



## NetApp 7.1 API Guide

BROADCOM CONFIDENTIAL

## Revision History

<i>Revision</i>	<i>Date</i>	<i>Change Description</i>
NetApp-PG101-R	06/28/12	Updated for NetApp version 7.1.
NetApp-PG100-R	05/04/12	Initial release

Broadcom Corporation  
5300 California Avenue  
Irvine, CA 92617

© 2012 by Broadcom Corporation  
All rights reserved  
Printed in the U.S.A.

Broadcom®, the pulse logo, Connecting everything®, and the Connecting everything logo are among the registered trademarks of Broadcom Corporation and/or its subsidiaries in the United States, certain other countries, and/or the EU. Bluetooth® is a trademark of the Bluetooth SIG. Any other trademarks or trade names mentioned are the property of their respective owners.

Confidential and Proprietary Information: This document and the software are proprietary properties of Broadcom Corporation. This software package may only be used in accordance with the Broadcom Corporation license agreement.

# Table of Contents

<b>About This Document</b>	6
Purpose and Audience	6
Acronyms and Abbreviations	6
Document Conventions	6
References	7
<b>Technical Support</b>	7
<b>Section 1: Introduction</b>	<b>8</b>
<b>Section 2: Module Index</b>	<b>9</b>
Modules	9
<b>Section 3: Data Structure Index</b>	<b>10</b>
Data Structures	10
<b>Section 4: Module Documentation</b>	<b>11</b>
<b>NetApp API Overview</b>	11
Data Structures	11
Modules	12
Defines	12
Typedefs	13
Enumerations	14
Define Documentation	16
Typedef Documentation	17
Enumeration Type Documentation	18
<b>Core</b>	26
Functions	26
Function Documentation	27
<b>Wi-Fi API</b>	38
Modules	38
Functions	38
Function Documentation	39
<b>Wi-Fi Invite</b>	45
Functions	45
Function Documentation	45
<b>Wi-Fi Direct</b>	48
Functions	48

Function Documentation .....	48
<b>Zeroconf (Bonjour) .....</b>	<b>50</b>
Functions.....	50
Function Documentation .....	50
<b>Bluetooth .....</b>	<b>52</b>
Functions.....	52
Function Documentation .....	53
<b>Database APIs .....</b>	<b>57</b>
Functions.....	57
Function Documentation .....	57
<b>Section 5: Data Structure Documentation .....</b>	<b>59</b>
<b>NETAPP_BT_AUDIO_FORMAT Struct Reference .....</b>	<b>59</b>
Data Fields .....	59
Field Documentation .....	59
<b>NETAPP_BT_DEV_INFO Struct Reference .....</b>	<b>60</b>
Data Fields .....	60
Field Documentation .....	61
<b>NETAPP_BT_HID_INFO Struct Reference .....</b>	<b>62</b>
Data Fields .....	62
Field Documentation .....	62
<b>NETAPP_BT_HID_VOICE_INFO Struct Reference.....</b>	<b>63</b>
Data Fields .....	63
Field Documentation .....	63
<b>NETAPP_BT_SETTINGS Struct Reference .....</b>	<b>64</b>
Data Fields .....	64
Field Documentation .....	64
<b>NETAPP_HOTPLUG_DEVICE_INFO Struct Reference .....</b>	<b>65</b>
Data Fields .....	65
Field Documentation .....	65
<b>NETAPP_IFACE_INFO Struct Reference.....</b>	<b>67</b>
Data Fields .....	67
Field Documentation .....	67
<b>NETAPP_INIT_SETTINGS Struct Reference.....</b>	<b>68</b>
Data Fields .....	68
Field Documentation .....	68
<b>NETAPP_INPUT_INFO Struct Reference.....</b>	<b>70</b>

Data Fields .....	70
Field Documentation .....	70
<b>NETAPP_IP_SETTINGS Struct Reference .....</b>	<b>71</b>
Data Fields .....	71
Field Documentation .....	71
<b>NETAPP_OPEN_SETTINGS Struct Reference .....</b>	<b>72</b>
Data Fields .....	72
Field Documentation .....	72
<b>NETAPP_P2P_DISCOVER_PARAMS Struct Reference .....</b>	<b>73</b>
Data Fields .....	73
Field Documentation .....	73
<b>NETAPP_P2P_PEER_INFO Struct Reference .....</b>	<b>74</b>
Data Fields .....	74
Field Documentation .....	74
<b>NETAPP_SETTINGS Struct Reference .....</b>	<b>75</b>
Data Fields .....	75
Field Documentation .....	75
<b>NETAPP_SOFTAP_SETTINGS Struct Reference .....</b>	<b>77</b>
Data Fields .....	77
Field Documentation .....	77
<b>NETAPP_WIFI_AP_INFO Struct Reference .....</b>	<b>77</b>
Data Fields .....	77
Field Documentation .....	78
<b>NETAPP_WOWL_NET_PATTERN Struct Reference .....</b>	<b>80</b>
Data Fields .....	80
Field Documentation .....	80
<b>NETAPP_WOWL_SETTINGS Struct Reference .....</b>	<b>81</b>
Data Fields .....	81
Field Documentation .....	81
<b>sNETAPP_ZEROCONF_SERVICE_INFO Struct Reference .....</b>	<b>82</b>
Data Fields .....	82
Field Documentation .....	82
<b>Index .....</b>	<b>84</b>

## About This Document

### Purpose and Audience

This document describes the NetApp API modules, directories, data structures, and files. The contents are generated from the netapp.h file and are for NetApp version 7.0 (released alongside AppLibs 3.0).

### 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
<b>Bold</b>	User input and actions: for example, type <b>exit</b> , click <b>OK</b> , press <b>Alt+C</b>
Monospace	Code: <code>#include &lt;iostream&gt;</code> HTML: <code>&lt;td rowspan = 3&gt;</code> Command line commands and parameters: <code>w1 [-1] &lt;command&gt;</code>
<code>&lt; &gt;</code>	Placeholders for <i>required</i> elements: enter your <code>&lt;username&gt;</code> or <code>w1 &lt;command&gt;</code>
<code>[ ]</code>	Indicates <i>optional</i> command-line parameters: <code>w1 [-1]</code> Indicates bit and byte ranges (inclusive): <code>[0:3]</code> or <code>[7:0]</code>

## References

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



**Note:** 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.

<i>Document Title</i>	<i>Number</i>	<i>Source</i>
<b><i>Broadcom Documents</i></b>		
[1] <i>NetApp User Guide</i>	NetApp-SWUM1XX-R	DocSAFE
[2] <i>Wake-On-Wireless-LAN Features and Requirements</i>	WoWL-AN1XX-R	DocSAFE
<b><i>Other Documents</i></b>		
[3] <i>Wi-Fi Protected Setup™ Test Plan</i>	–	Wi-Fi Alliance®
[4] <i>Wi-Fi Protected Setup Specifications 1.0h</i>		

---

## 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: Introduction

The network API is designed to abstract the Linux® IPv4 network stack to allow the application to configure networking devices like wired, wireless, or Bluetooth® with a thin and simple API.

NetApp currently supports these features:

- 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)



## Section 2: Module Index

---

### Modules

Here is a list of all modules:

- NetApp API Overview
- Core
- Wi-Fi API
- Wi-Fi Invite
- Wi-Fi Direct
- Zeroconf (Bonjour)
- Bluetooth
- Database APIs

## Section 3: Data Structure Index

### Data Structures

Here are the data structures with brief descriptions:

- NETAPP\_BT\_AUDIO\_FORMAT — Bluetooth Audio Format Information
- NETAPP\_BT\_DEV\_INFO — Bluetooth Device Information
- NETAPP\_BT\_HID\_INFO — Bluetooth HID Information
- NETAPP\_BT\_HID\_VOICE\_INFO — HID Voice Info structure
- NETAPP\_BT\_SETTINGS — Bluetooth Settings
- NETAPP\_HOTPLUG\_DEVICE\_INFO — USB hotplug information sent when NetApp detects a hotplug event.
- NETAPP\_IFACE\_INFO — Interface information
- NETAPP\_INIT\_SETTINGS — NetApp Initialization Settings Structure
- NETAPP\_INPUT\_INFO — Input Event information
- NETAPP\_IP\_SETTINGS — NetApp Settings. This structure contains the network configuration settings.
- NETAPP\_OPEN\_SETTINGS — NetApp Open Settings Structure
- NETAPP\_P2P\_DISCOVER\_PARAMS — Parameters for a Wi-Fi Direct Discovery
- NETAPP\_P2P\_PEER\_INFO — Wi-Fi Direct Peer Info
- NETAPP\_SETTINGS — General NetApp Settings Structure
- NETAPP\_SOFTAP\_SETTINGS — SoftAp Settings
- NETAPP\_WIFI\_AP\_INFO — NetApp Wi-Fi Access Point Information
- NETAPP\_WOWL\_NET\_PATTERN — WoWL Net Pattern Info
- NETAPP\_WOWL\_SETTINGS — WoWL Settings
- sNETAPP\_ZEROCONF\_SERVICE\_INFO — Zero Configuration Service Information

## Section 4: Module Documentation

### NetApp API Overview

The NetApp API is a collection of APIs that are used to control and configure the wired and wireless network interfaces.

### Data Structures

- struct **NETAPP\_IFACE\_INFO**  
Interface information.
- struct **NETAPP\_IP\_SETTINGS**  
NetApp Settings. This structure contains the network configuration settings.
- struct **NETAPP\_WIFI\_AP\_INFO**  
NetApp Wi-Fi Access Point Information.
- struct **NETAPP\_SOFTAP\_SETTINGS**  
SoftAp Settings.
- struct **NETAPP\_WOWL\_NET\_PATTERN**  
WoWL Net Pattern Info.
- struct **NETAPP\_P2P\_DISCOVER\_PARAMS**  
Parameters for a Wi-Fi Direct Discovery.
- struct **NETAPP\_P2P\_PEER\_INFO**  
Wi-Fi Direct Peer Info.
- struct **NETAPP\_WOWL\_SETTINGS**  
WoWL Settings.
- struct **NETAPP\_BT\_SETTINGS**  
Bluetooth Settings.
- struct **NETAPP\_BT\_HID\_INFO**  
Bluetooth HID Information.
- struct **NETAPP\_BT\_HID\_VOICE\_INFO**  
HID Voice Info structure.
- struct **NETAPP\_BT\_DEV\_INFO**  
Bluetooth Device Information.
- struct **NETAPP\_HOTPLUG\_DEVICE\_INFO**  
USB hotplug information sent when NetApp detects a hotplug event.
- struct **NETAPP\_BT\_AUDIO\_FORMAT**  
Bluetooth Audio Format Information.

- struct **NETAPP\_INPUT\_INFO**  
Input Event information.
- struct **NETAPP\_INIT\_SETTINGS**  
NetApp Initialization Settings Structure.
- struct **NETAPP\_OPEN\_SETTINGS**  
NetApp Open Settings Structure.
- struct **NETAPP\_SETTINGS**  
General NetApp Settings Structure.
- struct **sNETAPP\_ZEROCONF\_SERVICE\_INFO**  
Zero Configuration Service Information.

## Modules

- **Core** — A set of Core APIs are used to control and configure all interfaces.
- **Wi-Fi API** — This includes APIs used to control and configure the wireless interface.
- **Zeroconf (Bonjour)** — Zero Configuration (Bonjour) library to support Multicast-DNS and DNS-Service Discovery.
- **Bluetooth** — Bluetooth library to support various Bluetooth profiles (HID, AV, etc.)
- **Database APIs** — API to fetch information from the built-in database back end.

## Defines

- #define **NETAPP\_VERSION\_MAJOR** 7  
NetApp major version .
- #define **NETAPP\_VERSION\_MINOR** 1  
NetApp inc version.
- #define **NETAPP\_VERSION\_INC** 0
- #define **NETAPP\_ENET\_LEN** 17  
Ethernet address, e.g., 00:00:00:00:00:00.
- #define **NETAPP\_HW\_ADDR\_LEN** 6  
Hardware address length.
- #define **NETAPP\_NO\_WAIT** 0  
Not wait.
- #define **NETAPP\_WAIT\_FOREVER** -1  
Wait forever.
- #define **NETAPP\_IFACE\_NAME\_LEN** 10  
Interface name length.
- #define **NETAPP\_WOWL\_NET\_PATTERN\_MAX\_LENGTH** 128  
Maximum size of a net pattern.
- #define **NETAPP\_WOWL\_MAX\_NET\_PATTERNS** 4  
Maximum # net patterns we can set.

- **#define NETAPP\_BT\_NAME\_LEN 248**  
Length of a Bluetooth device name.
- **#define NETAPP\_MAX\_SSID\_LEN 32**  
Maximum SSID name.
- **#define NETAPP\_MAX\_PASSWORD\_LEN 64**  
Maximum password length.
- **#define NETAPP\_UUID\_LEN 16**  
Length of a UUID in Bytes.
- **#define NETAPP\_ZEROCONF\_NAME\_LEN 32**  
Length of service type and name.
- **#define NETAPP\_BT\_PIN\_CODE\_LEN 128**  
Length of a pin code.
- **#define NETAPP\_HID\_DSCINFO\_MAX 800**
- **#define NETAPP\_LINK\_KEY\_LEN 16**
- **#define NETAPP\_BT\_HID\_AUDIO\_FILENAME\_LEN 50**  
Max filename path.
- **#define BT\_DEVICE\_FEATURE\_LEN 8**  
Length of Bluetooth device features list.

## Typedefs

- **typedef uint8\_t NETAPP\_HW\_ADDR [NETAPP\_HW\_ADDR\_LEN]**  
Hardware Address (MAC or BD/Bluetooth)
- **typedef void \* NETAPP\_HANDLE**  
NetApp Module Handle.
- **typedef uint32\_t NETAPP\_IPV4\_ADDR**  
IPv4 Internet address type definition. The networking stack used for the BCM7XXX family of chips uses an unsigned 32-bit integer.
- **typedef void(\* NETAPP\_CALLBACK )(void \*pParam, NETAPP\_CB\_TYPE tCbType, const void \*pvBuffer, uint32\_t ulData0, NETAPP\_RETCODE tResult, NETAPP\_IFACE tIFace)**  
NetApp Wi-Fi Callback.
- **typedef struct**
- **sNETAPP\_ZEROCONF\_SERVICE\_INFO NETAPP\_ZEROCONF\_SERVICE\_INFO**  
Zero Configuration Service Information.

## Enumerations

- enum **NETAPP\_RETCODE** { **NETAPP\_SUCCESS** = 0, **NETAPP\_FAILURE**, **NETAPP\_INVALID\_PARAMETER**, **NETAPP\_NULL\_PTR**, **NETAPP\_OUT\_OF\_MEMORY**, **NETAPP\_NOT\_IMPLEMENTED**, **NETAPP\_NETWORK\_UNREACHABLE**, **NETAPP\_SOCKET\_ERROR**, **NETAPP\_TIMEOUT**, **NETAPP\_DHCP\_FAILURE**, **NETAPP\_HOST\_NOT\_FOUND**, **NETAPP\_CANCELED**, **NETAPP\_INCORRECT\_PASSWORD**, **NETAPP\_INVALID\_PIN**, **NETAPP\_NOT\_FOUND**, **NETAPP\_NOT\_SUPPORTED**, **NETAPP\_WPS\_MULTIPLE\_AP\_FOUND**, **NETAPP\_SCAN\_EMPTY**, **NETAPP\_INVALID\_STATE**, **NETAPP\_WPS\_2\_ERR\_INCOMPATIBLE** }

The return code for most NetApp APIs.

- enum **NETAPP\_IFACE** { **NETAPP\_IFACE\_WIRED**, **NETAPP\_IFACE\_ETH0** = **NETAPP\_IFACE\_WIRED**, **NETAPP\_IFACE\_ETH1**, **NETAPP\_IFACE\_ETH2**, **NETAPP\_IFACE\_ETH3**, **NETAPP\_IFACE\_ETH4**, **NETAPP\_IFACE\_ETH5**, **NETAPP\_IFACE\_WIRED\_MAX**, **NETAPP\_IFACE\_WIRELESS**, **NETAPP\_IFACE\_LOOPBACK**, **NETAPP\_IFACE\_P2P**, **NETAPP\_IFACE\_BLUETOOTH**, **NETAPP\_IFACE\_MAX** }

Which interface to use, wired or wireless.

- enum **NETAPP\_IP\_MODE** { **NETAPP\_IP\_MODE\_OFF** = 0, **NETAPP\_IP\_MODE\_STATIC**, **NETAPP\_IP\_MODE\_DYNAMIC**, **NETAPP\_IP\_MODE\_AUTO\_IP** }

Network Access settings.

- enum **NETAPP\_WIFI\_SECURITY** { **NETAPP\_WIFI\_SECURITY\_INVALID** = 0, **NETAPP\_WIFI\_SECURITY\_AUTO\_DETECT**, **NETAPP\_WIFI\_SECURITY\_NONE**, **NETAPP\_WIFI\_SECURITY\_WEP**, **NETAPP\_WIFI\_SECURITY\_WPA\_PSK\_AES**, **NETAPP\_WIFI\_SECURITY\_WPA\_PSK\_TKIP**, **NETAPP\_WIFI\_SECURITY\_WPA2\_PSK\_AES**, **NETAPP\_WIFI\_SECURITY\_WPA2\_PSK\_TKIP**, **NETAPP\_WIFI\_SECURITY\_NOT\_SUPPORTED** }

Wi-Fi Security Type.

- enum **NETAPP\_WIFI\_802\_11\_MODE** { **NETAPP\_WIFI\_802\_11\_NONE** = 0x0000, **NETAPP\_WIFI\_802\_11\_MODE\_A** = 0x0001, **NETAPP\_WIFI\_802\_11\_MODE\_B** = 0x0002, **NETAPP\_WIFI\_802\_11\_MODE\_G** = 0x0004, **NETAPP\_WIFI\_802\_11\_MODE\_N** = 0x0008 }

Wi-Fi IEEE 802.11 Modes.

- enum **NETAPP\_WIFI\_RSSI** { **NETAPP\_WIFI\_RSSI\_NONE** = 0, **NETAPP\_WIFI\_RSSI\_POOR**, **NETAPP\_WIFI\_RSSI\_FAIR**, **NETAPP\_WIFI\_RSSI\_GOOD**, **NETAPP\_WIFI\_RSSI\_EXCELLENT** }

- Wi-Fi Received Signal Strength Indicator.*

- enum **NETAPP\_LINK\_STATE** { **NETAPP\_LINK\_DOWN** = 0, **NETAPP\_LINK\_UP**, **NETAPP\_LINK\_ACQUIRING** }

Link Status.

- enum **NETAPP\_WIFI\_BANDWIDTH** { **NETAPP\_WIFI\_BANDWIDTH\_INVALID**, **NETAPP\_WIFI\_BANDWIDTH\_10MHz**, **NETAPP\_WIFI\_BANDWIDTH\_20MHz**, **NETAPP\_WIFI\_BANDWIDTH\_40MHz** }

Wi-Fi Channel Bandwidth.

- enum **NETAPP\_CB\_TYPE** { **NETAPP\_CB\_INVALID** = 0, **NETAPP\_CB\_LINK**, **NETAPP\_CB\_CONNECT**, **NETAPP\_CB\_DISCONNECT**, **NETAPP\_CB\_INPUT\_EVENT**, **NETAPP\_CB\_PING**, **NETAPP\_CB\_DNSLOOKUP**, **NETAPP\_CB\_INVITE**, **NETAPP\_CB\_SCAN\_DONE**, **NETAPP\_CB\_SCANNED\_APIINFO**, **NETAPP\_CB\_FETCHED\_APIINFO**, **NETAPP\_CB\_NTPDATE**, **NETAPP\_CB\_SETSETTINGS**, **NETAPP\_CB\_HOTPLUG**, **NETAPP\_CB\_RSSI\_EVENT**, **NETAPP\_CB\_ZEROCONF**, **NETAPP\_CB\_P2P\_PEER**, **NETAPP\_CB\_P2P\_CONNECT**, **NETAPP\_CB\_BT\_DISCOVERY\_RESULTS**, **NETAPP\_CB\_BT\_SP\_CONFIRM\_REQ**, **NETAPP\_CB\_BT\_SP\_NOTIFY**, **NETAPP\_CB\_BT\_AUTH\_COMPLETE**, **NETAPP\_CB\_BT\_HID\_VOICE\_INFO**, **NETAPP\_CB\_VOICE\_REC\_DONE**, **NETAPP\_CB\_DHCP\_LEASE\_RESPONSE**, **NETAPP\_CB\_BT\_AVK\_STATE**, **NETAPP\_CB\_BT\_AVK\_CHUNK**, **NETAPP\_CB\_DYING**, **NETAPP\_CB\_MAX** = **NETAPP\_CB\_DYING** }  
Callback Type.
- enum **NETAPP\_BT\_AVK\_STATE** { **NETAPP\_BT\_AVK\_STATE\_PLAY**, **NETAPP\_BT\_AVK\_STATE\_STOP** }  
AVK State notification from the AV Source device.
- enum **NETAPP\_ZEROCONF\_SERVICE\_STATE** { **NETAPP\_ZEROCONF\_SERVICE\_FOUND**, **NETAPP\_ZEROCONF\_SERVICE\_REMOVED** }  
The Browsed service state "hotplug" information (inserted or removed).
- enum **NETAPP\_DEVICE\_TYPE** { **NETAPP\_DEVICE\_TYPE\_OTHER**, **NETAPP\_DEVICE\_TYPE\_DTV**, **NETAPP\_DEVICE\_TYPE\_BD** }  
The P2P device type.
- enum **NETAPP\_P2P\_SERVICES** { **NETAPP\_P2P\_SVC\_NONE** = 0, **NETAPP\_P2P\_SVC\_FILE\_TX** = 0x001, **NETAPP\_P2P\_SVC\_PRINT** = 0x0002, **NETAPP\_P2P\_SVC\_DISPLAY** = 0x0004, **NETAPP\_P2P\_SVC\_ALL** }  
Wi-Fi Direct Service List.
- enum **NETAPP\_WOWL\_EVENT** { **NETAPP\_WOWL\_EVENT\_NONE** = 0x00, **NETAPP\_WOWL\_EVENT\_MAGIC\_PATTERN** = 0x01, **NETAPP\_WOWL\_EVENT\_DISASSOC\_DEAUTH** = 0x02, **NETAPP\_WOWL\_EVENT\_LOSS\_OF\_BEACON** = 0x04, **NETAPP\_WOWL\_EVENT\_NET\_PATTERN** = 0x08 }  
Wake-on-Wireless-LAN Wakeup Event Type.
- enum **NETAPP\_BT\_SERVICE\_TYPE** { **NETAPP\_BT\_SERVICE\_NONE** = 0x0000, **NETAPP\_BT\_SERVICE\_HID** = 0x0001, **NETAPP\_BT\_SERVICE\_HSP** = 0x0002, **NETAPP\_BT\_SERVICE\_HFP** = 0x0004, **NETAPP\_BT\_SERVICE\_OPP** = 0x0008, **NETAPP\_BT\_SERVICE\_FTP** = 0x0010, **NETAPP\_BT\_SERVICE\_A2DP** = 0x0020, **NETAPP\_BT\_SERVICE\_AVRCP** = 0x0040, **NETAPP\_BT\_SERVICE\_ALL** = 0xffff }  
Bluetooth Service Type.
- enum **NETAPP\_BT\_SP\_EVENT** { **NETAPP\_BT\_SP\_CONFIRM\_REQUEST**, **NETAPP\_BT\_SP\_NOTIFY** }  
Bluetooth Simple Pairing Notification Event.
- enum **NETAPP\_HOTPLUG\_ACTION** { **NETAPP\_HOTPLUG\_ADD**, **NETAPP\_HOTPLUG\_REMOVE** }  
Hotplug Action Type (Add/Remove)
- enum **NETAPP\_HOTPLUG\_DEVICE\_TYPE** { **NETAPP\_HOTPLUG\_DEVICE\_USB\_INPUT**, **NETAPP\_HOTPLUG\_DEVICE\_USB**, **NETAPP\_HOTPLUG\_DEVICE\_BLUETOOTH**, **NETAPP\_HOTPLUG\_DEVICE\_WIFI** }  
Hotplug Device Type.
- enum **NETAPP\_BT\_AV\_MODE** { **NETAPP\_BT\_AV\_MODE\_NONE** = 0, **NETAPP\_BT\_AV\_MODE\_MONO**, **NETAPP\_BT\_AV\_MODE\_STEREO** }  
AV Audio mode (number of channels)

## Define Documentation

**#define BT\_DEVICE\_FEATURE\_LEN 8**

Length of Bluetooth device features list.

**#define NETAPP\_BT\_HID\_AUDIO\_FILENAME\_LEN 50**

Max filename path.

**#define NETAPP\_BT\_NAME\_LEN 248**

Length of a Bluetooth device name.

**#define NETAPP\_BT\_PIN\_CODE\_LEN 128**

Length of a pin code.

**#define NETAPP\_ENET\_LEN 17**

Ethernet address, e.g., 00:00:00:00:00:00.

Length of bytes for displaying an

**#define NETAPP\_HID\_DSCINFO\_MAX 800****#define NETAPP\_HW\_ADDR\_LEN 6**

Hardware address length.

**#define NETAPP\_IFACE\_NAME\_LEN 10**

Interface name length.

**#define NETAPP\_LINK\_KEY\_LEN 16****#define NETAPP\_MAX\_PASSWORD\_LEN 64**

Maximum password length.

**#define NETAPP\_MAX\_SSID\_LEN 32**

Maximum SSID name.

**#define NETAPP\_NO\_WAIT 0**

Not wait.

**#define NETAPP\_UUID\_LEN 16**

Length of a UUID in Bytes.

**#define NETAPP\_VERSION\_INC 0**

NetApp inc version.

**#define NETAPP\_VERSION\_MAJOR 7**

NetApp major version.

**#define NETAPP\_VERSION\_MINOR 1**

NetApp minor version.



**#define NETAPP\_WAIT\_FOREVER -1**

Wait forever.

**#define NETAPP\_WOWL\_MAX\_NET\_PATTERNS 4**

Maximum # net patterns we can set.

**#define NETAPP\_WOWL\_NET\_PATTERN\_MAX\_LENGTH 128**

Maximum size of a net pattern.

**#define NETAPP\_ZEROCONF\_NAME\_LEN 32**

Length of service type and name.

## Typedef Documentation

**typedef void(\* NETAPP\_CALLBACK)(void \*pParam,NETAPP\_CB\_TYPE tCbType,const void \*pvBuffer,uint32\_t ulData0,NETAPP\_RETCODE tResult,NETAPP\_IFACE tIFace)**

NetApp Wi-Fi Callback.

Refer to the *NetApp User Guide* ([Reference \[1\] on page 7](#)) for more information on the parameters passed to each NetApp Callback. Callback Info structure.

**typedef void\* NETAPP\_HANDLE**

NetApp Module Handle.

**typedef uint8\_t NETAPP\_HW\_ADDR[NETAPP\_HW\_ADDR\_LEN]**

Hardware Address (MAC or BD/Bluetooth)

**typedef uint32\_t NETAPP\_IPV4\_ADDR**

IPv4 Internet address type definition. The networking stack used for the BCM7XXX family of chips uses an unsigned 32-bit integer.

**typedef struct sNETAPP\_ZEROCONF\_SERVICE\_INFO NETAPP\_ZEROCONF\_SERVICE\_INFO**

Zero Configuration Service Information.

The following structure is passed in the callback `NETAPP_CB_ZEROCONF_SERVICE` when we browse for a service and a service is found. The service information is cached inside of **NetApp API Overview** and you can get a reference to the cached data by calling **NetAppZeroConfGetBrowseResults()**.

### Remarks:

DO NOT free this structure; NetApp will take care of cleaning up.

## Enumeration Type Documentation

### enum NETAPP\_BT\_AV\_MODE

AV Audio mode (number of channels)

**Enumerator:**

**NETAPP\_BT\_AV\_MODE\_NONE** None (invalid case)

**NETAPP\_BT\_AV\_MODE\_MONO** Mono.

**NETAPP\_BT\_AV\_MODE\_STEREO** Stereo.

### enum NETAPP\_BT\_AVK\_STATE

AVK State notification from the AV Source device.

**Enumerator:**

**NETAPP\_BT\_AVK\_STATE\_PLAY** Received the Play notification.

**NETAPP\_BT\_AVK\_STATE\_STOP** Received the Stop notification.

### enum NETAPP\_BT\_SERVICE\_TYPE

Bluetooth Service Type.

List of the possible service types discovered or supported

**Enumerator:**

**NETAPP\_BT\_SERVICE\_NONE** None.

**NETAPP\_BT\_SERVICE\_HID** Human Interface Device.

**NETAPP\_BT\_SERVICE\_HSP** Headset profile.

**NETAPP\_BT\_SERVICE\_HFP** Hands-free profile.

**NETAPP\_BT\_SERVICE\_OPP** Object push.

**NETAPP\_BT\_SERVICE\_FTP** File transfer.

**NETAPP\_BT\_SERVICE\_A2DP** Advanced audio distribution.

**NETAPP\_BT\_SERVICE\_AVRCP** A/V remote control.

**NETAPP\_BT\_SERVICE\_ALL** All Services.

### enum NETAPP\_BT\_SP\_EVENT

Bluetooth Simple Pairing Notification Event.

This enum is passed as the ulData0 in the NETAPP\_CB\_BT\_SIMPLE\_PAIRING callback.

**Enumerator:**

**NETAPP\_BT\_SP\_CONFIRM\_REQUEST** Notify the user that they must accept or reject a simple pairing request.

**NETAPP\_BT\_SP\_NOTIFY** Inform the application of a simple pairing notification event.

**enum NETAPP\_CB\_TYPE**

Callback Type.

List of supported callbacks from events that occur in the NetApp API.

**Enumerator:**

**NETAPP\_CB\_INVALID** Initialization for this enum.

**NETAPP\_CB\_LINK** Link change Event:

pvBuffer: N/A

ulData0: NETAPP\_LINK\_STATE

**NETAPP\_CB\_CONNECT** Connection results for Wi-Fi or Bluetooth

pvBuffer: Pointer to either the **NETAPP\_WIFI\_AP\_INFO** or **NETAPP\_BT\_DEV\_INFO** structure

ulData0: NETAPP\_BT\_SERVICE\_TYPE for Bluetooth callbacks

**NETAPP\_CB\_DISCONNECT** Disconnection results for Bluetooth.

pvBuffer: **NETAPP\_BT\_DEV\_INFO** for Bluetooth

ulData0: N/A

**NETAPP\_CB\_INPUT\_EVENT** AV Remote control from Bluetooth audio

pvBuffer: Pointer to the **NETAPP\_INPUT\_INFO** structure

ulData0: N/A

**NETAPP\_CB\_PING** Results from a Ping request.

pvBuffer: The server name passed to the ping request

ulData0: N/A

**NETAPP\_CB\_DNSLOOKUP** DNSLookup results.

pvBuffer: The server name passed to the lookup request

ulData0: N/A

**NETAPP\_CB\_INVITE** Wi-Fi Invite request received.

pvBuffer: The SSID from the inviting device

ulData0: N/A

**NETAPP\_CB\_SCAN\_DONE** A scan is complete and results are available.

pvBuffer: N/A

ulData0:

**NETAPP\_CB\_SCANNED\_APINFO** A scan is complete and scanned AP info is included.

pvBuffer: The server name passed to the lookup request

ulData0: Scan count (for background scans)

**NETAPP\_CB\_FETCHED\_APINFO** Received the results from the API **NetAppWiFiGetApInfo()**

pvBuffer: Pointer to the **NETAPP\_WIFI\_AP\_INFO**

ulData0: N/A

**NETAPP\_CB\_NTPDATE** NTPDate request is finished with results.

pvBuffer: N/A

ulData0: N/A

**NETAPP\_CB\_SETSETTINGS** The result from a call to **NetAppSetNetworkSettings()**.

pvBuffer: N/A

ulData0: N/A

**NETAPP\_CB\_HOTPLUG** NetApp Detected a hotplug.

pvBuffer: Pointer to the **NETAPP\_HOTPLUG\_DEVICE\_INFO** structure

ulData0: N/A

**NETAPP\_CB\_RSSI\_EVENT** The RSSI of a connected AP changed levels.

pvBuffer: N/A

ulData0: The RSSI (**NETAPP\_WIFI\_RSSI**)

**NETAPP\_CB\_ZEROCONF** NetApp has found a browse/discovery request.

service or the service is removed. pvBuffer: Service name

ulData0: Service state (**NETAPP\_ZEROCONF\_SERVICE\_STATE**)

**NETAPP\_CB\_P2P\_PEER** Discovered Wi-Fi Direct Peer information.

pvBuffer: Pointer to the peer info structure **NETAPP\_P2P\_PEER\_INFO**

ulData0: Discovery count

**NETAPP\_CB\_P2P\_CONNECT** Wi-Fi Direct Connection is established.

pvBuffer: Pointer to the peer info structure **NETAPP\_P2P\_PEER\_INFO**

ulData0: N/A

**NETAPP\_CB\_BT\_DISCOVERY\_RESULTS** Bluetooth discovery is complete and results are available.

pvBuffer: N/A

ulData0: N/A

**NETAPP\_CB\_BT\_SP\_CONFIRM\_REQ** Simple pairing confirm request

The user must then Accept or Reject the SP request.

pvBuffer: Pointer to the device info structure **NETAPP\_BT\_DEV\_INFO**

ulData0: Simple pairing password key

**NETAPP\_CB\_BT\_SP\_NOTIFY** Simple pairing notification.

pvBuffer: N/A

ulData0: Simple pairing key

**NETAPP\_CB\_BT\_AUTH\_COMPLETE** Bluetooth authentication.

pvBuffer: N/A

ulData0: Simple pairing key

**NETAPP\_CB\_BT\_HID\_VOICE\_INFO** Notify the application a file has been created for HID audio.

pvBuffer: Pointer to the **NETAPP\_BT\_HID\_VOICE\_INFO** structure

ulData0: N/A

**NETAPP\_CB\_VOICE\_REC\_DONE** Finished a voice recognition request and a string is available.

pvBuffer: Voice recognized string (char\*)

ulData0: N/A

**NETAPP\_CB\_DHCP\_LEASE\_RESPONSE** Responded to a DHCP lease request when NetApp is DHCP server.  
(SoftAP or P2P)

pvBuffer: N/A

ulData0: IP Address in IPv4 notation

**NETAPP\_CB\_BT\_AVK\_STATE** Received an AVK state change notification that must be acted upon.

pvBuffer: Pointer to the **NETAPP\_BT\_DEV\_INFO** structure

ulData0: State information (PLAY, PAUSE, STOP, etc.) **NETAPP\_BT\_AVK\_STATE**

**NETAPP\_CB\_BT\_AVK\_CHUNK** Received Audio buffer that needs to be pushed to some playback engine

pvBuffer: Pointer to the received PCM data

ulData0: Buffer size

**NETAPP\_CB\_DYING** NetApp encountered a fatal error and cannot recover.

**NETAPP\_CB\_MAX** End of the callback list.

#### enum NETAPP\_DEVICE\_TYPE

The P2P device type.

**Enumerator:**

**NETAPP\_DEVICE\_TYPE\_OTHER** Device type is not specified or is not one of those in the list below.

**NETAPP\_DEVICE\_TYPE\_DTV** Digital Television.

**NETAPP\_DEVICE\_TYPE\_BD** Blu-ray Player.

#### enum NETAPP\_HOTPLUG\_ACTION

Hotplug Action Type (Add/Remove)

**Enumerator:**

**NETAPP\_HOTPLUG\_ADD** The device is inserted/added.

**NETAPP\_HOTPLUG\_REMOVE** Device has been removed.

#### enum NETAPP\_HOTPLUG\_DEVICE\_TYPE

Hotplug Device Type.

**Enumerator:**

**NETAPP\_HOTPLUG\_DEVICE\_USB\_INPUT** USB Input Device.

**NETAPP\_HOTPLUG\_DEVICE\_USB** Lower level USB device information.

**NETAPP\_HOTPLUG\_DEVICE\_BLUETOOTH** Bluetooth device.

**NETAPP\_HOTPLUG\_DEVICE\_WIFI** Wi-Fi Interface.

**enum NETAPP\_IFACE**

Determines which interface to use: wired or wireless.

**Enumerator:**

**NETAPP\_IFACE\_WIRED** Backwards compatibility.

**NETAPP\_IFACE\_ETH0** 1st Wired

**NETAPP\_IFACE\_ETH1** 2nd Wired

**NETAPP\_IFACE\_ETH2** 3rd Wired

**NETAPP\_IFACE\_ETH3** 4th Wired

**NETAPP\_IFACE\_ETH4** 5th Wired

**NETAPP\_IFACE\_ETH5** 6th Wired

**NETAPP\_IFACE\_WIRED\_MAX** Number of Wired interfaces (used internally).

**NETAPP\_IFACE\_WIRELESS** Wireless (might be remapped to an ETHx interface).

**NETAPP\_IFACE\_LOOPBACK** Loopback (LO)

**NETAPP\_IFACE\_P2P** Wi-Fi Direct.

**NETAPP\_IFACE\_BLUETOOTH** Bluetooth.

**NETAPP\_IFACE\_MAX**

**enum NETAPP\_IP\_MODE**

Network Access settings.

**Enumerator:**

**NETAPP\_IP\_MODE\_OFF** Network off.

**NETAPP\_IP\_MODE\_STATIC** Network static.

**NETAPP\_IP\_MODE\_DYNAMIC** Network dynamic using dhcpcd.

**NETAPP\_IP\_MODE\_AUTO\_IP** RFC 3927-compliant IPv4LL.

**enum NETAPP\_LINK\_STATE**

Link Status.

**Enumerator:**

**NETAPP\_LINK\_DOWN** Network link is down.

**NETAPP\_LINK\_UP** Network link is up and IP address is obtained.

**NETAPP\_LINK\_ACQUIRING** In the process of fetching the IP address from DHCPD.

**enum NETAPP\_P2P\_SERVICES**

Wi-Fi Direct Service List.

Enum to tell the other device what type of device we are.

**Enumerator:**

**NETAPP\_P2P\_SVC\_NONE**

**NETAPP\_P2P\_SVC\_FILE\_TX** File Transfer.

**NETAPP\_P2P\_SVC\_PRINT** Print service.

**NETAPP\_P2P\_SVC\_DISPLAY** Display.

**NETAPP\_P2P\_SVC\_ALL** All services.

**enum NETAPP\_RETCODE**

The return code for most NetApp APIs.

**Enumerator:**

**NETAPP\_SUCCESS** Success.

**NETAPP\_FAILURE** General failure.

**NETAPP\_INVALID\_PARAMETER** Invalid parameter.

**NETAPP\_NULL\_PTR** Null handle detected or invalid state.

**NETAPP\_OUT\_OF\_MEMORY** Malloc has failed.

**NETAPP\_NOT\_IMPLEMENTED** Function not implemented.

**NETAPP\_NETWORK\_UNREACHABLE** Unable to reach destination network.

**NETAPP\_SOCKET\_ERROR** Error creating the Linux socket.

**NETAPP\_TIMEOUT** Timeout error occurred.

**NETAPP\_DHCP\_FAILURE** Failure to fetch DHCPD address.

**NETAPP\_HOST\_NOT\_FOUND** Not able to find host in DNS server.

**NETAPP\_CANCELED** The function was canceled.

**NETAPP\_INCORRECT\_PASSWORD** Incorrect password provided.

**NETAPP\_INVALID\_PIN** Invalid WPS pin used.

**NETAPP\_NOT\_FOUND** Tried to execute system command and the search string was not found.

**NETAPP\_NOT\_SUPPORTED** Requesting an API or function that was not supported/compiled in.

**NETAPP\_WPS\_MULTIPLE\_AP\_FOUND** Found more than one AP in WPS PBC (overlap).

**NETAPP\_SCAN\_EMPTY** Scan was complete and no access points found.

**NETAPP\_INVALID\_STATE** Calling the API when the system is in an invalid state.

**NETAPP\_WPS\_2\_ERR\_INCOMPATIBLE** WPS detected an AP that support a WPS 1.0 deprecated setting that is not supported in WPS 2.0. The application should restart WPS with **NETAPP\_SETTINGS.bwps2\_0** set to false.

**enum NETAPP\_WIFI\_802\_11\_MODE**

Wi-Fi IEEE 802.11 Modes.

**Enumerator:**

**NETAPP\_WIFI\_802\_11\_NONE** None are supported (invalid)

**NETAPP\_WIFI\_802\_11\_MODE\_A** IEEE 802.11A.

**NETAPP\_WIFI\_802\_11\_MODE\_B** IEEE 802.11B.

**NETAPP\_WIFI\_802\_11\_MODE\_G** IEEE 802.11G.

**NETAPP\_WIFI\_802\_11\_MODE\_N** IEEE 802.11N.

**enum NETAPP\_WIFI\_BANDWIDTH**

Wi-FiChannel Bandwidth.

**Enumerator:**

**NETAPP\_WIFI\_BANDWIDTH\_INVALID** Invalid bandwidth setting.

**NETAPP\_WIFI\_BANDWIDTH\_10MHz** 10 MHz

**NETAPP\_WIFI\_BANDWIDTH\_20MHz** 20 MHz

**NETAPP\_WIFI\_BANDWIDTH\_40MHz** 40 MHz

**enum NETAPP\_WIFI\_RSSI**

Wi-Fi Received Signal Strength Indicator.

**Enumerator:**

**NETAPP\_WIFI\_RSSI\_NONE** No signal (0 bar)

**NETAPP\_WIFI\_RSSI\_POOR** Poor (1 bar)

**NETAPP\_WIFI\_RSSI\_FAIR** Fair (2 bars)

**NETAPP\_WIFI\_RSSI\_GOOD** Good (3 bars)

**NETAPP\_WIFI\_RSSI\_EXCELLENT** Excellent (4 bars)

**enum NETAPP\_WIFI\_SECURITY**

Wi-Fi Security Type.

**Enumerator:**

**NETAPP\_WIFI\_SECURITY\_INVALID** The security is not set or Invalid.

**NETAPP\_WIFI\_SECURITY\_AUTO\_DETECT** Auto-detect the security type.

**NETAPP\_WIFI\_SECURITY\_NONE** No Security.

**NETAPP\_WIFI\_SECURITY\_WEP** Shared or Open, WEP.

**NETAPP\_WIFI\_SECURITY\_WPA\_PSK\_AES** WPA-Personal, AES encryption.

**NETAPP\_WIFI\_SECURITY\_WPA\_PSK\_TKIP** WPA-Personal, TKIP encryption.

**NETAPP\_WIFI\_SECURITY\_WPA2\_PSK\_AES** WPA2-Personal, AES encryption.

**NETAPP\_WIFI\_SECURITY\_WPA2\_PSK\_TKIP** WPA-Personal, TKIP encryption.

**NETAPP\_WIFI\_SECURITY\_NOT\_SUPPORTED** Security format not supported.

**enum NETAPP\_WOWL\_EVENT**

Wake-on-Wireless-LAN Wakeup Event Type.

**Enumerator:**

**NETAPP\_WOWL\_EVENT\_NONE** Do not wake up on any event.

**NETAPP\_WOWL\_EVENT\_MAGIC\_PATTERN** Wake up on magic pattern.

**NETAPP\_WOWL\_EVENT\_DISASSOC\_DEAUTH** Wake up on disassociate from AP.

**NETAPP\_WOWL\_EVENT\_LOSS\_OF\_BEACON** Wake up on loss of beacon.

**NETAPP\_WOWL\_EVENT\_NET\_PATTERN** Wake up on a special net pattern.



**enum NETAPP\_ZEROCONF\_SERVICE\_STATE**

The Browsed service state "hotplug" information (inserted or removed).

**Enumerator:**

**NETAPP\_ZEROCONF\_SERVICE\_FOUND** Service name is found (discovered).

**NETAPP\_ZEROCONF\_SERVICE\_REMOVED** Service name was removed, no longer available.

---

## Core

A set of Core APIs are used to control and configure all interfaces.

## Functions

- **NETAPP\_RETCODE NetAppGetDefaultSettings (NETAPP\_SETTINGS \*pSettings)**  
Fetch Default Settings.
- **NETAPP\_RETCODE NetAppGetDefaultInitSettings (NETAPP\_INIT\_SETTINGS \*pSettings)**  
Fetch Default Initialization Settings.
- **NETAPP\_RETCODE NetAppOpen (NETAPP\_HANDLE \*tHandle, NETAPP\_OPEN\_SETTINGS \*pOpenSettings, NETAPP\_INIT\_SETTINGS \*pInitSettings, NETAPP\_SETTINGS \*pSettings)**  
Open the NetApp API.
- **NETAPP\_RETCODE NetAppSetSettings (NETAPP\_HANDLE tHandle, NETAPP\_SETTINGS tSettings)**  
Update NetApp settings NetApp with updated Settings.
- **NETAPP\_RETCODE NetAppGetSettings (NETAPP\_HANDLE tHandle, NETAPP\_SETTINGS \*pSettings)**  
Retrieve NetApp's current settings.
- **NETAPP\_RETCODE NetAppClose (NETAPP\_HANDLE tHandle)**  
Close the NetApp API.
- **NETAPP\_RETCODE NetAppSetNetworkSettings (NETAPP\_HANDLE tHandle, NETAPP\_IFACE tiface, NETAPP\_IP\_MODE tMode, NETAPP\_IP\_SETTINGS \*pSettings)**  
Change the network settings.
- **NETAPP\_RETCODE NetAppSetMacAddress (NETAPP\_HANDLE tHandle, NETAPP\_IFACE tiface, char \*pMacAddress)**  
Set the MAC address.
- **NETAPP\_RETCODE NetAppGetNetworkSettings (NETAPP\_HANDLE tHandle, NETAPP\_IFACE tiface, NETAPP\_IP\_SETTINGS \*pSettings)**  
Fetch the current network settings.
- **NETAPP\_RETCODE NetAppGetLinkState (NETAPP\_HANDLE tHandle, NETAPP\_IFACE tiface, NETAPP\_LINK\_STATE \*pLink)**  
Poll the link state from the kernel.
- **NETAPP\_RETCODE NetAppPing (NETAPP\_HANDLE tHandle, int32\_t lTimeoutMs, const char \*pcAddress)**  
Ping a network server.
- **NETAPP\_RETCODE NetAppDNSLookup (NETAPP\_HANDLE tHandle, const char \*pcHostname)**  
DNS Lookup.
- **NETAPP\_RETCODE NetAppNtpSetDate (NETAPP\_HANDLE tHandle, uint32\_t ulPeriodMs)**  
Set the Date/Time using NTPDate.
- **NETAPP\_RETCODE NetAppSetIfaceUp (NETAPP\_HANDLE tHandle, NETAPP\_IFACE tiface, bool bUp)**  
Set Interface Up or Down.

- **NETAPP\_RETCODE NetAppGetIfaceName** (NETAPP\_HANDLE tHandle, NETAPP\_IFACE tIface, char \*\*pString)  
Get Interface Name.
- **NETAPP\_RETCODE NetAppGetDefaultIface** (NETAPP\_HANDLE tHandle, NETAPP\_IFACE \*pIface)  
Get Default Interface.
- **char \* NetAppNtoa** (NETAPP\_IPV4\_ADDR ulAddress)  
Convert network notation to string.
- **NETAPP\_IPV4\_ADDR NetAppAton** (char \*pcString)  
Convert string to internet notation.
- **char \* NetAppHwAddrToA** (NETAPP\_HW\_ADDR tHwAddr, char \*pcString, uint32\_t ulLength)  
Convert a hardware address to a string.
- **NETAPP\_RETCODE NetAppAtoHwAddr** (char \*pcString, NETAPP\_HW\_ADDR tHwAddr)  
Convert string to hardware address.
- **NETAPP\_RETCODE NetAppHttpVoiceSearch** (NETAPP\_BT\_HID\_VOICE\_INFO \*pHidVoiceInfo, const char \*pLanguage)  
Asynchronous voice recognition search using NETAPP\_BT\_HID\_VOICE\_INFO.
- **NETAPP\_RETCODE NetAppGetIfaceInfo** (NETAPP\_HANDLE tHandle, NETAPP\_IFACE\_INFO \*\*pIfaceInfo, uint32\_t \*pListLength)  
Return the systems interface information.

## Function Documentation

### **NETAPP\_RETCODE NetAppAtoHwAddr** (char \* *pcString*, NETAPP\_HW\_ADDR *tHwAddr*)

Convert string to hardware address.

Function converts the String hardware address xx:xx:xx:xx:xx:xx into binary data.

#### Parameters:

in	<i>pcString</i>	Numbers-and-dots notation of IPv4 address
out	<i>tHwAddr</i>	Hardware Address

#### Returns:

NETAPP\_RETCODE

**NETAPP\_IPV4\_ADDR NetAppAtoN (char \* *pcString*)**

Convert string to internet notation.

Function converts the Internet host address *pcString* from the standard numbers-and-dots notation into binary data. This function wraps the IPv4 address manipulation function `inet_aton()`.

**Parameters:**

in        *pcString*                      Numbers-and-dots notation of IPV4 address

**Returns:**

NETAPP\_IPV4\_ADDR

**NETAPP\_RETCODE NetAppClose (NETAPP\_HANDLE *tHandle*)**

Close the NetApp API.

This function will close the NetApp API and unregister the callback.

**Parameters:**

in        *tHandle*                      NetApp handle

**Returns:**

NETAPP\_RETCODE

**See also:**

**NetAppOpen**

**NETAPP\_RETCODE NetAppDNSLookup (NETAPP\_HANDLE *tHandle*, const char \* *pcHostname*)**

DNS Lookup.

This function will kick off a background DNS request to lookup an IP address for the passed hostname. The results are fed back to the application in the form of a NETAPP\_CP\_DNSLOOKUP since NetApp can only make one asynchronous DNSLookup request at a time.

**Parameters:**

in        *tHandle*                      NetApp handle  
in        *pcHostname*                  Server name to lookup

**Returns:**

NETAPP\_RETCODE

**NETAPP\_RETCODE NetAppGetDefaultIface (NETAPP\_HANDLE *tHandle*, NETAPP\_IFACE \* *pIface*)**

Get Default Interface.

API will parse the routing table and determine what is the current default route which in essence can refer to the default interface (where all nonlocalized packets are sent).

**Parameters:**

in	<i>tHandle</i>	NetApp handle
out	<i>pIface</i>	Reference to a NETAPP_IFACE that will be set with the default interface.

**Returns:**

NETAPP\_RETCODE

**NETAPP\_RETCODE NetAppGetDefaultInitSettings (NETAPP\_INIT\_SETTINGS \* *pSettings*)**

Fetch Default Initialization Settings.

Fill the passed **NETAPP\_INIT\_SETTINGS** structure with the default values.

**Parameters:**

out	<i>pSettings</i>	- NetApp Settings structure
-----	------------------	-----------------------------

**Returns:**

NETAPP\_RETCODE

**See also:**

**NetAppClose**

**NETAPP\_RETCODE NetAppGetDefaultSettings (NETAPP\_SETTINGS \* *pSettings*)**

Fetch Default Settings.

Fill the passed **NETAPP\_SETTINGS** structure with the default settings

**Parameters:**

out	<i>pSettings</i>	- NetApp Settings structure
-----	------------------	-----------------------------

**Returns:**

NETAPP\_RETCODE

**See also:**

**NetAppClose**

**NETAPP\_RETCODE NetAppGetifaceInfo (NETAPP\_HANDLE *tHandle*, NETAPP\_IFACE\_INFO \*\* *plfaceInfo*, uint32\_t \* *pListLength*)**

Return the systems interface information.

**Parameters:**

in	<i>tHandle</i>	NetApp handle
in	<i>plfaceInfo</i>	Pointer to contain the list of interface information
out	<i>pListLength</i>	Length of the list (should always be NETAPP_IFACE_MAX)

**Remarks:**

The caller of this API must free the returned pointer list

**Returns:**

NETAPP\_RETCODE

**NETAPP\_RETCODE NetAppGetifaceName (NETAPP\_HANDLE *tHandle*, NETAPP\_IFACE *tlface*, char \*\* *pString*)**

Get Interface Name.

Return a strdup string for the interface name of the passed NETAPP\_IFACE enum.

**Remarks:**

This API will return NETAPP\_NOT\_SUPPORTED if the interface support was not compiled in.

YOU MUST FREE THE STRING RETURNED in *pString*.

**Parameters:**

in	<i>tHandle</i>	NetApp handle
in	<i>tlface</i>	Interface to fetch the name
out	<i>pString</i>	Reference of a pointer to an Interface name that YOU MUST FREE THIS STRING WHEN YOU ARE FINISHED

**Returns:**

NETAPP\_RETCODE

**NETAPP\_RETCODE NetAppGetLinkState (NETAPP\_HANDLE *tHandle*, NETAPP\_IFACE *iface*, NETAPP\_LINK\_STATE \* *pLink*)**

Poll the link state from the kernel.

The function does not block and will poll the kernel for the current link state for the passed interface. Generally the link state is disseminated to the application through the callback mechanism but this API is added in case the application wants to also poll the link state.

**Parameters:**

in	<i>tHandle</i>	NetApp handle
in	<i>iface</i>	The interface
out	<i>pLink</i>	Current link state

**Returns:**

NETAPP\_RETCODE

**NETAPP\_RETCODE NetAppGetNetworkSettings (NETAPP\_HANDLE *tHandle*, NETAPP\_IFACE *iface*, NETAPP\_IP\_SETTINGS \* *pSettings*)**

Fetch the current network settings.

This function retrieves the various network settings such as the MAC, IP, netmask, gateway, and DNS addresses. The MAC, IP, and netmask addresses are fetched using IOCTLs SIOCGIFHWADDR, SIOCGIFADDR, and SIOCGIFNETMASK respectively. The gateway address is fetched by using the AF\_NETLINK socket and sending the request to fetch the routing tables by RTM\_GETROUTE and then nlmsg\_flags = NLM\_F\_DUMP | NLM\_F\_REQUEST. In order to ensure that the get request does not interfere with the link change notification or other get requests (make this function thread safe), a separate AF\_NETLINK socket is used. Finally, the DNS servers are read from the resolv.conf file.

**Parameters:**

in	<i>tHandle</i>	NetApp handle
in	<i>iface</i>	The interface
out	<i>pSettings</i>	Settings structure to fill

**Returns:**

NETAPP\_RETCODE

**See also:**

**NetAppOpen NETAPP\_SETTINGS**

**NETAPP\_RETCODE NetAppGetSettings (NETAPP\_HANDLE *tHandle*, NETAPP\_SETTINGS \*  
*pSettings*)**

Retrieve NetApp's current settings.

**Parameters:**

in	<i>tHandle</i>	NetApp handle
out	<i>pSettings</i>	NetApp settings

**Returns:**

NETAPP\_RETCODE

**See also:**

**NetAppOpen**

**NetAppSetSettings**

**NETAPP\_RETCODE NetAppHttpVoiceSearch (NETAPP\_BT\_HID\_VOICE\_INFO \*  
*pHidVoiceInfo*, const char \* *pLanguage*)**

Asynchronous voice recognition search using **NETAPP\_BT\_HID\_VOICE\_INFO**.

Function takes in **NETAPP\_BT\_HID\_VOICE\_INFO** and first converts the PCM to FLAC to then perform a Google voice recognition query to convert the voice to a string. the result is passed in the callback **NETAPP\_CB\_VOICE\_REC\_DONE**

**Remarks:**

The passed pString MUST be freed after it is finished being used

**Parameters:**

in	<i>pHidVoiceInfo</i>	Bluetooth HID Voice info structure
in	<i>pLanguage</i>	Language string represented in ISO 639-1 Code

**See also:**

[http://en.wikipedia.org/wiki/List\\_of\\_ISO\\_639-1\\_codes](http://en.wikipedia.org/wiki/List_of_ISO_639-1_codes)

**Returns:**

NETAPP\_RETCODE



### **char\* NetAppHwAddrToA (NETAPP\_HW\_ADDR *tHwAddr*, char \* *pcString*, uint32\_t *ulLength*)**

Convert a hardware address to a string.

This function converts the standard hardware address (e.g., BSSID, MAC address, etc.) to a string.

#### **Remarks:**

You must pass a buffer to contain at least **NETAPP\_ENET\_LEN** +1 bytes.

#### **Parameters:**

in	<i>tHwAddr</i>	NETAPP_HW_ADDR to convert
in	<i>pcString</i>	Pointer to a buffer to store the string
in	<i>ulLength</i>	Length of the buffer <i>pcString</i>

#### **Returns:**

String representation of the hardware address.

### **char\* NetAppNtoA (NETAPP\_IPV4\_ADDR *ulAddress*)**

Convert network notation to string.

Function shall convert the Internet host address specified by *ulAddress* to a string in the Internet standard dot notation. This function wraps the IPv4 address manipulation function `inet_ntoa()`.

#### **Parameters:**

in	<i>ulAddress</i>	uint32_t representation of IPV4 address
----	------------------	---

#### **Returns:**

pointer to the Numbers-and-dots notation of IPV4 address

### **NETAPP\_RETCODE NetAppNtpSetDate (NETAPP\_HANDLE *tHandle*, uint32\_t *ulPeriodMs*)**

Set the Date/Time using NTPDate.

The function will kick off a background the NTPDate request reading the server list from `/etc/ntp/step-tickers`

When the background request is finished, the callback `NETAPP_CB_NTPDATE` is called passing the results if the `NtpSetDate` request was not canceled

#### **Remarks:**

Calling this API while another `NtpSetDate` request is in process will result in the first request being canceled and the second (new) request is made

#### **Parameters:**

in	<i>tHandle</i>	NetApp handler
in	<i>ulPeriodMs</i>	How often we want to update the date and time in the background in milliseconds. 0 means only update once

#### **Returns:**

NETAPP\_RETCODE

**NETAPP\_RETCODE NetAppOpen (NETAPP\_HANDLE \* *tHandle*, NETAPP\_OPEN\_SETTINGS \* *pOpenSettings*, NETAPP\_INIT\_SETTINGS \* *pInitSettings*, NETAPP\_SETTINGS \* *pSettings*)**

Open the NetApp API.

API is now multi entry and can be called by numerous applications passing more than one callback. Each callback is registered in a dynamic linklist so each callback will be called when a network event occurs. The first time this function is called, we create the NetApp handle and opens a DGRAM socket to get/set ip settings. If the application is run by NFS, the Wi-Fi Hotplug handler is called to initialize BWL APIs (if compiled with BWL support).

**Remarks:**

NetApp Settings structure can be NULL, if so the default settings will be used or the existing settings from a previous API init.

It is recommended to set the settings accordingly when opening NetApp the first time. The main settings structure is saved the first time NetApp is opened, upon subsequent calls to NetAppOpen only the callback information is saved.

**Parameters:**

out	<i>tHandle</i>	Returned handle to the NetApp API
in	<i>pOpenSettings</i>	Open settings to set callbacks. Can be NULL
in	<i>pInitSettings</i>	Initialization Settings called on the first call to this API. Must be NULL on subsequent calls to this API
in	<i>pSettings</i>	General configurable (on the fly) NetApp Settings. Can be NULL as well.

**Returns:**

NETAPP\_RETCODE

**See also:**

**NetAppClose**

**NetAppSetSettings**

**NetAppGetSettings**

**NETAPP\_RETCODE NetAppPing (NETAPP\_HANDLE *tHandle*, int32\_t *lTimeoutMs*, const char \* *pcAddress*)**

Ping a network server.

Asynchronous ping request using a NetAppSystem call. The system call will call the callback when either the ping returns successfully or times out.

**Remarks:**

Use caution with the timeout value of NETAPP\_WAIT\_FOREVER since this could result in this function blocking forever if the network is unreachable.

**Parameters:**

in	<i>tHandle</i>	NetApp handle
in	<i>lTimeoutMs</i>	Time to wait for a response
in	<i>pcAddress</i>	Server name to ping

**Returns:**

NETAPP\_RETCODE

**NETAPP\_RETCODE NetAppSetIfaceUp (NETAPP\_HANDLE *tHandle*, NETAPP\_IFACE *tlface*, bool *bUp*)**

Set Interface Up or Down.

NetApp automatically controls the interface state (up or down) for you so under normal circumstances you do not need to call this API.

NetApp wired link monitoring uses the Linux NetLink API to detect when the link goes up or down and in order to continue to receive these events from the Kernel when the cable is removed, we need to keep the interface up. This API can allow the application to completely disable the interface.

**Remarks:**

This API is not needed under normal circumstances and using it will disable NETLINK interface monitoring. USE CAUTION WHEN USING THIS API.

The API **NetAppSetNetworkSettings()** MUST be called with NETAPP\_IP\_MODE\_OFF before calling this API

**Parameters:**

in	<i>tHandle</i>	NetApp handle
in	<i>tlface</i>	Interface to bring up/down
in	<i>bUp</i>	TRUE for up, false for DOWN

**Returns:**

NETAPP\_RETCODE

**NETAPP\_RETCODE NetAppSetMacAddress (NETAPP\_HANDLE *tHandle*, NETAPP\_IFACE *tlface*, char \* *pMacAddress*)**

Set the MAC address.

Function to change the Hardware MAC address for the Specified interface.

**Parameters:**

in	<i>tHandle</i>	NetApp handle
in	<i>tlface</i>	The interface
in	<i>pMacAddress</i>	New MAC Address

**Returns:**

NETAPP\_RETCODE

**NETAPP\_RETCODE NetAppSetNetworkSettings (NETAPP\_HANDLE *tHandle*, NETAPP\_IFACE *tlface*, NETAPP\_IP\_MODE *tMode*, NETAPP\_IP\_SETTINGS \* *pSettings*)**

Change the network settings.

This function is responsible for applying the new network settings. Behavior is different for each network IP mode discussed below. If the network is configured in either NETAPP\_IP\_MODE\_OFF or NETAPP\_IP\_MODE\_STATIC we first check to see if the dhcpd daemon is running, if so we turn it off. Then we use the IOCTL SIOCGIFFLAGS and SIOCSIFFLAGS to turn on or off the interface depending again on the NETAPP\_IP\_MODE. the network is configured as static IP, the ip, netmask address are set using IOCTLs SIOCSIFADDR and SIOCSIFNETMASK to the AF\_PACKET interface. The gateway address is configured by adding a default routes in the routing table using again AF\_PACKET and the IOCTL SIOCADDRT. DNS servers are configured by reading and writing resolv.conf file in the root file system.

The Linux resolver is also re-initialized each time the network settings are applied.

**Parameters:**

in	<i>tHandle</i>	NetApp handle
in	<i>tlface</i>	The interface
in	<i>tMode</i>	IP Mode
in	<i>pSettings</i>	Settings to apply. This parameter can be NULL for any IP mode other than NETAPP_IP_MODE_STATIC.

**Returns:**

NETAPP\_RETCODE

**See also:**

**NetAppOpen NETAPP\_SETTINGS**

**NETAPP\_RETCODE NetAppSetSettings (NETAPP\_HANDLE *tHandle*, NETAPP\_SETTINGS *tSettings*)**

Update NetApp settings NetApp with updated Settings.

There are some settings of NetApp (like WPS 2.0 support) that can be enabled/ disabled on the fly. This method allows you to change these settings.

**Parameters:**

in	<i>tHandle</i>	NetApp handle
in	<i>tSettings</i>	NetApp settings structure

**Returns:**

NETAPP\_RETCODE

**See also:**

**NetAppOpen**

**NetAppGetSettings**

---

## Wi-Fi API

This includes APIs used to control and configure the wireless interface.

### Modules

- **Wi-Fi Invite**

APIs to support Wi-Fi Invite Feature.

- **Wi-Fi Direct**

This section describes Broadcom's P2P implementation of Wi-Fi Direct.

### Functions

- **NETAPP\_RETCODE NetAppWiFiStartScan** (NETAPP\_HANDLE tHandle, int32\_t lTickMs, int32\_t lScanTimeMs)  
Scan for Wireless Networks.
- **NETAPP\_RETCODE NetAppWiFiStopScan** (NETAPP\_HANDLE tHandle)  
Stop a Wireless Scan.
- **NETAPP\_RETCODE NetAppWiFiGetScanResults** (NETAPP\_HANDLE tHandle, NETAPP\_WIFI\_AP\_INFO \*\*pApInfoList, uint32\_t \*pulScanCount)  
Get Wi-Fi Scan Results.
- **NETAPP\_RETCODE NetAppWiFiConnectByPb** (NETAPP\_HANDLE tHandle)  
Wi-Fi Protected Setup Push Button.
- **NETAPP\_RETCODE NetAppWiFiConnectByPin** (NETAPP\_HANDLE tHandle, char \*pSsid, uint32\_t ulPin, bool bEnrollee)  
Start an Wi-Fi Auto Setup Configuration using a Pin.
- **NETAPP\_RETCODE NetAppWiFiGenerateWPSPin** (uint32\_t \*pulPin)  
Generate a WPS Pin.
- **NETAPP\_RETCODE NetAppWiFiConnect** (NETAPP\_HANDLE tHandle, NETAPP\_WIFI\_AP\_INFO \*pApInfo)  
Connect to a particular Access point.
- **NETAPP\_RETCODE NetAppWiFiDisconnect** (NETAPP\_HANDLE tHandle)  
Wi-Fi Disconnect.
- **NETAPP\_RETCODE NetAppWiFiGetConnectedApInfo** (NETAPP\_HANDLE tHandle, NETAPP\_WIFI\_AP\_INFO \*pApInfo)  
Return current Connected Access Point.
- **NETAPP\_RETCODE NetAppWiFiGetScannedApInfo** (NETAPP\_HANDLE tHandle, NETAPP\_WIFI\_AP\_INFO \*pApInfo)  
Return Access Point Info for a Scanned AP.
- **NETAPP\_RETCODE NetAppWiFiIsConnected** (NETAPP\_HANDLE tHandle, bool \*pIsConnected)  
Check if Wi-Fi is connected or not.

- **NETAPP\_RETCODE NetAppWiFisEnabled** (NETAPP\_HANDLE tHandle, bool \*pIsEnabled)  
Check if Wi-Fi is enabled.
- **NETAPP\_RETCODE NetAppWiFiGetApInfo** (NETAPP\_HANDLE tHandle, char \*pSSID)  
Fetch **NETAPP\_WIFI\_AP\_INFO** for a particular SSID.

## Function Documentation

### **NETAPP\_RETCODE NetAppWiFiConnect** (NETAPP\_HANDLE *tHandle*, NETAPP\_WIFI\_AP\_INFO \* *pApInfo*)

Connect to a particular Access point.

#### Parameters:

in	<i>tHandle</i>	NetApp handle
in	<i>pApInfo</i>	The access point information structure.

#### Remarks:

All that is needed to connect to an AP is the SSID and a password; the security settings will be automatically detected if set to NETAPP\_WIFI\_SECURITY\_AUTO\_DETECT.

#### Returns:

NETAPP\_RETCODE

#### See also:

NETAPP\_WIFI\_AP\_INFO

### **NETAPP\_RETCODE NetAppWiFiConnectByPb** (NETAPP\_HANDLE *tHandle*)

Wi-Fi Protected Setup Push Button.

The auto configuration will take place in the background with the router and the result of the auto configuration will be sent to the application through the registered callback function tCallback.

#### Remarks:

WPS will run with the WPS mode specified from the **NETAPP\_SETTINGS** structure passed to NetAppSetSettings or the first call to NetAppOpen. To change to change WPS mode (2.0 vs 1.0), call NetAppSetSettings again.

#### Parameters:

in	<i>tHandle</i>	NetApp handle
----	----------------	---------------

#### Returns:

NETAPP\_RETCODE

**NETAPP\_RETCODE NetAppWiFiConnectByPin (NETAPP\_HANDLE *tHandle*, char \* *pSsid*, uint32\_t *ulPin*, bool *bEnrollee*)**

Start an Wi-Fi Auto Setup Configuration using a Pin.

The auto configuration will take place in the background with the router and the result of the auto configuration will be sent to the application through the registered callback function *tCallback*.

**Remarks:**

*bEnrollee*(true) implements Section 5.1, "Add to AP as an Enrollee" (more precisely, sections 5.1.1, 5.1.3, 5.1.4, and 5.1.5) of the WPS Test Plan ver 1.10 ([Reference \[3\] on page 7](#)).

*bEnrollee*(false) implements Section 5.1, "Act as Registrar and Configure AP" also from the WPS Test Plan ver 1.10 ([Reference \[3\] on page 7](#)).

It is no longer necessary to specify the SSID of the router to perform WPS Pin with (as an Enrollee only) as NetApp will automatically scan for APs that have opened up a WPS window if the SSID is not specified. If you do not specify an SSID you MUST start WPS on the AP first before calling this API.

WPS will run with the WPS mode specified from the **NETAPP\_SETTINGS** structure passed to *NetAppSetSettings* or the first call to *NetAppOpen*. Call *NetAppSetSettings* again to change WPS mode (2.0 vs 1.0)

**Parameters:**

in	<i>tHandle</i>	NetApp handle
in	<i>pSsid</i>	The name of the router connect too (or null to do a search for APs).
in	<i>ulPin</i>	Autoconfiguration pin code
in	<i>bEnrollee</i>	This device will be the enrollee; otherwise, NetApp is the registrar.

**Returns:**

NETAPP\_RETCODE

NETAPP\_INVALID\_PIN if the *ulPin* is not a valid WPS pin

**NETAPP\_RETCODE NetAppWiFiDisconnect (NETAPP\_HANDLE *tHandle*)**

Wi-Fi Disconnect.

Disconnect/disassociate from the current connected access point (if any) and stops any ongoing connection attempt.

**Parameters:**

in	<i>tHandle</i>	NetApp handle
----	----------------	---------------

**Returns:**

NETAPP\_RETCODE



**NETAPP\_RETCODE NetAppWiFiGenerateWSPin (uint32\_t \* *pulPin*)**

Generate a WPS Pin.

API will generate a WPS PIN that meets Section 6.4.1 of the *Wi-Fi Protected Setup Specifications 1.0h* ([Reference \[4\] on page 7](#)).

**Parameters:**

out      *pulPin*                      The autogenerated WPS pin.

**Returns:**

NETAPP\_RETCODE

**NETAPP\_RETCODE NetAppWiFiGetApInfo (NETAPP\_HANDLE *tHandle*, char \* *pSSID*)**

Fetch **NETAPP\_WIFI\_AP\_INFO** for a particular SSID.

This asynchronous API will fetch in the background the full AP info (minus the password) of an access point with the SSID *pSSID*. The applications of this is to fetch all the credentials (minus the password) of a hidden access point to be able to do a manual configuration to that hidden AP without having to prompt the user for the security type.

When NetApp has finished attempting to fetch the **NETAPP\_WIFI\_AP\_INFO** for the AP, the status notification

**See also:**

**NETAPP\_CB\_FETCHED\_APINFO** is called passing the **NETAPP\_WIFI\_AP\_INFO** structure and the **NETAPP\_RETCODE** that can be either:

**NETAPP\_SUCCESS**: Successfully fetched the AP info.

**NETAPP\_FAILURE**: Failure in **NetApp API Overview** to fetch the AP Info.

**NETAPP\_TIMEOUT**: Timed out trying to connect to the AP

**Remarks:**

If the **NETAPP\_TIMEOUT** return code is sent to the app with the **NETAPP\_CB\_FETCHED\_APINFO** then it is possible that the SSID is misspelled or the AP is not found.

Calling this function will disconnect the interface from a current connected access point (if any) and stop any active scan.

**Parameters:**

in      *tHandle*                      NetApp handle  
in      *pSSID*                      Null terminated SSID of the access point

**Returns:**

NETAPP\_RETCODE

**NETAPP\_RETCODE NetAppWiFiGetConnectedApInfo (NETAPP\_HANDLE *tHandle*, NETAPP\_WIFI\_AP\_INFO \* *pApInfo*)**

Return current Connected Access Point.

Function will return the full **NETAPP\_WIFI\_AP\_INFO** structure for the current connected AP (if we are connected)

**Parameters:**

in	<i>tHandle</i>	NetApp handle
out	<i>pApInfo</i>	the AP Info structure filled for the connected AP

**Returns:**

NETAPP\_RETCODE

**NETAPP\_RETCODE NetAppWiFiGetScannedApInfo (NETAPP\_HANDLE *tHandle*, NETAPP\_WIFI\_AP\_INFO \* *pApInfo*)**

Return Access Point Info for a Scanned AP.

This function will return the full **NETAPP\_WIFI\_AP\_INFO** structure for a scanned AP where the name is set in cSSID of APInfo structure.

**Parameters:**

in	<i>tHandle</i>	NetApp handle
out	<i>pApInfo</i>	the AP Info structure filled for the connected AP

**Returns:**

NETAPP\_RETCODE

**NETAPP\_RETCODE NetAppWiFiGetScanResults (NETAPP\_HANDLE *tHandle*, NETAPP\_WIFI\_AP\_INFO \*\* *pApInfoList*, uint32\_t \* *pulScanCount*)**

Get Wi-Fi Scan Results.

After the interface has notified the application that scan results are available, the application can call this function to fetch the scan results. This function will create a dynamic array of **NETAPP\_WIFI\_AP\_INFO** structures one for each detected AP.

**Remarks:**

CALLERS OF THIS FUNCTION MUST FREE THE POINTER *pApInfoList*

**Parameters:**

in	<i>tHandle</i>	NetApp handle
out	<i>pApInfoList</i>	Pointer to the list of APs scanned.
out	<i>pulScanCount</i>	The number of scanned APs.

**Returns:**

NETAPP\_RETCODE

**NETAPP\_RETCODE NetAppWiFisConnected (NETAPP\_HANDLE *tHandle*, bool \* *plsConnected*)**

Check if Wi-Fi is connected or not.

**Parameters:**

in	<i>tHandle</i>	NetApp handle
out	<i>plsConnected</i>	True if connected, otherwise false

**Returns:**

NETAPP\_RETCODE

**NETAPP\_RETCODE NetAppWiFisEnabled (NETAPP\_HANDLE *tHandle*, bool \* *plsEnabled*)**

Check if Wi-Fi is enabled.

NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE

THIS API IS BEING DEPRECIATED AND REPLACED BY NetAppIsEnabled()

**Parameters:**

in	<i>tHandle</i>	NetApp handle
out	<i>plsEnabled</i>	True if enabled, otherwise false

**Returns:**

NETAPP\_RETCODE

**NETAPP\_RETCODE NetAppWiFiStartScan (NETAPP\_HANDLE *tHandle*, int32\_t *lTickMs*, int32\_t *lScanTimeMs*)**

Scan for Wireless Networks.

This API will start the Wi-Fi network scan that will return results every *ulTickMs*.

**Parameters:**

in	<i>tHandle</i>	NetApp handle
in	<i>lTickMs</i>	How often the scan and results should be done and sent to the application.
in	<i>lScanTimeMs</i>	How long to spent scanning each channel in ms.

**Remarks:**

Default *ScanTimeMs* is 40 when connected to an AP and 80 when not connected to an AP. It is recommended that you choose a value greater than 100 ms to pick up more APs, however, note that the larger the number, the longer a scan will take.

Setting *lTickMs* to 0 will result in only one scan

**Returns:**

NETAPP\_RETCODE

**NETAPP\_RETCODE NetAppWiFiStopScan (NETAPP\_HANDLE *tHandle*)**

Stop a Wireless Scan.

**Parameters:**

in        *tHandle*                    NetApp handle

**Returns:**

NETAPP\_RETCODE

---

## Wi-Fi Invite

APIs to support Wi-Fi Invite Feature.

### Functions

- **NETAPP\_RETCODE NetAppWiFilInviteStart (NETAPP\_HANDLE tHandle)**  
Start Wi-Fi Invite.
- **NETAPP\_RETCODE NetAppWiFilInviteStop (NETAPP\_HANDLE tHandle)**  
Stop Wi-Fi Invite.
- **NETAPP\_RETCODE NetAppWiFilInviteAccept (NETAPP\_HANDLE tHandle, char \*pBSSID)**  
Accept a Wi-Fi Invite Request.
- **NETAPP\_RETCODE NetAppWiFilInviteReject (NETAPP\_HANDLE tHandle, char \*pBSSID)**  
Reject a Wi-Fi Invite Request.

### Function Documentation

#### **NETAPP\_RETCODE NetAppWiFilInviteAccept (NETAPP\_HANDLE *tHandle*, char \* *pBSSID*)**

Accept a Wi-Fi Invite Request.

Kick off an asynchronous Wi-Fi Invite accept request that will fetch the access point credentials using WPS and once the credentials are obtained NetApp will connect to the access point and call the NETAPP\_CB\_CONNECT callback when finished.

#### **Remarks:**

Calling this function will free the invite context that NetApp was saving for this AP

WPS will run with the WPS mode specified from the **NETAPP\_SETTINGS** structure passed to NetAppSetSettings or the first call to NetAppOpen. To change WPS mode (2.0 vs 1.0), call NetAppSetSettings again.

#### **Parameters:**

in	<i>tHandle</i>	NetApp handle
in	<i>pBSSID</i>	The BSSID of the AP we want to accept or reject.

#### **Returns:**

NETAPP\_RETCODE

#### **See also:**

**NetAppWiFilInviteReject**

**NETAPP\_RETCODE NetAppWiFilInviteReject (NETAPP\_HANDLE *tHandle*, char \* *pBSSID*)**

Reject a Wi-Fi Invite Request.

Send an asynchronous Wi-Fi Invite reject notification to the access point so that we no longer receive invites from this AP.

**Remarks:**

Calling this function will free the invite context that NetApp was saving for this AP.

WPS will run with the WPS mode specified from the **NETAPP\_SETTINGS** structure passed to NetAppSetSettings or the first call to NetAppOpen. Please call NetAppSetSettings again to change WPS mode (2.0 vs 1.0).

**Parameters:**

in	<i>tHandle</i>	NetApp handle
in	<i>pBSSID</i>	The BSSID of the access point to reject

**Returns:**

NETAPP\_RETCODE

**See also:**

**NetAppWiFilInviteAccept**

**NETAPP\_RETCODE NetAppWiFilInviteStart (NETAPP\_HANDLE *tHandle*)**

Start Wi-Fi Invite.

Start the Wi-Fi Invite feature which will kick off a prob request to notify all Wi-Fi enabled routers that the client device is Wi-Fi Invite capable.

**Remarks:**

This API should only be called when the AP is not connected and where the application is not trying to reconnect to an AP.

**Parameters:**

in	<i>tHandle</i>	NetApp handle
----	----------------	---------------

**Returns:**

NETAPP\_RETCODE Will return a failure if the AP is currently connected to an access point.

**NETAPP\_RETCODE NetAppWiFilInviteStop (NETAPP\_HANDLE *tHandle*)**

Stop Wi-Fi Invite.

**Parameters:**

in        *tHandle*                      NetApp handle

**Returns:**

NETAPP\_RETCODE

## Wi-Fi Direct

This section describes Broadcom's P2P implementation of Wi-Fi Direct.

### Functions

- **NETAPP\_RETCODE NetAppWiFiP2PDiscover (NETAPP\_HANDLE tHandle, NETAPP\_P2P\_DISCOVER\_PARAMS \*pParams)**  
Start Wi-Fi Direct Discovery.
- **NETAPP\_RETCODE NetAppWiFiP2PStopDiscovery (NETAPP\_HANDLE tHandle)**  
Stop Wi-Fi Direct Discovery.
- **NETAPP\_RETCODE NetAppWiFiP2PConnect (NETAPP\_HANDLE tHandle, char \*pName, uint32\_t ulTimeoutSec)**  
Start Wi-Fi Direct Connection.
- **NETAPP\_RETCODE NetAppWiFiP2PDisconnect (NETAPP\_HANDLE tHandle)**  
Stop P2P Connection attempt and disconnect.
- **NETAPP\_RETCODE NetAppWiFiP2PGetSSID (NETAPP\_HANDLE tHandle, char \*pBuf, uint32\_t ulBufSize)**

### Function Documentation

#### **NETAPP\_RETCODE NetAppWiFiP2PConnect (NETAPP\_HANDLE *tHandle*, char \* *pName*, uint32\_t *ulTimeoutSec*)**

Start Wi-Fi Direct Connection.

##### Parameters:

in	<i>tHandle</i>	NetApp handle
in	<i>pName</i>	the P2P Device Name
in	<i>ulTimeoutSec</i>	How long to wait for P2P connect before we timeout

##### Returns:

NETAPP\_RETCODE

#### **NETAPP\_RETCODE NetAppWiFiP2PDisconnect (NETAPP\_HANDLE *tHandle*)**

Stop P2P Connection attempt and disconnect.

##### Parameters:

in	<i>tHandle</i>	NetApp handle
----	----------------	---------------

##### Returns:

NETAPP\_RETCODE



**NETAPP\_RETCODE NetAppWiFiP2PDiscover (NETAPP\_HANDLE *tHandle*, NETAPP\_P2P\_DISCOVER\_PARAMS \* *pParams*)**

Start Wi-Fi Direct Discovery.

This API will start a Wi-Fi Direct discovery to find all P2P capable devices. As part of the discovery, the device will be put in the Listen Mode, scan and find phases as defined in Wi-Fi Peer-to-Peer (P2P) Technical Specification Draft Version 1.15.

**Parameters:**

in	<i>tHandle</i>	NetApp handle
in	<i>pParams</i>	Discovery Parameter

**See also:**

**NETAPP\_P2P\_DISCOVER\_PARAMS**

**Returns:**

NETAPP\_RETCODE

**See also:**

NetAppWiFi P2PStopDiscovery()

**NETAPP\_RETCODE NetAppWiFiP2PGetSSID (NETAPP\_HANDLE *tHandle*, char \* *pBuf*, uint32\_t *ulBufSize*)**

Retrieve the group owner SSID.

**Parameters:**

in	<i>tHandle</i>	NetApp handle
out	<i>pBuf</i>	Pointer to a string buffer to store the SSID
in	<i>ulBufSize</i>	Buffer size

**Returns:**

NETAPP\_RETCODE

**NETAPP\_RETCODE NetAppWiFiP2PStopDiscovery (NETAPP\_HANDLE *tHandle*)**

Stop Wi-Fi Direct Discovery.

Stop Wi-Fi Direct Discovery and automatically re-start Wi-Fi Invite if we are not associated.

**Parameters:**

in	<i>tHandle</i>	NetApp handle
----	----------------	---------------

**Returns:**

NETAPP\_RETCODE

**See also:**

**NetAppWiFiP2PDiscover()**

---

## Zeroconf (Bonjour)

Zero Configuration (Bonjour) library to support Multicast-DNS and DNS-Service Discovery.

### Functions

- **NETAPP\_RETCODE NetAppZeroConfPublish** (NETAPP\_HANDLE tHandle, char \*pName, char \*pType, uint32\_t ulPort, char \*pTxtRecord, uint32\_t ulTxtLength)  
Publish a Service using Bonjour/Zeroconf.
- **NETAPP\_RETCODE NetAppZeroConfBrowse** (NETAPP\_HANDLE tHandle, char \*pType)  
Browse for Zeroconfig/Bonjour services.
- **NETAPP\_RETCODE NetAppZeroConfGetBrowseResults** (NETAPP\_HANDLE tHandle, char \*pName, NETAPP\_ZEROCONF\_SERVICE\_INFO \*pInfo)  
Get Browsed service results.

### Function Documentation

#### **NETAPP\_RETCODE NetAppZeroConfBrowse** (NETAPP\_HANDLE *tHandle*, char \* *pType*)

Browse for Zeroconfig/Bonjour services.

Initiate a single service discovery. When a service is found, NetApp will call the callback NETAPP\_CB\_FOUNDED\_SERVICE and the application should then call NetAppZeroConfGetServices() to return a list of discovered services and the TXT record for each discovered service

#### Remarks:

Currently we can only browse for one service at a time, this can change when there are more services supported.

#### Parameters:

in	<i>tHandle</i>	NetApp handle
in	<i>pType</i>	Service type to browse for (e.g., _http._tcp)

#### Returns:

NETAPP\_RETCODE

#### See also:

**NetAppZeroConfGetBrowseResults**

#### **NETAPP\_RETCODE NetAppZeroConfGetBrowseResults** (NETAPP\_HANDLE *tHandle*, char \* *pName*, NETAPP\_ZEROCONF\_SERVICE\_INFO \* *pInfo*)

Get Browsed service results.

Fetch a reference to the NETAPP\_ZEROCONF\_SERVICE\_INFO for the passed service name. The reference to pInfo will be available until the service is removed at which point NetApp will free the handle.

**Remarks:**

DO NOT free the `plInfo` structure, garbage collection of the service information is taken care of inside of NetApp. The reference to `NETAPP_ZEROCONF_SERVICE_INFO` will be available for the whole life of the service until when the service is removed. NetApp will free the `NETAPP_ZEROCONF_SERVICE_INFO` after the callback `NETAPP_CB_ZEROCONF_SERVICE` is called.

**Parameters:**

in	<i>tHandle</i>	NetApp handle
in	<i>pName</i>	Service name that we want to fetch the info for
out	<i>plInfo</i>	Pointer to the <code>SERVICE_INFO</code> structure that is cached inside of NetApp

**Returns:**

`NETAPP_RETCODE`

**NETAPP\_RETCODE NetAppZeroConfPublish (NETAPP\_HANDLE *tHandle*, char \* *pName*, char \* *pType*, uint32\_t *ulPort*, char \* *pTxtRecord*, uint32\_t *ulTxtLength*)**

Publish a Service using Bonjour/Zeroconf.

Using DNS Service Discovery portion of Zero Configuration Networking, this API will publish a service that the device will support.

NetApp will add in the following TXT records automatically taken from the `NETAPP_SETTINGS` structure passed to **NetAppOpen()**:

- manufacturer=*pManufacturer*
- model\_name=*pModelName*
- model\_number=*pModelNumber*
- serial\_number=*pSerialNumber*

**Remarks:**

the value of `NETAPP_SETTINGS.pDeviceName` is used for the `HostName` where any spaces are converted to underscores.

**Parameters:**

in	<i>tHandle</i>	NetApp handle
in	<i>pName</i>	The Service Name
in	<i>pType</i>	The Service Type (e.g., <code>_http._tcp</code> )
in	<i>ulPort</i>	The Port for the service
in	<i>pTxtRecord</i>	Pointer to a buffer containing the TXT record (if any)
in	<i>ulTxtLength</i>	Length of the TXT record

**Returns:**

`NETAPP_RETCODE`

---

# Bluetooth

Bluetooth library to support various Bluetooth profiles (HID, AV, etc.)

## Functions

- **NETAPP\_RETCODE NetAppBluetoothDiscovery (NETAPP\_HANDLE tHandle, uint32\_t tServices)**  
Bluetooth Asynchronous Discovery.
- **NETAPP\_RETCODE NetAppBluetoothGetDiscoveryResults (NETAPP\_HANDLE tHandle, NETAPP\_BT\_DEV\_INFO \*\*pBtDevInfo, uint32\_t \*pulCount)**  
Get Discovery Results.
- **NETAPP\_RETCODE NetAppBluetoothConnect (NETAPP\_HANDLE tHandle, NETAPP\_BT\_DEV\_INFO \*pBtDevInfo)**  
Connect to a Bluetooth Device.
- **NETAPP\_RETCODE NetAppBluetoothDisconnect (NETAPP\_HANDLE tHandle, NETAPP\_BT\_DEV\_INFO \*pBtDevInfo)**  
Disconnect the Bluetooth Device.
- **NETAPP\_RETCODE NetAppBluetoothSendAudioBuffer (NETAPP\_HANDLE tHandle, void \*pBuf, uint32\_t ulLength)**  
Send Audio Buffer to A2DP stream.
- **NETAPP\_RETCODE NetAppBluetoothAvStart (NETAPP\_HANDLE tHandle, bool bSynchronous, NETAPP\_BT\_AUDIO\_FORMAT \*pBtAudioFormat)**  
Start AV (Audio Source) streaming to Bluetooth headset.
- **NETAPP\_RETCODE NetAppBluetoothAvStop (NETAPP\_HANDLE tHandle)**  
Stop AV (Audio Source) streaming to Bluetooth headset.
- **NETAPP\_RETCODE NetAppBluetoothAvkStart (NETAPP\_HANDLE tHandle, NETAPP\_BT\_AUDIO\_FORMAT \*pBtAudioFormat)**  
Start AVK (Audio Sink) streaming from a Bluetooth device.
- **NETAPP\_RETCODE NetAppBluetoothAvkStop (NETAPP\_HANDLE tHandle)**  
Stop AVK (Audio Sink) streaming from a Bluetooth device.
- **NETAPP\_RETCODE NetAppBluetoothSimplePairingAck (NETAPP\_HANDLE tHandle, bool bAccept, NETAPP\_BT\_DEV\_INFO \*pDevInfo)**  
Accept or Reject a Simple Pairing Request.

## Function Documentation

### **NETAPP\_RETCODE NetAppBluetoothAvkStart (NETAPP\_HANDLE *tHandle*, NETAPP\_BT\_AUDIO\_FORMAT \* *pBtAudioFormat*)**

Start AVK (Audio Sink) streaming from a Bluetooth device.

#### **Parameters:**

in	<i>tHandle</i>	NetApp handle
in	<i>pBtAudioFormat</i>	Pointer to BT audio format parameters

#### **Returns:**

NETAPP\_RETCODE

### **NETAPP\_RETCODE NetAppBluetoothAvkStop (NETAPP\_HANDLE *tHandle*)**

Stop AVK (Audio Sink) streaming from a Bluetooth device.

#### **Parameters:**

in	<i>tHandle</i>	NetApp handle
----	----------------	---------------

#### **Returns:**

NETAPP\_RETCODE

### **NETAPP\_RETCODE NetAppBluetoothAvStart (NETAPP\_HANDLE *tHandle*, bool *bSynchronous*, NETAPP\_BT\_AUDIO\_FORMAT \* *pBtAudioFormat*)**

Start AV (Audio Source) streaming to Bluetooth headset.

#### **Parameters:**

in	<i>tHandle</i>	NetApp handle
in	<i>bSynchronous</i>	We will be feeding AV synchronously (ex. Nexus audio capture) or asynchronously (readying from a file).
in	<i>pBtAudioFormat</i>	Pointer to BT audio format parameters

#### **Returns:**

NETAPP\_RETCODE

### **NETAPP\_RETCODE NetAppBluetoothAvStop (NETAPP\_HANDLE *tHandle*)**

Stop AV (Audio Source) streaming to Bluetooth headset.

#### **Parameters:**

in	<i>tHandle</i>	NetApp handle
----	----------------	---------------

#### **Returns:**

NETAPP\_RETCODE

**NETAPP\_RETCODE NetAppBluetoothConnect (NETAPP\_HANDLE *tHandle*,  
NETAPP\_BT\_DEV\_INFO \* *pBtDevInfo*)**

Connect to a Bluetooth Device.

Initiate a pairing session with a discovered Bluetooth device. This API will do the correct pairing/bonding process depending on the service type the Bluetooth device is.

**Parameters:**

in	<i>tHandle</i>	NetApp handle.
in	<i>pBtDevInfo</i>	Pointer to an array of discovered Bluetooth devices.

**Returns:**

NETAPP\_RETCODE

**NETAPP\_RETCODE NetAppBluetoothDisconnect (NETAPP\_HANDLE *tHandle*,  
NETAPP\_BT\_DEV\_INFO \* *pBtDevInfo*)**

Disconnect the Bluetooth Device.

**Parameters:**

in	<i>tHandle</i>	NetApp handle.
in	<i>pBtDevInfo</i>	Pointer to an array of discovered Bluetooth devices.

**Returns:**

NETAPP\_RETCODE

**NETAPP\_RETCODE NetAppBluetoothDiscovery (NETAPP\_HANDLE *tHandle*, uint32\_t  
*tServices*)**

Bluetooth Asynchronous Discovery.

Kick off a background discovery request to find Bluetooth devices by a particular service type or all services. Once a device is found the callback NETAPP\_CB\_BT\_DISCOVERY\_RESULTS is called with the Bluetooth discovery is completed.

**Parameters:**

in	<i>tHandle</i>	NetApp handle
in	<i>tServices</i>	Service type to search for or all services

**See also:**

NETAPP\_BT\_SERVICE\_TYPE

**Returns:**

NETAPP\_RETCODE

**NETAPP\_RETCODE NetAppBluetoothGetDiscoveryResults (NETAPP\_HANDLE *tHandle*, NETAPP\_BT\_DEV\_INFO \*\* *pBtDevInfo*, uint32\_t \* *pulCount*)**

Get Discovery Results.

Return a pointer to a newly allocated array of discovered Bluetooth devices.

**Remarks:**

USER MUST FREE THE RETURNED ARRAY ONCE YOU ARE FINISH WITH IT!

**Parameters:**

in	<i>tHandle</i>	NetApp handle,
out	<i>pBtDevInfo</i>	Pointer to an array of discovered Bluetooth devices.
out	<i>pulCount</i>	The number of discovered devices.

**Returns:**

NETAPP\_RETCODE

**NETAPP\_RETCODE NetAppBluetoothSendAudioBuffer (NETAPP\_HANDLE *tHandle*, void \* *pBuf*, uint32\_t *ulLength*)**

Send Audio Buffer to A2DP stream.

**Parameters:**

in	<i>tHandle</i>	NetApp handle
in	<i>pBuf</i>	Pointer to buffer containing audio data
in	<i>ulLength</i>	Number of bytes sent

**Returns:**

NETAPP\_RETCODE

**NETAPP\_RETCODE NetAppBluetoothSimplePairingAck (NETAPP\_HANDLE *tHandle*, bool *bAccept*, NETAPP\_BT\_DEV\_INFO \* *pDevInfo*)**

Accept or Reject a Simple Pairing Request.

This API should be called after receiving the NETAPP\_CB\_BT\_SIMPLE\_PAIRING callback from NetApp to accept or reject a simple pairing request.

**Remarks:**

If the `bAutoPair` variable set in `BT_SETTINGS` structure then NetApp will automatically accept simple pairing requests.

**Parameters:**

in	<i>tHandle</i>	NetApp handle
in	<i>bAccept</i>	True to accept, false to reject
in	<i>pDevInfo</i>	Pointer to Bluetooth device information

**Returns:**

NETAPP\_RETCODE



## Database APIs

API to fetch information from the built-in database back end.

### Functions

- **NETAPP\_RETCODE NetAppWiFiDeleteSavedApInfo** (NETAPP\_HANDLE tHandle, NETAPP\_WIFI\_AP\_INFO \*pApInfo)  
Delete a Saved Access point.
- **NETAPP\_RETCODE NetAppWiFiGetSavedApInfoList** (NETAPP\_HANDLE tHandle, NETAPP\_WIFI\_AP\_INFO \*\*pApInfoList, uint32\_t \*pulCount)  
Fetch saved Access Point List.
- **NETAPP\_RETCODE NetAppBluetoothDeleteSavedDevInfo** (NETAPP\_HANDLE tHandle, NETAPP\_BT\_DEV\_INFO \*pDevInfo)  
Delete a Saved Bluetooth Device info from the database.
- **NETAPP\_RETCODE NetAppBluetoothGetSavedBtDevList** (NETAPP\_HANDLE tHandle, NETAPP\_BT\_DEV\_INFO \*\*pDevInfoList, uint32\_t \*pulCount)  
Fetch saved Bluetooth pre-paired list.

### Function Documentation

#### **NETAPP\_RETCODE NetAppBluetoothDeleteSavedDevInfo** (NETAPP\_HANDLE *tHandle*, NETAPP\_BT\_DEV\_INFO \* *pDevInfo*)

Delete a Saved Bluetooth Device info from the database.

Removed the saved information from the database back end.

##### Parameters:

in	<i>tHandle</i>	NetApp handle
in	<i>pDevInfo</i>	Bluetooth Device Info (only the tAddr value is used in this structure to lookup the hardware address to delete from the database back end.

##### Returns:

NETAPP\_RETCODE

**NETAPP\_RETCODE NetAppBluetoothGetSavedBtDevList (NETAPP\_HANDLE *tHandle*,  
NETAPP\_BT\_DEV\_INFO \*\* *pDevInfoList*, uint32\_t \* *pulCount*)**

Fetch saved Bluetooth pre-paired list.

**Remarks:**

This function returns a copy of the Bluetooth device list. You must free the list once you are finished with it.

**Parameters:**

in	<i>tHandle</i>	NetApp handle
out	<i>pDevInfoList</i>	Saved Bluetooth device list
out	<i>pulCount</i>	Number of saved AP

**Returns:**

NETAPP\_RETCODE

**NETAPP\_RETCODE NetAppWiFiDeleteSavedApInfo (NETAPP\_HANDLE *tHandle*,  
NETAPP\_WIFI\_AP\_INFO \* *pApInfo*)**

Delete a Saved Access point.

Removed the access point from the database back end.

**Parameters:**

in	<i>tHandle</i>	NetApp handle
in	<i>pApInfo</i>	the AP to delete

**Returns:**

NETAPP\_RETCODE

**NETAPP\_RETCODE NetAppWiFiGetSavedApInfoList (NETAPP\_HANDLE *tHandle*,  
NETAPP\_WIFI\_AP\_INFO \*\* *pApInfoList*, uint32\_t \* *pulCount*)**

Fetch saved Access Point List.

**Remarks:**

This function returns a copy of the saved list. You must free the list once you are finished with it.

**Parameters:**

in	<i>tHandle</i>	NetApp handle
out	<i>pApInfoList</i>	Saved AP list
out	<i>pulCount</i>	Number of saved AP

**Returns:**

NETAPP\_RETCODE

## Section 5: Data Structure Documentation

### NETAPP\_BT\_AUDIO\_FORMAT Struct Reference

Bluetooth Audio Format Information.

```
#include <netapp.h>
```

#### Data Fields

- **NETAPP\_BT\_AV\_MODE tMode**  
Mode (Number of channels)
- **uint32\_t ulSampleRate**  
Sample Rate.
- **uint16\_t ucBitsPerSample**  
Bits per sample.

#### Field Documentation

**NETAPP\_BT\_AV\_MODE NETAPP\_BT\_AUDIO\_FORMAT::tMode**

Mode (Number of channels)

**uint16\_t NETAPP\_BT\_AUDIO\_FORMAT::ucBitsPerSample**

Bits per sample.

**uint32\_t NETAPP\_BT\_AUDIO\_FORMAT::ulSampleRate**

Sample Rate.

The documentation for this struct was generated from netapp.h.

---

## NETAPP\_BT\_DEV\_INFO Struct Reference

Bluetooth Device Information.

```
#include <netapp.h>
```

### Data Fields

- char **cAddr** [NETAPP\_ENET\_LEN+1]
- char **cName** [NETAPP\_BT\_NAME\_LEN+1]  
Hardware Address in xx:xx:xx:xx:xx notation.
- int32\_t **IRssi**  
Receiver Signal Strength Indicator.
- uint32\_t **ulServiceMask**  
Discovered services.
- uint8\_t **usLinkKey** [NETAPP\_LINK\_KEY\_LEN]  
Link key obtained from connection.
- bool **bHasLinkKey**  
There is a link key present.
- **NETAPP\_BT\_HID\_INFO** **tHidInfo**  
HID descriptor information.
- uint8\_t **ucMajorClassDev**  
Major class of device (see spec)
- uint8\_t **ucMinorClassDev**  
Minor class of device (see spec)
- uint16\_t **usServiceClassDev**  
Service class of device (see spec)
- uint16\_t **usProductID**  
Product ID.
- uint16\_t **usVendorID**  
Vendor ID.
- uint32\_t **ulTrustedServiceMask**  
List of Trusted Service.
- uint8\_t **ucKeyType**  
Key Type Information.
- uint8\_t **ucDeviceFeatures** [BT\_DEVICE\_FEATURE\_LEN]  
Device Features.

## Field Documentation

**bool NETAPP\_BT\_DEV\_INFO::bHasLinkKey**

There is a link key present.

**char NETAPP\_BT\_DEV\_INFO::cAddr[NETAPP\_ENET\_LEN+1]**

**char NETAPP\_BT\_DEV\_INFO::cName[NETAPP\_BT\_NAME\_LEN+1]**

Hardware Address in xx:xx:xx:xx:xx notation.

Device.

**int32\_t NETAPP\_BT\_DEV\_INFO::lRssi**

Receiver Signal Strength Indicator.

**NETAPP\_BT\_HID\_INFO NETAPP\_BT\_DEV\_INFO::tHidInfo**

HID descriptor information.

**uint8\_t NETAPP\_BT\_DEV\_INFO::ucDeviceFeatures[BT\_DEVICE\_FEATURE\_LEN]**

Device Features.

**uint8\_t NETAPP\_BT\_DEV\_INFO::ucKeyType**

Key Type Information.

**uint8\_t NETAPP\_BT\_DEV\_INFO::ucMajorClassDev**

Major class of device (see spec)

**uint8\_t NETAPP\_BT\_DEV\_INFO::ucMinorClassDev**

Minor class of device (see spec)

**uint32\_t NETAPP\_BT\_DEV\_INFO::ulServiceMask**

Discovered services.

**uint32\_t NETAPP\_BT\_DEV\_INFO::ulTrustedServiceMask**

List of Trusted Service.

**uint8\_t NETAPP\_BT\_DEV\_INFO::usLinkKey[NETAPP\_LINK\_KEY\_LEN]**

Link key obtained from connection.

**uint16\_t NETAPP\_BT\_DEV\_INFO::usProductID**

Product ID.

**uint16\_t NETAPP\_BT\_DEV\_INFO::usServiceClassDev**

Service class of device (see spec)

**uint16\_t NETAPP\_BT\_DEV\_INFO::usVendorID**

Vendor ID.

The documentation for this struct was generated from netapp.h.

---

## NETAPP\_BT\_HID\_INFO Struct Reference

Bluetooth HID Information.

```
#include <netapp.h>
```

### Data Fields

- **uint32\_t ulLength**  
Length of the descriptor.
- **uint8\_t usData [NETAPP\_HID\_DSCPINFO\_MAX]**  
Buffer containing the descriptor.

### Field Documentation

**uint32\_t NETAPP\_BT\_HID\_INFO::ulLength**

Length of the descriptor.

**uint8\_t NETAPP\_BT\_HID\_INFO::usData[NETAPP\_HID\_DSCPINFO\_MAX]**

Buffer containing the descriptor.

The documentation for this struct was generated from netapp.h.

---

## NETAPP\_BT\_HID\_VOICE\_INFO Struct Reference

HID Voice Info structure.

```
#include <netapp.h>
```

### Data Fields

- **uint8\_t nbChannels**  
Number of channels (generally 1 mono)
- **uint32\_t sampleRate**  
Sample rate in Hertz.
- **uint16\_t bitsPerSample**  
Number of bits per sample.
- **char hidAudioFilename [NETAPP\_BT\_HID\_AUDIO\_FILENAME\_LEN]**  
Audio filename full path.
- **bool isAudioDevice**

### Field Documentation

**uint16\_t NETAPP\_BT\_HID\_VOICE\_INFO::bitsPerSample**

Number of bits per sample.

**char NETAPP\_BT\_HID\_VOICE\_INFO::hidAudioFilename[NETAPP\_BT\_HID\_AUDIO\_FILENAME\_LEN]**

Audio filename full path.

**bool NETAPP\_BT\_HID\_VOICE\_INFO::isAudioDevice**

**uint8\_t NETAPP\_BT\_HID\_VOICE\_INFO::nbChannels**

Number of channels (generally 1 mono)

**uint32\_t NETAPP\_BT\_HID\_VOICE\_INFO::sampleRate**

Sample rate in Hertz.

The documentation for this struct was generated from netapp.h.

---

## NETAPP\_BT\_SETTINGS Struct Reference

Bluetooth Settings.

```
#include <netapp.h>
```

### Data Fields

- **bool bDiscoverable**  
Device is/is not discoverable.
- **bool bAutoPair**  
Automatically accept Incoming Pair requests.
- **uint8\_t ucPinCode [NETAPP\_BT\_PIN\_CODE\_LEN]**  
Security pin used for Pairing.
- **uint32\_t ulPinLength**  
Length of the security pin.

### Field Documentation

**bool NETAPP\_BT\_SETTINGS::bAutoPair**

Automatically accept Incoming Pair requests.

**bool NETAPP\_BT\_SETTINGS::bDiscoverable**

Device is/is not discoverable.

**uint8\_t NETAPP\_BT\_SETTINGS::ucPinCode[NETAPP\_BT\_PIN\_CODE\_LEN]**

Security pin used for Pairing.

**uint32\_t NETAPP\_BT\_SETTINGS::ulPinLength**

Length of the security pin.

The documentation for this struct was generated from netapp.h.



---

## NETAPP\_HOTPLUG\_DEVICE\_INFO Struct Reference

USB hotplug information sent when NetApp detects a hotplug event.

```
#include <netapp.h>
```

### Data Fields

- **NETAPP\_HOTPLUG\_ACTION tAction**  
Hotplug Action (insert/remove)
- **NETAPP\_HOTPLUG\_DEVICE\_TYPE tType**  
Device Type (e.g., input)
- **const char \* pSysName**  
System name (e.g., event0, event1, etc.)
- **const char \* pVendorID**  
Vendor ID (VID)
- **const char \* pProductID**  
Product ID (PID)
- **const char \* pManufacturer**  
Manufacturer Name.
- **const char \* pProduct**  
Product Name.
- **const char \* pSerialNumber**  
Serial information.
- **const char \* pNode**  
Node.
- **const char \* pDevType**  
Device Type.

#### Remarks:

The strings returned from a hotplug event are not cached so you MUST keep a copy of them if you need them after the hotplug event.

### Field Documentation

**const char\* NETAPP\_HOTPLUG\_DEVICE\_INFO::pDevType**  
Device Type.

**const char\* NETAPP\_HOTPLUG\_DEVICE\_INFO::pManufacturer**  
Manufacturer Name.

**const char\* NETAPP\_HOTPLUG\_DEVICE\_INFO::pNode**

Node.

**const char\* NETAPP\_HOTPLUG\_DEVICE\_INFO::pProduct**

Product Name.

**const char\* NETAPP\_HOTPLUG\_DEVICE\_INFO::pProductID**

Product ID (PID)

**const char\* NETAPP\_HOTPLUG\_DEVICE\_INFO::pSerialNumber**

Serial information.

**const char\* NETAPP\_HOTPLUG\_DEVICE\_INFO::pSysName**

System name (e.g., event0, event1, etc.)

**const char\* NETAPP\_HOTPLUG\_DEVICE\_INFO::pVendorID**

Vendor ID (VID)

**NETAPP\_HOTPLUG\_ACTION NETAPP\_HOTPLUG\_DEVICE\_INFO::tAction**

Hotplug Action (insert/remove)

**NETAPP\_HOTPLUG\_DEVICE\_TYPE NETAPP\_HOTPLUG\_DEVICE\_INFO::tType**

Device Type (e.g., input)

The documentation for this struct was generated from netapp.h.

---

## NETAPP\_IFACE\_INFO Struct Reference

Interface information.

```
#include <netapp.h>
```

### Data Fields

- **NETAPP\_IFACE tiface**  
Interface type.
- **bool bPresent**  
Interface is present.
- **char cName [NETAPP\_IFACE\_NAME\_LEN+1]**  
String interface name.

See also:

**NetAppGetIfaceInfo()**

### Field Documentation

**bool NETAPP\_IFACE\_INFO::bPresent**

Interface is present.

**char NETAPP\_IFACE\_INFO::cName[NETAPP\_IFACE\_NAME\_LEN+1]**

String interface name.

**NETAPP\_IFACE NETAPP\_IFACE\_INFO::tiface**

Interface type.

The documentation for this struct was generated from netapp.h.

---

## NETAPP\_INIT\_SETTINGS Struct Reference

NetApp Initialization Settings Structure.

Settings structure that is passed to the first NetAppOpen Call as these settings can only be set when NetApp is initiated and last for the duration of the API.

```
#include <netapp.h>
```

### Data Fields

- **char \* pDeviceName**  
Null terminated Device Name string (max. 32 characters).
- **char \* WiFacePrefix**
- **bool bAllowNFS**
- **bool bBurstScanResults**
- **char \* pCountryCode**
- **char \* pManufacturer**  
Manufacturer Name (max. 64 characters).
- **char \* pModelName**  
Model Name (max. 32 characters).
- **char \* pModelNumber**  
Model Number (max. 32 characters).
- **char \* pSerialNumber**  
Manufacturer Name (max. 32 characters)
- **uint8\_t cWPSUUID [NETAPP\_UUID\_LEN]**  
UUID-E/UUID-R fields.
- **uint8\_t cTransportUUID [NETAPP\_UUID\_LEN]**  
vendor extension to support WCN-NET
- **const char \* pDBPath**  
The path to dump the database (default = /tmp)

### Field Documentation

**bool NETAPP\_INIT\_SETTINGS::bAllowNFS**

Allow Wired network config when there is an NFS mount

**bool NETAPP\_INIT\_SETTINGS::bBurstScanResults**

Send scan results one ap at a time (burst) or only as a single notification that the scan results are available.

**uint8\_t NETAPP\_INIT\_SETTINGS::cTransportUUID[NETAPP\_UUID\_LEN]**

vendor extension to support WCN-NET

UUID passed to Microsoft Rally Virtual Paring

**uint8\_t NETAPP\_INIT\_SETTINGS::cWPSUUID[NETAPP\_UUID\_LEN]**

UUID-E/UUID-R fields.

The WPS UUID inserted in the M1/M2

**char\* NETAPP\_INIT\_SETTINGS::pCountryCode**

The settings is used to determine the country and power level settings for the dongle. Normally this settings is programmed into the OTP of the dongle however sometimes the setting needs to change (i.e., FCC testing is done after the dongle was manufactured or a more optimized value is found. The correct country code setting should come from the WLAN team. Please consult your PM for the product to know the right setting. If left blank the country code is not set.

**const char\* NETAPP\_INIT\_SETTINGS::pDBPath**

The path to dump the database (default = /tmp)

**char\* NETAPP\_INIT\_SETTINGS::pDeviceName**

Null terminated Device Name string (max. 32 characters).

**char\* NETAPP\_INIT\_SETTINGS::pManufacturer**

Manufacturer Name (max. 64 characters.

**char\* NETAPP\_INIT\_SETTINGS::pModelName**

Model Name (max. 32 characters.

**char\* NETAPP\_INIT\_SETTINGS::pModelNumber**

Model Number (max. 32 characters.

**char\* NETAPP\_INIT\_SETTINGS::pSerialNumber**

Manufacturer Name (max. 32 characters)

**char\* NETAPP\_INIT\_SETTINGS::WiFacePrefix**

Wi-Fi interface name prefix, only used on **NetAppOpen()**, If not set the default is wln.

The documentation for this struct was generated from netapp.h.

## NETAPP\_INPUT\_INFO Struct Reference

Input Event information.

```
#include <netapp.h>
```

### Data Fields

- **uint32\_t ulKey**  
Input code.
- **bool bPressed**  
Pressed or released.

See also:

**NETAPP\_CB\_INPUT\_EVENT**

### Field Documentation

**bool NETAPP\_INPUT\_INFO::bPressed**  
Pressed or released.

**uint32\_t NETAPP\_INPUT\_INFO::ulKey**  
Input code.

The documentation for this struct was generated from netapp.h.

---

## NETAPP\_IP\_SETTINGS Struct Reference

NetApp Settings. This structure contains the network configuration settings.

```
#include <netapp.h>
```

### Data Fields

- **char cMacAddress [NETAPP\_ENET\_LEN+1]**  
MAC Address.
- **NETAPP\_IPV4\_ADDR tIpAddress**  
IP address.
- **NETAPP\_IPV4\_ADDR tSubnetMask**  
Subnet Mask.
- **NETAPP\_IPV4\_ADDR tGateway**  
Gateway Address.
- **NETAPP\_IPV4\_ADDR tPrimaryDNS**  
Primary DNS Address.
- **NETAPP\_IPV4\_ADDR tSecondaryDNS**  
Secondary DNS Address.

### Field Documentation

**char NETAPP\_IP\_SETTINGS::cMacAddress[NETAPP\_ENET\_LEN+1]**

MAC Address.

**NETAPP\_IPV4\_ADDR NETAPP\_IP\_SETTINGS::tGateway**

Gateway Address.

**NETAPP\_IPV4\_ADDR NETAPP\_IP\_SETTINGS::tIpAddress**

IP address.

**NETAPP\_IPV4\_ADDR NETAPP\_IP\_SETTINGS::tPrimaryDNS**

Primary DNS Address.

**NETAPP\_IPV4\_ADDR NETAPP\_IP\_SETTINGS::tSecondaryDNS**

Secondary DNS Address.

**NETAPP\_IPV4\_ADDR NETAPP\_IP\_SETTINGS::tSubnetMask**

Subnet Mask.

The documentation for this struct was generated from netapp.h.

---

## NETAPP\_OPEN\_SETTINGS Struct Reference

NetApp Open Settings Structure.

Structure passed to all **NetAppOpen()** API calls to set a callback.

```
#include <netapp.h>
```

### Data Fields

- **NETAPP\_CALLBACK tCallback**  
The callback to notify application of an event.
- **void \* pParam**  
Parameter passed to the callback (can be NULL)

### Field Documentation

**void\* NETAPP\_OPEN\_SETTINGS::pParam**

Parameter passed to the callback (can be NULL)

**NETAPP\_CALLBACK NETAPP\_OPEN\_SETTINGS::tCallback**

The callback to notify application of an event.

The documentation for this struct was generated from netapp.h.



---

## NETAPP\_P2P\_DISCOVER\_PARAMS Struct Reference

Parameters for a Wi-Fi Direct Discovery.

```
#include <netapp.h>
```

### Data Fields

- **int32\_t ITimeoutSec**  
How long do we discover for (sec) -1 is forever.
- **int32\_t IScanTimeMs**
- **uint32\_t ulServices**
- **uint32\_t ulSocialCh**

### Field Documentation

**int32\_t NETAPP\_P2P\_DISCOVER\_PARAMS::IScanTimeMs**

How long to linger on a channel (in ms). Setting to -1 will choose default.

**int32\_t NETAPP\_P2P\_DISCOVER\_PARAMS::ITimeoutSec**

How long do we discover for (sec) -1 is forever.

**uint32\_t NETAPP\_P2P\_DISCOVER\_PARAMS::ulServices**

Bitmask of services we want to support.

See also:

**NETAPP\_P2P\_SERVICES**

**uint32\_t NETAPP\_P2P\_DISCOVER\_PARAMS::ulSocialCh**

The listen channel to park on to listen for probe requests during the Listen phases of the P2P SIG discovery procedure. If 0, a default value will be used.

The documentation for this struct was generated from netapp.h.

---

## NETAPP\_P2P\_PEER\_INFO Struct Reference

Wi-Fi Direct Peer Info.

```
#include <netapp.h>
```

### Data Fields

- **NETAPP\_WIFI\_AP\_INFO** `tInfo`
- `uint32_t` **ulServices**
- `bool` **blsGO**  
the peer is group owner
- **NETAPP\_IPV4\_ADDR** `tlpAddress`  
IP address.

### Field Documentation

**bool** `NETAPP_P2P_PEER_INFO::blsGO`

The peer is the group owner.

**NETAPP\_WIFI\_AP\_INFO** `NETAPP_P2P_PEER_INFO::tInfo`

Common Wi-Fi info (SSID, BSSID, channel, signal strength, and IEEE 802.11 modes).

**NETAPP\_IPV4\_ADDR** `NETAPP_P2P_PEER_INFO::tlpAddress`

IP address.

**uint32\_t** `NETAPP_P2P_PEER_INFO::ulServices`

Bitmask of services we want to support

See also:

**NETAPP\_P2P\_SERVICES**

The documentation for this struct was generated from `netapp.h`.

---

## NETAPP\_SETTINGS Struct Reference

General NetApp Settings Structure.

These are settings that can change on the fly in NetApp and should be passed to the first **NetAppOpen()** call.

```
#include <netapp.h>
```

### Data Fields

- bool **bZeroconfOn**
- bool **bAutoReConnect**
- bool **bForceWiFi**
- bool **bWPS2\_0**  
WPS 2.0 Support enabled.
- bool **bHideDuplicateAPs**  
If we have multiple AP's with the same SSID, we will hide all the duplicates.
- **NETAPP\_WOWL\_SETTINGS** **tWoWLSettings**  
Wake-On-Wireless-LAN Settings.
- **NETAPP\_BT\_SETTINGS** **tBtSettings**  
Bluetooth Settings.
- **NETAPP\_P2P\_DISCOVER\_PARAMS** **tDefP2PParams**  
Default P2P connection Parameters.
- bool **bAutoP2PDiscover**
- bool **bIsSoftAp**  
Enable SoftAP, default: false.
- **NETAPP\_SOFTAP\_SETTINGS** **tSoftApSettings**

### Field Documentation

#### **bool NETAPP\_SETTINGS::bAutoP2PDiscover**

Run P2P Discovery in the background and allow automatic connection to the device (not used if bP2PGOset to true.

#### **bool NETAPP\_SETTINGS::bAutoReConnect**

Automatically reconnect to the previously successful connected Wi-Fi access point if the connection goes down or if the wired interface goes down and we need to bring up the wireless interface. Also automatically reconnect to saved Bluetooth devices.

#### **bool NETAPP\_SETTINGS::bForceWiFi**

This will force the "default interface" to be Wi-Fi and will configure the Wi-Fi even when wired is LINK\_UP

**bool NETAPP\_SETTINGS::bHideDuplicateAPs**

If we have multiple AP's with the same SSID, we will hide all the duplicates.

**bool NETAPP\_SETTINGS::blsSoftAp**

Enable SoftAP, default: false.

**bool NETAPP\_SETTINGS::bWPS2\_0**

WPS 2.0 Support enabled.

**bool NETAPP\_SETTINGS::bZeroconfOn**

Automatically run the zeroconf networking upon the interface coming up/down.

**NETAPP\_BT\_SETTINGS NETAPP\_SETTINGS::tBtSettings**

Bluetooth Settings.

**NETAPP\_P2P\_DISCOVER\_PARAMS NETAPP\_SETTINGS::tDefP2PParams**

Default P2P connection Parameters.

**NETAPP\_SOFTAP\_SETTINGS NETAPP\_SETTINGS::tSoftApSettings**

SoftAp Settings to configure NetApp when the Wi-Fi interface is configured as an access point.

**NETAPP\_WOWL\_SETTINGS NETAPP\_SETTINGS::tWoWLSetsings**

Wake-On-Wireless-LAN Settings.

The documentation for this struct was generated from netapp.h.

---

## NETAPP\_SOFTAP\_SETTINGS Struct Reference

SoftAp Settings.

Structure containing the various settings when NetApp is configured as a SoftAP or when we are chosen to be the group owner in a Wi-Fi Direct Connection

```
#include <netapp.h>
```

### Data Fields

- **NETAPP\_WIFI\_AP\_INFO** **tApInfo**  
Access point info (SSID, security, etc...)
- **NETAPP\_IPV4\_ADDR** **tlpAddress**  
IP address.
- **NETAPP\_IPV4\_ADDR** **tSubnetMask**  
Subnet Mask.

### Field Documentation

**NETAPP\_WIFI\_AP\_INFO** **NETAPP\_SOFTAP\_SETTINGS::tApInfo**

Access point info (SSID, security, etc...)

**NETAPP\_IPV4\_ADDR** **NETAPP\_SOFTAP\_SETTINGS::tlpAddress**

IP address.

**NETAPP\_IPV4\_ADDR** **NETAPP\_SOFTAP\_SETTINGS::tSubnetMask**

Subnet Mask.

The documentation for this struct was generated from netapp.h.

---

## NETAPP\_WIFI\_AP\_INFO Struct Reference

NetApp Wi-Fi Access Point Information.

```
#include <netapp.h>
```

### Data Fields

- **char cSSID** [**NETAPP\_MAX\_SSID\_LEN**+1]  
Service Set Identifier (BSSID)
- **char cBSSID** [**NETAPP\_ENET\_LEN**+1]

- **char cPassword [NETAPP\_MAX\_PASSWORD\_LEN+1]**  
Basic Service set Identifier (BSSID)
- **NETAPP\_WIFI\_RSSI tRSSI**  
Received Signal Strength Indicator (generalized)
- **int32\_t IRSSI**  
Received Signal Strength Indicator (-db)
- **uint32\_t tMode**
- **NETAPP\_WIFI\_SECURITY tSecurity**  
Security modes supported.
- **bool bAdHoc**  
AdHoc network or not.
- **bool bWPS**  
AP supports/implements WPS.
- **uint32\_t ulChannel**  
Channel AP is configured on.
- **int32\_t IRate**  
Calculated Speed/Rate in 500 Kbps .units.
- **int32\_t IPhyNoise**  
The physical noise (in dBm).
- **NETAPP\_WIFI\_BANDWIDTH tChanBandwidth**  
The current channel bandwidth (20 MHz, 40 MHz, etc.).

## Field Documentation

**bool NETAPP\_WIFI\_AP\_INFO::bAdHoc**

AdHoc network or not.

**bool NETAPP\_WIFI\_AP\_INFO::bWPS**

AP supports/implements WPS.

**char NETAPP\_WIFI\_AP\_INFO::cBSSID[NETAPP\_ENET\_LEN+1]**

**char NETAPP\_WIFI\_AP\_INFO::cPassword[NETAPP\_MAX\_PASSWORD\_LEN+1]**

Basic Service set Identifier (BSSID)

Password

**char NETAPP\_WIFI\_AP\_INFO::cSSID[NETAPP\_MAX\_SSID\_LEN+1]**

Service Set Identifier (BSSID)

**int32\_t NETAPP\_WIFI\_AP\_INFO::IPhyNoise**

The physical noise (in dBm).

**int32\_t NETAPP\_WIFI\_AP\_INFO::lRate**

Calculated Speed/Rate in 500 Kbps .units.

**int32\_t NETAPP\_WIFI\_AP\_INFO::lRSSI**

Received Signal Strength Indicator (-db)

**NETAPP\_WIFI\_BANDWIDTH NETAPP\_WIFI\_AP\_INFO::tChanBandwidth**

The current channel bandwidth (20 MHz, 40 MHz, etc.).

**uint32\_t NETAPP\_WIFI\_AP\_INFO::tMode**

Supported IEEE 802.11 modes (a, b, g, n, ac, etc.) This is a bitmask using NETAPP\_WIFI\_802\_11\_MODE.

**NETAPP\_WIFI\_RSSI NETAPP\_WIFI\_AP\_INFO::tRSSI**

Received Signal Strength Indicator (generalized)

**NETAPP\_WIFI\_SECURITY NETAPP\_WIFI\_AP\_INFO::tSecurity**

Security modes supported.

**uint32\_t NETAPP\_WIFI\_AP\_INFO::ulChannel**

Channel AP is configured on.

The documentation for this struct was generated from netapp.h.

---

## NETAPP\_WOWL\_NET\_PATTERN Struct Reference

WoWL Net Pattern Info.

```
#include <netapp.h>
```

### Data Fields

- `uint32_t ulOffset`
- `char cMask [NETAPP_WOWL_NET_PATTERN_MAX_LENGTH/8]`
- `char cValue [NETAPP_WOWL_NET_PATTERN_MAX_LENGTH]`  
in bytes, of payload to match against.
- `uint8_t ucLength`  
Pattern Length (bytes).

### Field Documentation

**`char NETAPP_WOWL_NET_PATTERN::cMask[NETAPP_WOWL_NET_PATTERN_MAX_LENGTH/8]`**

Each bit of the mask corresponds to a byte of data in 'value' of the pattern -- bit i of the mask = 1 => match byte i of the pattern with payload.

**`char NETAPP_WOWL_NET_PATTERN::cValue[NETAPP_WOWL_NET_PATTERN_MAX_LENGTH]`**

in bytes, of payload to match against.  
Pattern data,

**`uint8_t NETAPP_WOWL_NET_PATTERN::ucLength`**

Pattern Length (bytes).

**`uint32_t NETAPP_WOWL_NET_PATTERN::ulOffset`**

Offset within payload to start looking for the pattern.

The documentation for this struct was generated from netapp.h.



---

## NETAPP\_WOWL\_SETTINGS Struct Reference

WoWL Settings.

```
#include <netapp.h>
```

### Data Fields

- **uint32\_t ulMask**  
Mask of events to wakeup on.
- **NETAPP\_WOWL\_NET\_PATTERN tNetPattern [NETAPP\_WOWL\_MAX\_NET\_PATTERNS]**  
Net Patterns.
- **uint32\_t ulBeaconLossSeconds**  
Number of second of beacon loss.

### Field Documentation

**NETAPP\_WOWL\_NET\_PATTERN**

**NETAPP\_WOWL\_SETTINGS::tNetPattern[NETAPP\_WOWL\_MAX\_NET\_PATTERNS]**

Net Patterns.

**uint32\_t NETAPP\_WOWL\_SETTINGS::ulBeaconLossSeconds**

Number of second of beacon loss.

**uint32\_t NETAPP\_WOWL\_SETTINGS::ulMask**

Mask of events to wakeup on.

The documentation for this struct was generated from netapp.h.

---

## sNETAPP\_ZEROCONF\_SERVICE\_INFO Struct Reference

Zero Configuration Service Information.

The following structure is passed in the callback `NETAPP_CB_ZEROCONF_SERVICE` when we browse for a service and a service is found. The service information is cached inside of **NetApp API Overview** and you can get a reference to the cached data by calling **NetAppZeroConfGetBrowseResults()**.

DO NOT free this structure; NetApp will take care of cleaning up.

```
#include <netapp.h>
```

### Data Fields

- **char \* pName**  
Name (used to lookup the rest of the data)
- **char \* pType**  
type (e.g., `_http_tcp`)
- **char \* pDomain**  
Domain (e.g., `local`)
- **char \* pHostName**  
Host Name.
- **uint32\_t ulPort**  
Port number used for the service.
- **char \* pTxtRecord**  
TXT Records for the discovered service.
- **uint32\_t ulTxtLength**  
TXT Records for the discovered service.
- **NETAPP\_IPV4\_ADDR tIpAddress**  
IP address.

### Field Documentation

**char\* sNETAPP\_ZEROCONF\_SERVICE\_INFO::pDomain**  
Domain (e.g., `local`)

**char\* sNETAPP\_ZEROCONF\_SERVICE\_INFO::pHostName**  
Host Name.

**char\* sNETAPP\_ZEROCONF\_SERVICE\_INFO::pName**  
Name (used to lookup the rest of the data)

**char\* sNETAPP\_ZEROCONF\_SERVICE\_INFO::pTxtRecord**  
TXT Records for the discovered service.

**char\* sNETAPP\_ZEROCONF\_SERVICE\_INFO::pType**

type (e.g., \_http.\_tcp)

**NETAPP\_IPV4\_ADDR sNETAPP\_ZEROCONF\_SERVICE\_INFO::tIpAddress**

IP address.

**uint32\_t sNETAPP\_ZEROCONF\_SERVICE\_INFO::ulPort**

Port number used for the service.

**uint32\_t sNETAPP\_ZEROCONF\_SERVICE\_INFO::ulTxtLength**

TXT Records for the discovered service.

The documentation for this struct was generated from netapp.h.

# Index

## B

bAdHoc  
    NETAPP\_WIFI\_AP\_INFO 78  
bAllowNFS  
    NETAPP\_INIT\_SETTINGS 68  
bAutoP2PDiscover  
    NETAPP\_SETTINGS 75  
bAutoPair  
    NETAPP\_BT\_SETTINGS 64  
bAutoReConnect  
    NETAPP\_SETTINGS 75  
bBurstScanResults  
    NETAPP\_INIT\_SETTINGS 68  
bDiscoverable  
    NETAPP\_BT\_SETTINGS 64  
bForceWiFi  
    NETAPP\_SETTINGS 75  
bHasLinkKey  
    NETAPP\_BT\_DEV\_INFO 61  
bHideDuplicateAPs  
    NETAPP\_SETTINGS 76  
bIsGO  
    NETAPP\_P2P\_PEER\_INFO 74  
bIsSoftAp  
    NETAPP\_SETTINGS 76  
bitsPerSample  
    NETAPP\_BT\_HID\_VOICE\_INFO 63  
Bluetooth 52  
    NetAppBluetoothAvkStart 53  
    NetAppBluetoothAvkStop 53  
    NetAppBluetoothAvStart 53  
    NetAppBluetoothAvStop 53  
    NetAppBluetoothConnect 54  
    NetAppBluetoothDisconnect 54  
    NetAppBluetoothDiscovery 54  
    NetAppBluetoothGetDiscoveryResults 55  
    NetAppBluetoothSendAudioBuffer 55  
    NetAppBluetoothSimplePairingAck 56  
bPresent  
    NETAPP\_IFACE\_INFO 67  
bPressed  
    NETAPP\_INPUT\_INFO 70  
BT\_DEVICE\_FEATURE\_LEN  
    NetApp API Overview 16  
bWPS

    NETAPP\_WIFI\_AP\_INFO 78  
bWPS2\_0  
    NETAPP\_SETTINGS 76  
bZeroconfOn  
    NETAPP\_SETTINGS 76

## C

cAddr  
    NETAPP\_BT\_DEV\_INFO 61  
cBSSID  
    NETAPP\_WIFI\_AP\_INFO 78  
cMacAddress  
    NETAPP\_IP\_SETTINGS 71  
cMask  
    NETAPP\_WOWL\_NET\_PATTERN 80  
cName  
    NETAPP\_BT\_DEV\_INFO 61  
    NETAPP\_IFACE\_INFO 67  
Core 26  
    NetAppAtoHwAddr 27  
    NetAppAtoN 28  
    NetAppClose 28  
    NetAppDNSLookup 28  
    NetAppGetDefaultIface 29  
    NetAppGetDefaultInitSettings 29  
    NetAppGetDefaultSettings 29  
    NetAppGetIfaceInfo 30  
    NetAppGetIfaceName 30  
    NetAppGetLinkState 31  
    NetAppGetNetworkSettings 31  
    NetAppGetSettings 32  
    NetAppHttpVoiceSearch 32  
    NetAppHwAddrToA 33  
    NetAppNtoA 33  
    NetAppNtpSetDate 33  
    NetAppOpen 34  
    NetAppPing 35  
    NetAppSetIfaceUp 35  
    NetAppSetMacAddress 36  
    NetAppSetNetworkSettings 36  
    NetAppSetSettings 37  
cPassword  
    NETAPP\_WIFI\_AP\_INFO 78  
cSSID

NETAPP\_WIFI\_AP\_INFO 78  
 cTransportUUID  
 NETAPP\_INIT\_SETTINGS 69  
 cValue  
 NETAPP\_WOWL\_NET\_PATTERN 80  
 cWPSUUID  
 NETAPP\_INIT\_SETTINGS 69

## D

Database APIs 57  
 NetAppBluetoothDeleteSavedDevInfo 57  
 NetAppBluetoothGetSavedBtDevList 58  
 NetAppWiFiDeleteSavedApInfo 58  
 NetAppWiFiGetSavedApInfoList 58

## H

hidAudioFilename  
 NETAPP\_BT\_HID\_VOICE\_INFO 63

## I

isAudioDevice  
 NETAPP\_BT\_HID\_VOICE\_INFO 63

## L

lPhyNoise  
 NETAPP\_WIFI\_AP\_INFO 78  
 lRate  
 NETAPP\_WIFI\_AP\_INFO 79  
 lRSSI  
 NETAPP\_WIFI\_AP\_INFO 79  
 lRssi  
 NETAPP\_BT\_DEV\_INFO 61  
 lScanTimeMs  
 NETAPP\_P2P\_DISCOVER\_PARAMS 73  
 lTimeoutSec  
 NETAPP\_P2P\_DISCOVER\_PARAMS 73

## N

nbChannels  
 NETAPP\_BT\_HID\_VOICE\_INFO 63  
 NetApp API Overview  
 BT\_DEVICE\_FEATURE\_LEN 16  
 NETAPP\_BT\_AVK\_STATE 18  
 NETAPP\_BT\_AVK\_STATE\_PLAY 18  
 NETAPP\_BT\_AVK\_STATE\_STOP 18

NETAPP\_BT\_AV\_MODE 18  
 NETAPP\_BT\_AV\_MODE\_MONO 18  
 NETAPP\_BT\_AV\_MODE\_NONE 18  
 NETAPP\_BT\_AV\_MODE\_STEREO 18  
 NETAPP\_BT\_HID\_AUDIO\_FILENAME\_LEN 16  
 NETAPP\_BT\_NAME\_LEN 16  
 NETAPP\_BT\_PIN\_CODE\_LEN 16  
 NETAPP\_BT\_SERVICE\_A2DP 18  
 NETAPP\_BT\_SERVICE\_ALL 18  
 NETAPP\_BT\_SERVICE\_AVRCP 18  
 NETAPP\_BT\_SERVICE\_FTP 18  
 NETAPP\_BT\_SERVICE\_HFP 18  
 NETAPP\_BT\_SERVICE\_HID 18  
 NETAPP\_BT\_SERVICE\_HSP 18  
 NETAPP\_BT\_SERVICE\_NONE 18  
 NETAPP\_BT\_SERVICE\_OPP 18  
 NETAPP\_BT\_SERVICE\_TYPE 18  
 NETAPP\_BT\_SP\_CONFIRM\_REQUEST 18  
 NETAPP\_BT\_SP\_EVENT 18  
 NETAPP\_BT\_SP\_NOTIFY 18  
 NETAPP\_CALLBACK 17  
 NETAPP\_CANCELED 23  
 NETAPP\_CB\_BT\_AUTH\_COMPLETE 20  
 NETAPP\_CB\_BT\_AVK\_CHUNK 21  
 NETAPP\_CB\_BT\_AVK\_STATE 21  
 NETAPP\_CB\_BT\_DISCOVERY\_RESULTS 20  
 NETAPP\_CB\_BT\_HID\_VOICE\_INFO 20  
 NETAPP\_CB\_BT\_SP\_CONFIRM\_REQ 20  
 NETAPP\_CB\_BT\_SP\_NOTIFY 20  
 NETAPP\_CB\_CONNECT 19  
 NETAPP\_CB\_DHCP\_LEASE\_RESPONSE 20  
 NETAPP\_CB\_DISCONNECT 19  
 NETAPP\_CB\_DNSLOOKUP 19  
 NETAPP\_CB\_DYING 21  
 NETAPP\_CB\_FETCHED\_APINFO 19  
 NETAPP\_CB\_HOTPLUG 20  
 NETAPP\_CB\_INPUT\_EVENT 19  
 NETAPP\_CB\_INVALID 19  
 NETAPP\_CB\_INVITE 19  
 NETAPP\_CB\_LINK 19  
 NETAPP\_CB\_MAX 21  
 NETAPP\_CB\_NTPDATE 19  
 NETAPP\_CB\_P2P\_CONNECT 20  
 NETAPP\_CB\_P2P\_PEER 20  
 NETAPP\_CB\_PING 19  
 NETAPP\_CB\_RSSI\_EVENT 20  
 NETAPP\_CB\_SCAN\_DONE 19  
 NETAPP\_CB\_SCANNED\_APINFO 19

NETAPP\_CB\_SETSETTINGS [19](#)  
NETAPP\_CB\_TYPE [19](#)  
NETAPP\_CB\_VOICE\_REC\_DONE [20](#)  
NETAPP\_CB\_ZEROCONF [20](#)  
NETAPP\_DEVICE\_TYPE [21](#)  
NETAPP\_DEVICE\_TYPE\_BD [21](#)  
NETAPP\_DEVICE\_TYPE\_DTV [21](#)  
NETAPP\_DEVICE\_TYPE\_OTHER [21](#)  
NETAPP\_DHCP\_FAILURE [23](#)  
NETAPP\_ENET\_LEN [16](#)  
NETAPP\_FAILURE [23](#)  
NETAPP\_HANDLE [17](#)  
NETAPP\_HID\_DSCPINFO\_MAX [16](#)  
NETAPP\_HOST\_NOT\_FOUND [23](#)  
NETAPP\_HOTPLUG\_ACTION [21](#)  
NETAPP\_HOTPLUG\_ADD [21](#)  
NETAPP\_HOTPLUG\_DEVICE\_BLUETOOTH [21](#)  
NETAPP\_HOTPLUG\_DEVICE\_TYPE [21](#)  
NETAPP\_HOTPLUG\_DEVICE\_USB [21](#)  
NETAPP\_HOTPLUG\_DEVICE\_USB\_INPUT [21](#)  
NETAPP\_HOTPLUG\_DEVICE\_WIFI [21](#)  
NETAPP\_HOTPLUG\_REMOVE [21](#)  
NETAPP\_HW\_ADDR [17](#)  
NETAPP\_HW\_ADDR\_LEN [16](#)  
NETAPP\_IFACE [22](#)  
NETAPP\_IFACE\_BLUETOOTH [22](#)  
NETAPP\_IFACE\_ETH0 [22](#)  
NETAPP\_IFACE\_ETH1 [22](#)  
NETAPP\_IFACE\_ETH2 [22](#)  
NETAPP\_IFACE\_ETH3 [22](#)  
NETAPP\_IFACE\_ETH4 [22](#)  
NETAPP\_IFACE\_ETH5 [22](#)  
NETAPP\_IFACE\_LOOPBACK [22](#)  
NETAPP\_IFACE\_MAX [22](#)  
NETAPP\_IFACE\_NAME\_LEN [16](#)  
NETAPP\_IFACE\_P2P [22](#)  
NETAPP\_IFACE\_WIRED [22](#)  
NETAPP\_IFACE\_WIRED\_MAX [22](#)  
NETAPP\_IFACE\_WIRELESS [22](#)  
NETAPP\_INCORRECT\_PASSWORD [23](#)  
NETAPP\_INVALID\_PARAMETER [23](#)  
NETAPP\_INVALID\_PIN [23](#)  
NETAPP\_INVALID\_STATE [23](#)  
NETAPP\_IP\_MODE [22](#)  
NETAPP\_IP\_MODE\_AUTO\_IP [22](#)  
NETAPP\_IP\_MODE\_DYNAMIC [22](#)  
NETAPP\_IP\_MODE\_OFF [22](#)  
NETAPP\_IP\_MODE\_STATIC [22](#)  
NETAPP\_IPV4\_ADDR [17](#)  
NETAPP\_LINK\_ACQUIRING [22](#)  
NETAPP\_LINK\_DOWN [22](#)  
NETAPP\_LINK\_KEY\_LEN [16](#)  
NETAPP\_LINK\_STATE [22](#)  
NETAPP\_LINK\_UP [22](#)  
NETAPP\_MAX\_PASSWORD\_LEN [16](#)  
NETAPP\_MAX\_SSID\_LEN [16](#)  
NETAPP\_NETWORK\_UNREACHABLE [23](#)  
NETAPP\_NOT\_FOUND [23](#)  
NETAPP\_NOT\_IMPLEMENTED [23](#)  
NETAPP\_NOT\_SUPPORTED [23](#)  
NETAPP\_NO\_WAIT [16](#)  
NETAPP\_NULL\_PTR [23](#)  
NETAPP\_OUT\_OF\_MEMORY [23](#)  
NETAPP\_P2P\_SERVICES [22](#)  
NETAPP\_P2P\_SVC\_ALL [22](#)  
NETAPP\_P2P\_SVC\_DISPLAY [22](#)  
NETAPP\_P2P\_SVC\_FILE\_TX [22](#)  
NETAPP\_P2P\_SVC\_NONE [22](#)  
NETAPP\_P2P\_SVC\_PRINT [22](#)  
NETAPP\_RETCODE [23](#)  
NETAPP\_SCAN\_EMPTY [23](#)  
NETAPP\_SOCKET\_ERROR [23](#)  
NETAPP\_SUCCESS [23](#)  
NETAPP\_TIMEOUT [23](#)  
NETAPP\_UUID\_LEN [16](#)  
NETAPP\_VERSION\_INC [16](#)  
NETAPP\_VERSION\_MAJOR [16](#)  
NETAPP\_VERSION\_MINOR [16](#)  
NETAPP\_WAIT\_FOREVER [17](#)  
NETAPP\_WIFI\_802\_11\_MODE [23](#)  
NETAPP\_WIFI\_802\_11\_MODE\_A [23](#)  
NETAPP\_WIFI\_802\_11\_MODE\_B [23](#)  
NETAPP\_WIFI\_802\_11\_MODE\_G [23](#)  
NETAPP\_WIFI\_802\_11\_MODE\_N [23](#)  
NETAPP\_WIFI\_802\_11\_NONE [23](#)  
NETAPP\_WIFI\_BANDWIDTH [24](#)  
NETAPP\_WIFI\_BANDWIDTH\_10MHz [24](#)  
NETAPP\_WIFI\_BANDWIDTH\_20MHz [24](#)  
NETAPP\_WIFI\_BANDWIDTH\_40MHz [24](#)  
NETAPP\_WIFI\_BANDWIDTH\_INVALID [24](#)  
NETAPP\_WIFI\_RSSI [24](#)  
NETAPP\_WIFI\_RSSI\_EXCELLENT [24](#)  
NETAPP\_WIFI\_RSSI\_FAIR [24](#)  
NETAPP\_WIFI\_RSSI\_GOOD [24](#)  
NETAPP\_WIFI\_RSSI\_NONE [24](#)  
NETAPP\_WIFI\_RSSI\_POOR [24](#)

NETAPP\_WIFI\_SECURITY [24](#)  
NETAPP\_WIFI\_SECURITY\_AUTO\_DETECT [24](#)  
NETAPP\_WIFI\_SECURITY\_INVALID [24](#)  
NETAPP\_WIFI\_SECURITY\_NONE [24](#)  
NETAPP\_WIFI\_SECURITY\_NOT\_SUPPORTED [24](#)  
NETAPP\_WIFI\_SECURITY\_WEP [24](#)  
NETAPP\_WIFI\_SECURITY\_WPA2\_PSK\_AES [24](#)  
NETAPP\_WIFI\_SECURITY\_WPA2\_PSK\_TKIP [24](#)  
NETAPP\_WIFI\_SECURITY\_WPA\_PSK\_AES [24](#)  
NETAPP\_WIFI\_SECURITY\_WPA\_PSK\_TKIP [24](#)  
NETAPP\_WOWL\_EVENT [24](#)  
NETAPP\_WOWL\_EVENT\_DISASSOC\_DEAUTH [24](#)  
NETAPP\_WOWL\_EVENT\_LOSS\_OF\_BEACON [24](#)  
NETAPP\_WOWL\_EVENT\_MAGIC\_PATTERN [24](#)  
NETAPP\_WOWL\_EVENT\_NET\_PATTERN [24](#)  
NETAPP\_WOWL\_EVENT\_NONE [24](#)  
NETAPP\_WOWL\_MAX\_NET\_PATTERNS [17](#)  
NETAPP\_WOWL\_NET\_PATTERN\_MAX\_LENGTH

- [17](#)
- NETAPP\_WPS\_2\_ERR\_INCOMPATIBLE [23](#)
- NETAPP\_WPS\_MULTIPLE\_AP\_FOUND [23](#)
- NETAPP\_ZEROCONF\_NAME\_LEN [17](#)
- NETAPP\_ZEROCONF\_SERVICE\_FOUND [25](#)
- NETAPP\_ZEROCONF\_SERVICE\_INFO [17](#)
- NETAPP\_ZEROCONF\_SERVICE\_REMOVED [25](#)
- NETAPP\_ZEROCONF\_SERVICE\_STATE [25](#)
- NetAppAtoHwAddr
  - Core [27](#)
- NetAppAtoN
  - Core [28](#)
- NetAppBluetoothAvkStart
  - Bluetooth [53](#)
- NetAppBluetoothAvkStop
  - Bluetooth [53](#)
- NetAppBluetoothAvStart
  - Bluetooth [53](#)
- NetAppBluetoothAvStop
  - Bluetooth [53](#)
- NetAppBluetoothConnect
  - Bluetooth [54](#)
- NetAppBluetoothDeleteSavedDevInfo
  - Database APIs [57](#)
- NetAppBluetoothDisconnect
  - Bluetooth [54](#)
- NetAppBluetoothDiscovery
  - Bluetooth [54](#)
- NetAppBluetoothGetDiscoveryResults
  - Bluetooth [55](#)
- NetAppBluetoothGetSavedBtDevList
  - Database APIs [58](#)
- NetAppBluetoothSendAudioBuffer
  - Bluetooth [55](#)
- NetAppBluetoothSimplePairingAck
  - Bluetooth [56](#)
- NETAPP\_BT\_AUDIO\_FORMAT [59](#)
  - tMode [59](#)
  - ucBitsPerSample [59](#)
  - ulSampleRate [59](#)
- NETAPP\_BT\_AVK\_STATE
  - NetApp API Overview [18](#)
- NETAPP\_BT\_AVK\_STATE\_PLAY
  - NetApp API Overview [18](#)
- NETAPP\_BT\_AVK\_STATE\_STOP
  - NetApp API Overview [18](#)
- NETAPP\_BT\_AV\_MODE
  - NetApp API Overview [18](#)
- NETAPP\_BT\_AV\_MODE\_MONO
  - NetApp API Overview [18](#)
- NETAPP\_BT\_AV\_MODE\_NONE
  - NetApp API Overview [18](#)
- NETAPP\_BT\_AV\_MODE\_STEREO
  - NetApp API Overview [18](#)
- NETAPP\_BT\_DEV\_INFO [60](#)
  - bHasLinkKey [61](#)
  - cAddr [61](#)
  - cName [61](#)
  - lRssi [61](#)
  - tHidInfo [61](#)
  - ucDeviceFeatures [61](#)
  - ucKeyType [61](#)
  - ucMajorClassDev [61](#)
  - ucMinorClassDev [61](#)
  - ulServiceMask [61](#)
  - ulTrustedServiceMask [61](#)
  - usLinkKey [61](#)
  - usProductID [61](#)
  - usServiceClassDev [61](#)
  - usVendorID [61](#)
- NETAPP\_BT\_HID\_AUDIO\_FILENAME\_LEN
  - NetApp API Overview [16](#)
- NETAPP\_BT\_HID\_INFO [62](#)
  - ulLength [62](#)
  - usData [62](#)
- NETAPP\_BT\_HID\_VOICE\_INFO [63](#)
  - bitsPerSample [63](#)
  - hidAudioFilename [63](#)
  - isAudioDevice [63](#)
  - nbChannels [63](#)
  - sampleRate [63](#)
- NETAPP\_BT\_NAME\_LEN
  - NetApp API Overview [16](#)
- NETAPP\_BT\_PIN\_CODE\_LEN
  - NetApp API Overview [16](#)
- NETAPP\_BT\_SERVICE\_A2DP
  - NetApp API Overview [18](#)
- NETAPP\_BT\_SERVICE\_ALL
  - NetApp API Overview [18](#)
- NETAPP\_BT\_SERVICE\_AVRCP
  - NetApp API Overview [18](#)
- NETAPP\_BT\_SERVICE\_FTP
  - NetApp API Overview [18](#)
- NETAPP\_BT\_SERVICE\_HFP
  - NetApp API Overview [18](#)
- NETAPP\_BT\_SERVICE\_HID



NetApp API Overview	18	NETAPP_CB_FETCHED_APINFO	NetApp API Overview	19
NETAPP_BT_SERVICE_HSP		NetApp API Overview	19	
NetApp API Overview	18	NETAPP_CB_HOTPLUG	NetApp API Overview	20
NETAPP_BT_SERVICE_NONE		NetApp API Overview	19	
NetApp API Overview	18	NETAPP_CB_INPUT_EVENT	NetApp API Overview	19
NETAPP_BT_SERVICE_OPP		NETAPP_CB_INVALID	NetApp API Overview	19
NetApp API Overview	18	NETAPP_CB_INVITE	NetApp API Overview	19
NETAPP_BT_SERVICE_TYPE		NETAPP_CB_LINK	NetApp API Overview	19
NetApp API Overview	18	NETAPP_CB_MAX	NetApp API Overview	21
NETAPP_BT_SETTINGS	64	NETAPP_CB_NTPDATE	NetApp API Overview	19
bAutoPair	64	NETAPP_CB_P2P_CONNECT	NetApp API Overview	20
bDiscoverable	64	NETAPP_CB_P2P_PEER	NetApp API Overview	20
ucPinCode	64	NETAPP_CB_PING	NetApp API Overview	19
ulPinLength	64	NETAPP_CB_RSSI_EVENT	NetApp API Overview	20
NETAPP_BT_SP_CONFIRM_REQUEST		NETAPP_CB_SCAN_DONE	NetApp API Overview	19
NetApp API Overview	18	NETAPP_CB_SCANNED_APINFO	NetApp API Overview	19
NETAPP_BT_SP_EVENT		NETAPP_CB_SETSETTINGS	NetApp API Overview	19
NetApp API Overview	18	NETAPP_CB_TYPE	NetApp API Overview	19
NETAPP_BT_SP_NOTIFY		NETAPP_CB_VOICE_REC_DONE	NetApp API Overview	20
NetApp API Overview	18	NETAPP_CB_ZEROCONF	NetApp API Overview	20
NETAPP_CALLBACK		NetAppClose		
NetApp API Overview	17	Core	28	
NETAPP_CANCELED		NETAPP_DEVICE_TYPE	NetApp API Overview	21
NetApp API Overview	23	NETAPP_DEVICE_TYPE_BD	NetApp API Overview	21
NETAPP_CB_BT_AUTH_COMPLETE		NETAPP_DEVICE_TYPE_DTV	NetApp API Overview	21
NetApp API Overview	20	NETAPP_DEVICE_TYPE_OTHER	NetApp API Overview	21
NETAPP_CB_BT_AVK_CHUNK		NETAPP_DHCP_FAILURE	NetApp API Overview	23
NetApp API Overview	21			
NETAPP_CB_BT_AVK_STATE				
NetApp API Overview	21			
NETAPP_CB_BT_DISCOVERY_RESULTS				
NetApp API Overview	20			
NETAPP_CB_BT_HID_VOICE_INFO				
NetApp API Overview	20			
NETAPP_CB_BT_SP_CONFIRM_REQ				
NetApp API Overview	20			
NETAPP_CB_BT_SP_NOTIFY				
NetApp API Overview	20			
NETAPP_CB_CONNECT				
NetApp API Overview	19			
NETAPP_CB_DHCP_LEASE_RESPONSE				
NetApp API Overview	20			
NETAPP_CB_DISCONNECT				
NetApp API Overview	19			
NETAPP_CB_DNSLOOKUP				
NetApp API Overview	19			
NETAPP_CB_DYING				
NetApp API Overview	21			

NetAppDNSLookup  
     Core [28](#)  
 NETAPP\_ENET\_LEN  
     NetApp API Overview [16](#)  
 NETAPP\_FAILURE  
     NetApp API Overview [23](#)  
 NetAppGetDefaultIface  
     Core [29](#)  
 NetAppGetDefaultInitSettings  
     Core [29](#)  
 NetAppGetDefaultSettings  
     Core [29](#)  
 NetAppGetIfaceInfo  
     Core [30](#)  
 NetAppGetIfaceName  
     Core [30](#)  
 NetAppGetLinkState  
     Core [31](#)  
 NetAppGetNetworkSettings  
     Core [31](#)  
 NetAppGetSettings  
     Core [32](#)  
 NETAPP\_HANDLE  
     NetApp API Overview [17](#)  
 NETAPP\_HID\_DSCPINFOMAX  
     NetApp API Overview [16](#)  
 NETAPP\_HOST\_NOT\_FOUND  
     NetApp API Overview [23](#)  
 NETAPP\_HOTPLUG\_ACTION  
     NetApp API Overview [21](#)  
 NETAPP\_HOTPLUG\_ADD  
     NetApp API Overview [21](#)  
 NETAPP\_HOTPLUG\_DEVICE\_BLUETOOTH  
     NetApp API Overview [21](#)  
 NETAPP\_HOTPLUG\_DEVICE\_INFO [65](#)  
     pDevType [65](#)  
     pManufacturer [65](#)  
     pNode [66](#)  
     pProduct [66](#)  
     pProductID [66](#)  
     pSerialNumber [66](#)  
     pSysName [66](#)  
     pVendorID [66](#)  
     tAction [66](#)  
     tType [66](#)  
 NETAPP\_HOTPLUG\_DEVICE\_TYPE  
     NetApp API Overview [21](#)  
 NETAPP\_HOTPLUG\_DEVICE\_USB  
     NetApp API Overview [21](#)  
 NETAPP\_HOTPLUG\_DEVICE\_USB\_INPUT  
     NetApp API Overview [21](#)  
 NETAPP\_HOTPLUG\_DEVICE\_WIFI  
     NetApp API Overview [21](#)  
 NETAPP\_HOTPLUG\_REMOVE  
     NetApp API Overview [21](#)  
 NetAppHttpVoiceSearch  
     Core [32](#)  
 NETAPP\_HW\_ADDR  
     NetApp API Overview [17](#)  
 NETAPP\_HW\_ADDR\_LEN  
     NetApp API Overview [16](#)  
 NetAppHwAddrToA  
     Core [33](#)  
 NETAPP\_IFACE  
     NetApp API Overview [22](#)  
 NETAPP\_IFACE\_BLUETOOTH  
     NetApp API Overview [22](#)  
 NETAPP\_IFACE\_ETH0  
     NetApp API Overview [22](#)  
 NETAPP\_IFACE\_ETH1  
     NetApp API Overview [22](#)  
 NETAPP\_IFACE\_ETH2  
     NetApp API Overview [22](#)  
 NETAPP\_IFACE\_ETH3  
     NetApp API Overview [22](#)  
 NETAPP\_IFACE\_ETH4  
     NetApp API Overview [22](#)  
 NETAPP\_IFACE\_ETH5  
     NetApp API Overview [22](#)  
 NETAPP\_IFACE\_INFO [67](#)  
     bPresent [67](#)  
     cName [67](#)  
     tIface [67](#)  
 NETAPP\_IFACE\_LOOPBACK  
     NetApp API Overview [22](#)  
 NETAPP\_IFACE\_MAX  
     NetApp API Overview [22](#)  
 NETAPP\_IFACE\_NAME\_LEN  
     NetApp API Overview [16](#)  
 NETAPP\_IFACE\_P2P  
     NetApp API Overview [22](#)  
 NETAPP\_IFACE\_WIRED  
     NetApp API Overview [22](#)  
 NETAPP\_IFACE\_WIRED\_MAX  
     NetApp API Overview [22](#)  
 NETAPP\_IFACE\_WIRELESS

NetApp API Overview <a href="#">22</a>	NETAPP_LINK_KEY_LEN
NETAPP_INCORRECT_PASSWORD	NetApp API Overview <a href="#">16</a>
NetApp API Overview <a href="#">23</a>	NETAPP_LINK_STATE
NETAPP_INIT_SETTINGS <a href="#">68</a>	NetApp API Overview <a href="#">22</a>
bAllowNFS <a href="#">68</a>	NETAPP_LINK_UP
bBurstScanResults <a href="#">68</a>	NetApp API Overview <a href="#">22</a>
cTransportUUID <a href="#">69</a>	NETAPP_MAX_PASSWORD_LEN
cWPSUUID <a href="#">69</a>	NetApp API Overview <a href="#">16</a>
pCountryCode <a href="#">69</a>	NETAPP_MAX_SSID_LEN
pDBPath <a href="#">69</a>	NetApp API Overview <a href="#">16</a>
pDeviceName <a href="#">69</a>	NETAPP_NETWORK_UNREACHABLE
pManufacturer <a href="#">69</a>	NetApp API Overview <a href="#">23</a>
pModelName <a href="#">69</a>	NETAPP_NOT_FOUND
pModelNumber <a href="#">69</a>	NetApp API Overview <a href="#">23</a>
pSerialNumber <a href="#">69</a>	NETAPP_NOT_IMPLEMENTED
WiFilfacePrefix <a href="#">69</a>	NetApp API Overview <a href="#">23</a>
NETAPP_INPUT_INFO <a href="#">70</a>	NETAPP_NOT_SUPPORTED
bPressed <a href="#">70</a>	NetApp API Overview <a href="#">23</a>
ulKey <a href="#">70</a>	NETAPP_NO_WAIT
NETAPP_INVALID_PARAMETER	NetApp API Overview <a href="#">16</a>
NetApp API Overview <a href="#">23</a>	NetAppNtoA
NETAPP_INVALID_PIN	Core <a href="#">33</a>
NetApp API Overview <a href="#">23</a>	NetAppNtpSetDate
NETAPP_INVALID_STATE	Core <a href="#">33</a>
NetApp API Overview <a href="#">23</a>	NETAPP_NULL_PTR
NETAPP_IP_MODE	NetApp API Overview <a href="#">23</a>
NetApp API Overview <a href="#">22</a>	NetAppOpen
NETAPP_IP_MODE_AUTO_IP	Core <a href="#">34</a>
NetApp API Overview <a href="#">22</a>	NETAPP_OPEN_SETTINGS <a href="#">72</a>
NETAPP_IP_MODE_DYNAMIC	pParam <a href="#">72</a>
NetApp API Overview <a href="#">22</a>	tCallback <a href="#">72</a>
NETAPP_IP_MODE_OFF	NETAPP_OUT_OF_MEMORY
NetApp API Overview <a href="#">22</a>	NetApp API Overview <a href="#">23</a>
NETAPP_IP_MODE_STATIC	NETAPP_P2P_DISCOVER_PARAMS <a href="#">73</a>
NetApp API Overview <a href="#">22</a>	lScanTimeMs <a href="#">73</a>
NETAPP_IP_SETTINGS <a href="#">71</a>	lTimeoutSec <a href="#">73</a>
cMacAddress <a href="#">71</a>	ulServices <a href="#">73</a>
tGateway <a href="#">71</a>	ulSocialCh <a href="#">73</a>
tIpAddress <a href="#">71</a>	NETAPP_P2P_PEER_INFO <a href="#">74</a>
tPrimaryDNS <a href="#">71</a>	bIsGO <a href="#">74</a>
tSecondaryDNS <a href="#">71</a>	tInfo <a href="#">74</a>
tSubnetMask <a href="#">71</a>	tIpAddress <a href="#">74</a>
NETAPP_IPV4_ADDR	ulServices <a href="#">74</a>
NetApp API Overview <a href="#">17</a>	NETAPP_P2P_SERVICES
NETAPP_LINK_ACQUIRING	NetApp API Overview <a href="#">22</a>
NetApp API Overview <a href="#">22</a>	NETAPP_P2P_SVC_ALL
NETAPP_LINK_DOWN	NetApp API Overview <a href="#">22</a>
NetApp API Overview <a href="#">22</a>	NETAPP_P2P_SVC_DISPLAY

[NetApp API Overview 22](#)  
[NETAPP\\_P2P\\_SVC\\_FILE\\_TX](#)  
[NetApp API Overview 22](#)  
[NETAPP\\_P2P\\_SVC\\_NONE](#)  
[NetApp API Overview 22](#)  
[NETAPP\\_P2P\\_SVC\\_PRINT](#)  
[NetApp API Overview 22](#)  
[NetAppPing](#)  
[Core 35](#)  
[NETAPP\\_RETCODE](#)  
[NetApp API Overview 23](#)  
[NETAPP\\_SCAN\\_EMPTY](#)  
[NetApp API Overview 23](#)  
[NetAppSetIfaceUp](#)  
[Core 35](#)  
[NetAppSetMacAddress](#)  
[Core 36](#)  
[NetAppSetNetworkSettings](#)  
[Core 36](#)  
[NetAppSetSettings](#)  
[Core 37](#)  
[NETAPP\\_SETTINGS 75](#)  
[bAutoP2PDiscover 75](#)  
[bAutoReConnect 75](#)  
[bForceWiFi 75](#)  
[bHideDuplicateAPs 76](#)  
[bIsSoftAp 76](#)  
[bWPS2\\_0 76](#)  
[bZeroconfOn 76](#)  
[tBtSettings 76](#)  
[tDefP2PParams 76](#)  
[tSoftApSettings 76](#)  
[tWoWlSettings 76](#)  
[NETAPP\\_SOCKET\\_ERROR](#)  
[NetApp API Overview 23](#)  
[NETAPP\\_SOFTAP\\_SETTINGS 77](#)  
[tApInfo 77](#)  
[tIpAddress 77](#)  
[tSubnetMask 77](#)  
[NETAPP\\_SUCCESS](#)  
[NetApp API Overview 23](#)  
[NETAPP\\_TIMEOUT](#)  
[NetApp API Overview 23](#)  
[NETAPP\\_UUID\\_LEN](#)  
[NetApp API Overview 16](#)  
[NETAPP\\_VERSION\\_INC](#)  
[NetApp API Overview 16](#)  
[NETAPP\\_VERSION\\_MAJOR](#)

[NetApp API Overview 16](#)  
[NETAPP\\_VERSION\\_MINOR](#)  
[NetApp API Overview 16](#)  
[NETAPP\\_WAIT\\_FOREVER](#)  
[NetApp API Overview 17](#)  
[NETAPP\\_WIFI\\_802\\_11\\_MODE](#)  
[NetApp API Overview 23](#)  
[NETAPP\\_WIFI\\_802\\_11\\_MODE\\_A](#)  
[NetApp API Overview 23](#)  
[NETAPP\\_WIFI\\_802\\_11\\_MODE\\_B](#)  
[NetApp API Overview 23](#)  
[NETAPP\\_WIFI\\_802\\_11\\_MODE\\_G](#)  
[NetApp API Overview 23](#)  
[NETAPP\\_WIFI\\_802\\_11\\_MODE\\_N](#)  
[NetApp API Overview 23](#)  
[NETAPP\\_WIFI\\_802\\_11\\_NONE](#)  
[NetApp API Overview 23](#)  
[NETAPP\\_WIFI\\_AP\\_INFO 77](#)  
[bAdHoc 78](#)  
[bWPS 78](#)  
[cBSSID 78](#)  
[cPassword 78](#)  
[cSSID 78](#)  
[lPhyNoise 78](#)  
[lRate 79](#)  
[lRSSI 79](#)  
[tChanBandwidth 79](#)  
[tMode 79](#)  
[tRSSI 79](#)  
[tSecurity 79](#)  
[ulChannel 79](#)  
[NETAPP\\_WIFI\\_BANDWIDTH](#)  
[NetApp API Overview 24](#)  
[NETAPP\\_WIFI\\_BANDWIDTH\\_10MHz](#)  
[NetApp API Overview 24](#)  
[NETAPP\\_WIFI\\_BANDWIDTH\\_20MHz](#)  
[NetApp API Overview 24](#)  
[NETAPP\\_WIFI\\_BANDWIDTH\\_40MHz](#)  
[NetApp API Overview 24](#)  
[NETAPP\\_WIFI\\_BANDWIDTH\\_INVALID](#)  
[NetApp API Overview 24](#)  
[NetAppWiFiConnect](#)  
[Wi-Fi API 39](#)  
[NetAppWiFiConnectByPb](#)  
[Wi-Fi API 39](#)  
[NetAppWiFiConnectByPin](#)  
[Wi-Fi API 40](#)  
[NetAppWiFiDeleteSavedApInfo](#)

- Database APIs [58](#)
- NetAppWiFiDisconnect
  - Wi-Fi API [40](#)
- NetAppWiFiGenerateWPSPin
  - Wi-Fi API [41](#)
- NetAppWiFiGetApInfo
  - Wi-Fi API [41](#)
- NetAppWiFiGetConnectedApInfo
  - Wi-Fi API [42](#)
- NetAppWiFiGetSavedApInfoList
  - Database APIs [58](#)
- NetAppWiFiGetScannedApInfo
  - Wi-Fi API [42](#)
- NetAppWiFiGetScanResults
  - Wi-Fi API [42](#)
- NetAppWiFiInviteAccept
  - Wi-Fi Invite [45](#)
- NetAppWiFiInviteReject
  - Wi-Fi Invite [46](#)
- NetAppWiFiInviteStart
  - Wi-Fi Invite [46](#)
- NetAppWiFiInviteStop
  - Wi-Fi Invite [47](#)
- NetAppWiFiIsConnected
  - Wi-Fi API [43](#)
- NetAppWiFiIsEnabled
  - Wi-Fi API [43](#)
- NetAppWiFiP2PConnect
  - Wi-Fi Direct [48](#)
- NetAppWiFiP2PDisconnect
  - Wi-Fi Direct [48](#)
- NetAppWiFiP2PDiscover
  - Wi-Fi Direct [49](#)
- NetAppWiFiP2PGetSSID
  - Wi-Fi Direct [49](#)
- NetAppWiFiP2PStopDiscovery
  - Wi-Fi Direct [49](#)
- NETAPP\_WIFI\_RSSI
  - NetApp API Overview [24](#)
- NETAPP\_WIFI\_RSSI\_EXCELLENT
  - NetApp API Overview [24](#)
- NETAPP\_WIFI\_RSSI\_FAIR
  - NetApp API Overview [24](#)
- NETAPP\_WIFI\_RSSI\_GOOD
  - NetApp API Overview [24](#)
- NETAPP\_WIFI\_RSSI\_NONE
  - NetApp API Overview [24](#)
- NETAPP\_WIFI\_RSSI\_POOR
  - NetApp API Overview [24](#)
- NetApp API Overview [24](#)
- NETAPP\_WIFI\_SECURITY
  - NetApp API Overview [24](#)
- NETAPP\_WIFI\_SECURITY\_AUTO\_DETECT
  - NetApp API Overview [24](#)
- NETAPP\_WIFI\_SECURITY\_INVALID
  - NetApp API Overview [24](#)
- NETAPP\_WIFI\_SECURITY\_NONE
  - NetApp API Overview [24](#)
- NETAPP\_WIFI\_SECURITY\_NOT\_SUPPORTED
  - NetApp API Overview [24](#)
- NETAPP\_WIFI\_SECURITY\_WEP
  - NetApp API Overview [24](#)
- NETAPP\_WIFI\_SECURITY\_WPA2\_PSK\_AES
  - NetApp API Overview [24](#)
- NETAPP\_WIFI\_SECURITY\_WPA2\_PSK\_TKIP
  - NetApp API Overview [24](#)
- NETAPP\_WIFI\_SECURITY\_WPA\_PSK\_AES
  - NetApp API Overview [24](#)
- NETAPP\_WIFI\_SECURITY\_WPA\_PSK\_TKIP
  - NetApp API Overview [24](#)
- NetAppWiFiStartScan
  - Wi-Fi API [43](#)
- NetAppWiFiStopScan
  - Wi-Fi API [44](#)
- NETAPP\_WOWL\_EVENT
  - NetApp API Overview [24](#)
- NETAPP\_WOWL\_EVENT\_DISASSOC\_DEAUTH
  - NetApp API Overview [24](#)
- NETAPP\_WOWL\_EVENT\_LOSS\_OF\_BEACON
  - NetApp API Overview [24](#)
- NETAPP\_WOWL\_EVENT\_MAGIC\_PATTERN
  - NetApp API Overview [24](#)
- NETAPP\_WOWL\_EVENT\_NET\_PATTERN
  - NetApp API Overview [24](#)
- NETAPP\_WOWL\_EVENT\_NONE
  - NetApp API Overview [24](#)
- NETAPP\_WOWL\_MAX\_NET\_PATTERNS
  - NetApp API Overview [17](#)
- NETAPP\_WOWL\_NET\_PATTERN [80](#)
  - cMask [80](#)
  - cValue [80](#)
  - ucLength [80](#)
  - uOffset [80](#)
- NETAPP\_WOWL\_NET\_PATTERN\_MAX\_LENGTH
  - NetApp API Overview [17](#)
- NETAPP\_WOWL\_SETTINGS [81](#)
  - tNetPattern [81](#)

ulBeaconLossSeconds [81](#)  
ulMask [81](#)  
NETAPP\_WPS\_2\_ERR\_INCOMPATIBLE  
  NetApp API Overview [23](#)  
NETAPP\_WPS\_MULTIPLE\_AP\_FOUND  
  NetApp API Overview [23](#)  
NetAppZeroConfBrowse  
  Zeroconf (Bonjour) [50](#)  
NetAppZeroConfGetBrowseResults  
  Zeroconf (Bonjour) [50](#)  
NETAPP\_ZEROCONF\_NAME\_LEN  
  NetApp API Overview [17](#)  
NetAppZeroConfPublish  
  Zeroconf (Bonjour) [51](#)  
NETAPP\_ZEROCONF\_SERVICE\_FOUND  
  NetApp API Overview [25](#)  
NETAPP\_ZEROCONF\_SERVICE\_INFO  
  NetApp API Overview [17](#)  
NETAPP\_ZEROCONF\_SERVICE\_REMOVED  
  NetApp API Overview [25](#)  
NETAPP\_ZEROCONF\_SERVICE\_STATE  
  NetApp API Overview [25](#)

## P

pCountryCode  
  NETAPP\_INIT\_SETTINGS [69](#)  
pDBPath  
  NETAPP\_INIT\_SETTINGS [69](#)  
pDeviceName  
  NETAPP\_INIT\_SETTINGS [69](#)  
pDevType  
  NETAPP\_HOTPLUG\_DEVICE\_INFO [65](#)  
pDomain  
  sNETAPP\_ZEROCONF\_SERVICE\_INFO [82](#)  
pHostName  
  sNETAPP\_ZEROCONF\_SERVICE\_INFO [82](#)  
pManufacturer  
  NETAPP\_HOTPLUG\_DEVICE\_INFO [65](#)  
  NETAPP\_INIT\_SETTINGS [69](#)  
pModelName  
  NETAPP\_INIT\_SETTINGS [69](#)  
pModelNumber  
  NETAPP\_INIT\_SETTINGS [69](#)  
pName  
  sNETAPP\_ZEROCONF\_SERVICE\_INFO [82](#)  
pNode  
  NETAPP\_HOTPLUG\_DEVICE\_INFO [66](#)  
pParam

  NETAPP\_OPEN\_SETTINGS [72](#)  
pProduct  
  NETAPP\_HOTPLUG\_DEVICE\_INFO [66](#)  
pProductID  
  NETAPP\_HOTPLUG\_DEVICE\_INFO [66](#)  
pSerialNumber  
  NETAPP\_HOTPLUG\_DEVICE\_INFO [66](#)  
  NETAPP\_INIT\_SETTINGS [69](#)  
pSysName  
  NETAPP\_HOTPLUG\_DEVICE\_INFO [66](#)  
pTxtRecord  
  sNETAPP\_ZEROCONF\_SERVICE\_INFO [82](#)  
pType  
  sNETAPP\_ZEROCONF\_SERVICE\_INFO [83](#)  
pVendorID  
  NETAPP\_HOTPLUG\_DEVICE\_INFO [66](#)

## S

sampleRate  
  NETAPP\_BT\_HID\_VOICE\_INFO [63](#)  
sNETAPP\_ZEROCONF\_SERVICE\_INFO [82](#)  
  pDomain [82](#)  
  pHostName [82](#)  
  pName [82](#)  
  pTxtRecord [82](#)  
  pType [83](#)  
  tIpAddress [83](#)  
  ulPort [83](#)  
  ulTxtLength [83](#)

## T

tAction  
  NETAPP\_HOTPLUG\_DEVICE\_INFO [66](#)  
tApInfo  
  NETAPP\_SOFTAP\_SETTINGS [77](#)  
tBtSettings  
  NETAPP\_SETTINGS [76](#)  
tCallback  
  NETAPP\_OPEN\_SETTINGS [72](#)  
tChanBandwidth  
  NETAPP\_WIFI\_AP\_INFO [79](#)  
tDefP2PParams  
  NETAPP\_SETTINGS [76](#)  
tGateway  
  NETAPP\_IP\_SETTINGS [71](#)  
tHidInfo  
  NETAPP\_BT\_DEV\_INFO [61](#)

tIface  
    NETAPP\_IFACE\_INFO [67](#)  
tInfo  
    NETAPP\_P2P\_PEER\_INFO [74](#)  
tIIPAddress  
    NETAPP\_IP\_SETTINGS [71](#)  
    NETAPP\_P2P\_PEER\_INFO [74](#)  
    NETAPP\_SOFTAP\_SETTINGS [77](#)  
    sNETAPP\_ZEROCONF\_SERVICE\_INFO [83](#)  
tMode  
    NETAPP\_BT\_AUDIO\_FORMAT [59](#)  
    NETAPP\_WIFI\_AP\_INFO [79](#)  
tNetPattern  
    NETAPP\_WOWL\_SETTINGS [81](#)  
tPrimaryDNS  
    NETAPP\_IP\_SETTINGS [71](#)  
tRSSI  
    NETAPP\_WIFI\_AP\_INFO [79](#)  
tSecondaryDNS  
    NETAPP\_IP\_SETTINGS [71](#)  
tSecurity  
    NETAPP\_WIFI\_AP\_INFO [79](#)  
tSoftApSettings  
    NETAPP\_SETTINGS [76](#)  
tSubnetMask  
    NETAPP\_IP\_SETTINGS [71](#)  
    NETAPP\_SOFTAP\_SETTINGS [77](#)  
tType  
    NETAPP\_HOTPLUG\_DEVICE\_INFO [66](#)  
tWoWLS settings  
    NETAPP\_SETTINGS [76](#)

## U

ucBitsPerSample  
    NETAPP\_BT\_AUDIO\_FORMAT [59](#)  
ucDeviceFeatures  
    NETAPP\_BT\_DEV\_INFO [61](#)  
ucKeyType  
    NETAPP\_BT\_DEV\_INFO [61](#)  
ucLength  
    NETAPP\_WOWL\_NET\_PATTERN [80](#)  
ucMajorClassDev  
    NETAPP\_BT\_DEV\_INFO [61](#)  
ucMinorClassDev  
    NETAPP\_BT\_DEV\_INFO [61](#)  
ucPinCode  
    NETAPP\_BT\_SETTINGS [64](#)  
ulBeaconLossSeconds

    NETAPP\_WOWL\_SETTINGS [81](#)  
ulChannel  
    NETAPP\_WIFI\_AP\_INFO [79](#)  
ulKey  
    NETAPP\_INPUT\_INFO [70](#)  
ulLength  
    NETAPP\_BT\_HID\_INFO [62](#)  
ulMask  
    NETAPP\_WOWL\_SETTINGS [81](#)  
ulOffset  
    NETAPP\_WOWL\_NET\_PATTERN [80](#)  
ulPinLength  
    NETAPP\_BT\_SETTINGS [64](#)  
ulPort  
    sNETAPP\_ZEROCONF\_SERVICE\_INFO [83](#)  
ulSampleRate  
    NETAPP\_BT\_AUDIO\_FORMAT [59](#)  
ulServiceMask  
    NETAPP\_BT\_DEV\_INFO [61](#)  
ulServices  
    NETAPP\_P2P\_DISCOVER\_PARAMS [73](#)  
    NETAPP\_P2P\_PEER\_INFO [74](#)  
ulSocialCh  
    NETAPP\_P2P\_DISCOVER\_PARAMS [73](#)  
ulTrustedServiceMask  
    NETAPP\_BT\_DEV\_INFO [61](#)  
ulTxtLength  
    sNETAPP\_ZEROCONF\_SERVICE\_INFO [83](#)  
usData  
    NETAPP\_BT\_HID\_INFO [62](#)  
usLinkKey  
    NETAPP\_BT\_DEV\_INFO [61](#)  
usProductID  
    NETAPP\_BT\_DEV\_INFO [61](#)  
usServiceClassDev  
    NETAPP\_BT\_DEV\_INFO [61](#)  
usVendorID  
    NETAPP\_BT\_DEV\_INFO [61](#)

## W

Wi-Fi API [38](#)  
    NetAppWiFiConnect [39](#)  
    NetAppWiFiConnectByPb [39](#)  
    NetAppWiFiConnectByPin [40](#)  
    NetAppWiFiDisconnect [40](#)  
    NetAppWiFiGenerateWPSPin [41](#)  
    NetAppWiFiGetApInfo [41](#)  
    NetAppWiFiGetConnectedApInfo [42](#)



NetAppWiFiGetScannedApInfo [42](#)

NetAppWiFiGetScanResults [42](#)

NetAppWiFiIsConnected [43](#)

NetAppWiFiIsEnabled [43](#)

NetAppWiFiStartScan [43](#)

NetAppWiFiStopScan [44](#)

Wi-Fi Direct [48](#)

NetAppWiFiP2PConnect [48](#)

NetAppWiFiP2PDisconnect [48](#)

NetAppWiFiP2PDiscover [49](#)

NetAppWiFiP2PGetSSID [49](#)

NetAppWiFiP2PStopDiscovery [49](#)

Wi-Fi Invite [45](#)

NetAppWiFiInviteAccept [45](#)

NetAppWiFiInviteReject [46](#)

NetAppWiFiInviteStart [46](#)

NetAppWiFiInviteStop [47](#)

WiFiFacePrefix

NETAPP\_INIT\_SETTINGS [69](#)

## Z

Zeroconf (Bonjour) [50](#)

NetAppZeroConfBrowse [50](#)

NetAppZeroConfGetBrowseResults [50](#)

NetAppZeroConfPublish [51](#)



Broadcom® Corporation reserves the right to make changes without further notice to any products or data herein to improve reliability, function, or design.

Information furnished by Broadcom Corporation is believed to be accurate and reliable. However, Broadcom Corporation does not assume any liability arising out of the application or use of this information, nor the application or use of any product or circuit described herein, neither does it convey any license under its patent rights nor the rights of others.

Connecting  
**everything®**



**BROADCOM CORPORATION**

5300 California Avenue

Irvine, CA 92617

© 2012 by BROADCOM CORPORATION. All rights reserved.

Phone: 949-926-5000

Fax: 949-926-5203

E-mail: [info@broadcom.com](mailto:info@broadcom.com)

Web: [www.broadcom.com](http://www.broadcom.com)