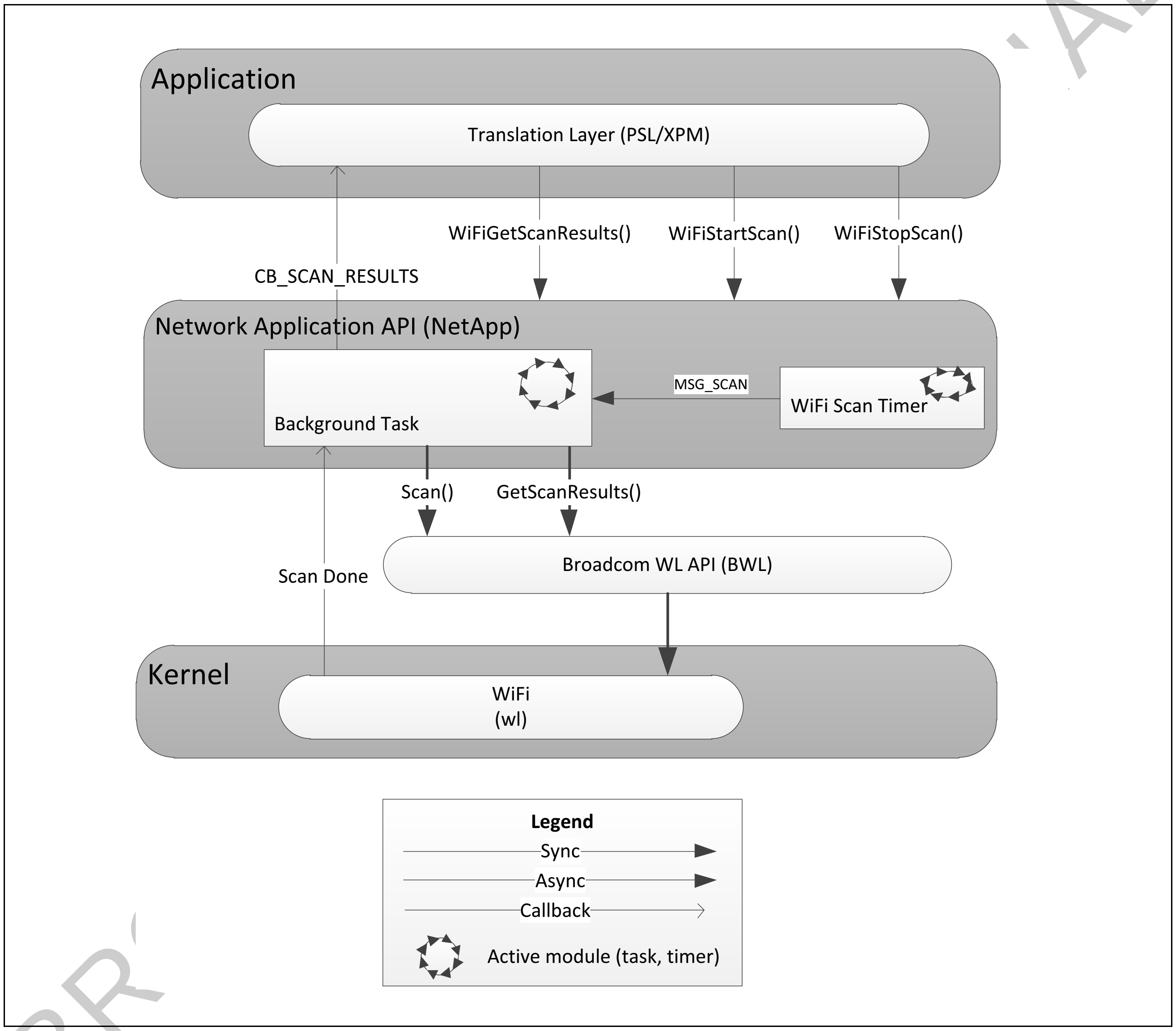


Figure 6: Obtaining Wi-Fi Scan Results

Wake-On-Wireless LAN

Wake-On-Wireless LAN (WoWL) gives the ability for the wireless interface to wake up from a magic or other type of packet. NetApp provides a simple and single API to be able to completely configure and control S0WL through the NETAPP_WOWL_SETTINGS of NETAPP_SETTINGS. WoWL requires a wireless adapter to wake up a system in Standby, Hibernate, Sleep (S3/S5 states) when one or more configured events occur. The W0WL settings need to be applied before the system enters standby; otherwise the interface will not know when and how it should wakeup.

More details regarding WoWL are defined in the *Wake-on-Wireless-LAN* application note.

Table 5 lists the WoWL events and descriptions.

Table 5: Wake-On-Wireless-LAN Events

Event	***Description***
None	Disable the WoWL feature because no wake up events are specified.
Magic Pattern	Magic packet is a unicast or broadcast frame sent to the sleeping station with a payload containing 6 repetitions of 0xFF followed by 16 repetitions of the station's MAC address. This is also sometimes referred to as "waking" by the management station. This packet can be sent by any machine on the network that can reach the dozing STA. A mini HOWTO guide can be found at on the Internet at http://gsd.di.uminho.pt/jpo/software/wakeonlan/mini-howto/wol-mini-howto-2.html.
Diss-association	Wake up when the NIC receives a deauthenticate or disassociate frame.
Loss of Beacon	Wake up after NetApp has not received a beacon for more than a predetermined number of seconds.
Net Pattern	Driver can also configure any arbitrary pattern (subjected to limitations) that microcode needs to match the payload against. This pattern is described by the following elements: • offset: The offset within the payload to start looking for the pattern. • mask: Each bit of the mask corresponds to a byte of date in the value of the pattern; so bit *i* of the mask = 1 => matchs byte *i* of the pattern with payload. • value: Pattern data, in bytes, of payload to match against. • pattern size: Size of the pattern. The mask size has to correspond to pattern size.

Additional Wi-Fi Information

- Country code/regulatory revision
- Wireless Ethernet Transceiver (L2 NAT)
- Current connection and enabled (Wi-Fi present or not) status
- Disconnection

Zero Configuration Networking

Zero configuration networking implements the automatic resolution distribution of computer hostnames (multicast DNS) and automatic location of network services such as printing devices through DNS service discovery. It utilizes Avahi (LGPL) as the backend M-DNS and DNS-SD engine. The service routine is embedded inside of NetApp.

NetApp maintains a linked list of discovered and published services that can be fetched from the application. The following APIs are available:

- NetAppZeroConfPublish(), to publish a service.
- NetAppZeroConfBrowse(), for discovery services.
- NetAppZeroConfGetBrowseResults(), to fetch the discovered results.

Bluetooth

NetApp supports Broadcom's BCM20702 Bluetooth devices connected by UART or USB. The currently supported profiles are HID keyboard, mouse and remote. This simple, thin API has the following functions:

- NetAppBluetoothDiscovery(), to discover Bluetooth devices.
- NetAppBluetoothGetDiscoveryResults(), to get the discovered results.
- NetAppBluetoothConnect(), to connect/pair Bluetooth devices.
- NetAppBluetoothDisconnect(), to disconnect connected/paired devices.

Location Aware Service

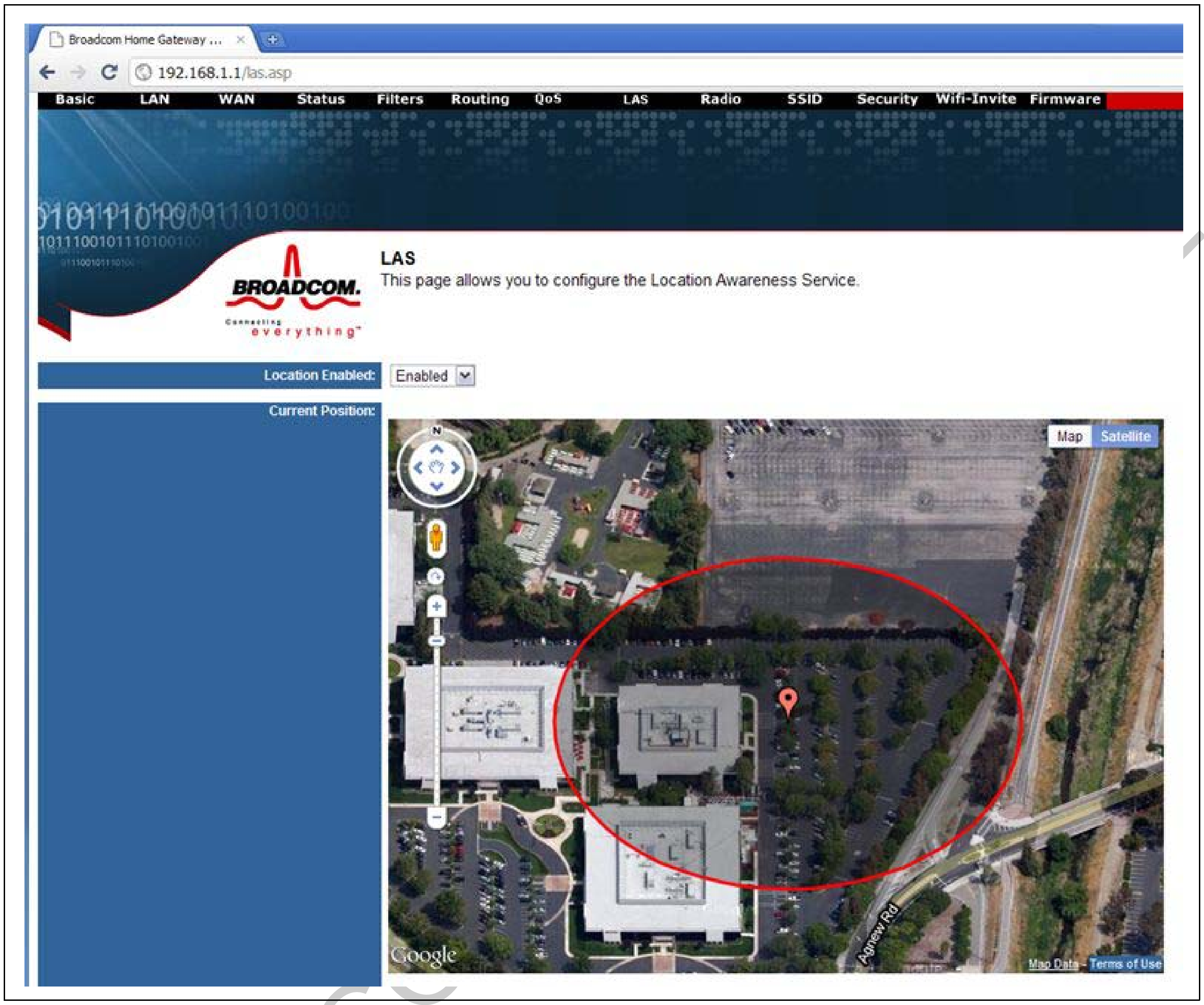
The Location Aware Service (LAS) is an enhanced application experience that provides the location of the device with the following parameters:

- Longitude
- Latitude
- Uncertainty

This feature is useful for many connected applications and location aware search. Millions of device locations have been collected using Broadcom data centers since 2009.

The simple option in NETAPP_SETTING will be used to enable and disable this feature and will be available in a future release.

Figure 7 shows the NetApp LAS configuration screen.



LAS

This page allows you to configure the Location Awareness Service.

Location Enabled: Enabled

Current Position:

Figure 7: Location Aware Service

Additional Networking Features

The following are a list of miscellaneous network features that are available though the NetApp API:

- Ping
- DNSLookup
- NTPSetDate (apply the system date and time using NTP)
- IPv4 address manipulation
- Fetch the interface name
- Set interface up/down
- Fetch the link state (up, down, acquiring IP address)

Section 3: Using NetApp

Makefile

The file netapp.inc is automatically generated when NetApp is built. It also auto generates CC, CCX, LD, STRIP, SETAPP_LDFLAGS, and NETAPP_CFLAGS that are required by an application to link and run with NetApp. The files is located in the ${NETAPP_OUTPUT_DIR} / include folder.

The following code is an example of the makefile:

```
include ${NETAPP_OUTPUT_DIR}/include/netapp.inc
%.o : %.c
 @printf "[${CC}] <------------- %s \n" $<
 ${CC} ${NETAPP_CFLAGS} -c $< -o $@
test_app: test_app.c
 ${CC} -o $@ -I. ${@}.o ${NETAPP_LDFLAGS}
```

Using netapp.h

Examples on how to use the NetApp API are is defined in the single header file netapp.h. Some code examples are provided below.

Set the Time and Date

```
/* NetApp Test Application */
#include <stdio.h>
#include <sys/time.h>
#include "netapp.h"
#include <unistd.h>

#define CHECK(api)                          \
{                                           \
    if (api != NETAPP_SUCCESS)              \
    {                                       \
        printf("%s() line %d failed\n", __FUNCTION__, __LINE__);       \
        goto err_out;                       \
    }                                       \
}

/* Main Routine */
int main(int argc, char* argv[])
{
    NETAPP_HANDLE   hNetApp = NULL;
    struct timeval  tv;
    struct timezone tz;

    printf("######################################################\n");
    printf("           NetApp  %s Test\n", argv[0]);
    printf("#####################################################\n");


    CHECK(NetAppOpen(&hNetApp, NULL, NULL, NULL));

    gettimeofday(&tv, &tz);
    printf("Time=%d\n", (int)tv.tv_sec);

    CHECK(NetAppNtpSetDate(hNetApp, 2000));

    while (1)
    {
        gettimeofday(&tv, &tz);
        printf("Time=%d\n", (int)tv.tv_sec);
        sleep(1);
    }


err_out:
    printf("Exiting now!\n");
    if (hNetApp != NULL)
    {
        NetAppClose(hNetApp);
    }
    return 0;
}
```

Build NetApp

This section describes how to build NetApp either from an AppLibs release or from a standalone NetApp tarball.

Build from an AppLibs Release

To build NetApp from an AppLibs release:

1. Setup your build environment according to the instructions in the AppLibs documentation (see "Reference Documents" on page 8).
2. The target to build NetApp is:

```
cd AppLibs/common
make netapp <build options>
```

You can replace the <*build options*> parameter with any combination of the build options that are defined in Appendix A: "Readme File," on page 29.

NetApp-Only Tarball

To build NetApp, run the **Make –C build install** command:

```
Make –C build install
```

Table 6 lists the available build rules.

Table 6: Make Build Rules

Rule	***Description***
install	Builds NetApp and all sub netapp components and install the build libraries, header files, etc., into NETAPP_TARGET_DIR (default is NetApp/stage).
examples	Build the example code in example folder.
clean	Clean and remove build objects.
distclean	Return to the default state and scrub all installed components.
docs	Build the API documentation.

Table 7 contains a list of all of the build variables available for NetApp.

Table 7: Variables

Variable	***Value***	***Description***
LINUX	[kernel path]	REQUIRED to build Kernel drivers and must point to the path of a pre-built Kernel and the same Kernel version that you will be running NetApp on.
TARGET_DIR	[path/not set]	The directory where build binaries will reside (i.e., /usr/local/bin). NetApp will dump the build kernel drivers and example code to this directory. Default = /usr/local
VERBOSE	[y/not set]	Enable/disable verbose build. Default = nonverbose.
NETAPP_OUTPUT_DIR	[path to output]	Where NetApp build system should output the built headers, libraries, modules, and binaries. Default = netapp/stage/usr/local

Testing

The wireless app includes following console app in /usr/local/bin directory.

- wlmips (wl command)
- iperf
- NetAppTester

The following is an example of an iperf test on a lab bench test setup:

```
BCM97425C ← wireless → Linksys E3000 ← Gigabit Ethernet → BCM94528_SATIPSW (7425B1+4528)

Client: BCM97425C
# wlmips rssi
-34
# wlmips rate
130 Mbps
# wlmips noise
-92

iPerf show around 50 Mbps throughput and start has packet drop above 50 Mbps.
#iperf -c192.168.0.137 -t30 -u -b50.0M -w4M -B192.168.0.136 -p5001 -i1
[  3] Server Report:
[  3]  0.0-30.0 sec   179 MBytes  50.0 Mbits/sec   0.733 ms    1/127659 (0.00078%)
[  3]  0.0-30.0 sec  1 datagrams received out-of-order

Server: BCM94528_SATIPSW
iperf -s -w4M -fm -u -p5001 -i1
```

Run NetAppTester

```
# source netapp_target_prep.sh  /* setup LD_LIBRARY_PATH */
# NetAppTester
######################################################
           NetApp Test
#####################################################
NetAppTester: [-i | (options below)]
Examples:
Interactive:      NetAppTester -i
Wi-Fi Connect:     NetAppTester -c <ssid> <password optional>
```

Test Example

NetAppTester –i is interactive CLI to support scan, connect, disconnect, show Wi-Fi status, IP address, etc.

```
# NetAppTester –i
Main Command Menu:
        0: Help
        1: Bluetooth menu
        Wi-Fi Command Options:
        2: Connect
        3: Disconnect
        4: Scan (Start/Stop)
        5: Wi-Fi Direct Menu
        6: Print IP Settings
        7: Print Wi-Fi Connected Info
        99: Exit
Select action =>

// the last connection info is save in /tmp/NetApp.db and it will reconnect once NetAppTester is
running.
**************************
SSID:     jefchiao-2GHz
BSSID:    C0:C1:C0:44:5C:A4
Password:
ulChan:   11
bAdHoc:   NO
bWPS:     YES
tRSSI:    Excellent (4 bars)
tSecurity:No Security
802.11:   b g n
Rate:     144 Mbps
Noise:    -57 dBm
**************************

NetAppTester –c <ssid> connect to AP and eth2 got IP address for iperf test.

Software:
Linux: 2.6.37-2.6
NetApp: V7.0.1
Wireless driver: falcon_rel_5_90_188_43

Hardware:
BCM97425C board (7425B1 chip)
BCM943236USB p532 with two external antennas
```

Appendix A: Readme File

```
/***************************************************************************
 *     Copyright (c) 2005-2011, Broadcom Corporation
 *     All Rights Reserved
 *     Confidential Property of Broadcom Corporation
 *
 *  THIS SOFTWARE MAY ONLY BE USED SUBJECT TO AN EXECUTED SOFTWARE LICENSE
 *  AGREEMENT  BETWEEN THE USER AND BROADCOM.  YOU HAVE NO RIGHT TO USE OR
 *  EXPLOIT THIS MATERIAL EXCEPT SUBJECT TO THE TERMS OF SUCH AN AGREEMENT.
 *
 ***************************************************************************/
Network Application API (NetApp)

Author:     Steven Hartley (steven@broadcom.com)

-------------------------------------------------------------------------------
1) BUILDING NETAPP:
--------------------------------------------------------------------------------
Building NetApp is very simple and can be done in the one step below:
    make -C build

The above command will build all of NetApp including the NetAppTester console test
application that can be used to test the API.

All environment variables will be discussed in more details later on in this document
however the only mandatory variable is LINUX (and your toolchain needs to be in the path).


-------------------
1.1) TARGETS:
-------------------
OPTION              DESCRIPTION
------              -----------
examples            Build the example code in example folder
                    (This is the default recommended target option and is the default option)

help                Printout all the configuration options and build variables for NetApp

tarball             Builds a tarball containing full source and prebuilt WLAN and BT code. NOTE:
                    WLAN and BT sourceMUST be stripped out before releasing to a customer.

install             Builds NetApp and all sub netapp components and dependent libraries

clean               clean all of the code (remove build objects)

distclean           Same as clean and in addition delete installed components
                    and remove untared directories

---------------------
1.2) VARIABLES:
---------------------

1.2.1) MODULES
```

```
By default all features are enabled in NetApp, to disable them you set the
variable below to n when compiling NetApp

VARIABLE            VALUE        DESCRIPTION
--------            -----        -----------
VOICE_RECOGNISION    n              Disable FLAC and google voice recognision search

HOTPLUG              n              Disable udev and hotplug support

ZEROCONF             n              Disable avahi and zeroconfiguration (bonjour)

DATABASE             n              Disable json-c, sqlite, and the database backend

WIFI                 n              Disable WiFi

WIFI_INVITE          n              Disable WiFi Invite

WIFI_P2P             n              Disable WiFi Direct

WIFI_DRIVER  [kirin|falcon|aardvark]Set the driver to kirin or falcon (default kirin)
                                    This option is not configurable for releases with
                                    prebuilt binaries.

WIFI_CHIP    [43236b|43238b|4360a0] Which chip to use 43236 family (236, 234, 235),
                                    43238, or 4360/43526 (default 43236b).
                                    This option is not configurable for releases with
                                    prebuilt binaries.
DISABLE_WIRED_SUPPORT [y]           By default not set. The wired interface can still be
                                    controlled through NetApp API's but this build option
                                    (when set) will tell NetApp to not automatically bring
                                    up the interface when the API is opened.


1.2.3) BUILD ENVIRONMENT:
VARIABLE            VALUE           DESCRIPTION
--------            -----           -----------
LINUX               [kernel path]   REQUIRED to build Kernel drivers and must point
                                    to the path of a pre-built Kernel and the same Kernel
                                    version that you will be running NetApp on.

TARGET_DIR         [path/not set]   The directory where build binaries will reside
                                   (i.e. /usr/local/bin). NetApp will dump the build kernel drivers
                                    and example code to this directory
                                    DEFAULT=/usr/local

VERBOSE             [y/not set]     Enable/disable verbose build. Default= non-verbose

NETAPP_OUTPUT_DIR   [path to output] Where NetApp build system should output the built
                                    headers, libraries, modules, and binaries.
                                    DEFAULT: netapp/stage/usr/local
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
1.3) NETAPP.INC:
-------------------
When you build and install NetApp, a file called netapp.inc is generated and
contains all the LDFLAGS and CFLAGS information that your application will need
```

```
to link and use NetApp. Example on how to use netapp.inc can be found in
the examples folder.

--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
2.0) RUNNING NETAPP:
-------------------
Below shows you how to use the basic test app NetAppTester in interactive mode
(the recommended mode).
NOTE: Change the contents of <> with the actual values

    mkdir -p /usr/local
    mount -t nfs <NFS SERVER>:<PATH TO NETAPP>/stage/usr/local /usr/local
    cd /usr/local/bin
    source netapp_target_prep.sh
    ./NetAppTester -i

--------------------------------------------------------------------------------
3.0) WAKE-ON-WLAN TESTING
-------------------
to Test WoWL you need a WiFi dongle that has the special wired up GPIO pins (please consult NetApp
docs for more information). You can test using NetAppTester on the platform (new menu option).
To generate the magic packet, please run on another linux machine the wakeonlan script found in the
etc folder, example useage of this script is:
    ./etc/wakeonlan -i 192.168.1.255  00:90:4C:03:21:23
```

Connecting everything®

BROADCOM®

BROADCOM CORPORATION
5300 California Avenue
Irvine, CA 92617

Phone: 949-926-5000
Fax: 949-926-5203
E-mail: info@broadcom.com
Web: www.broadcom.com

NetApp-SWUM101-R July 27, 2012