NetApp 7.1 API Guide

NetApp-PG101-R

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

June 28, 2012

Revision History

Revision	*Date*	*Change Description*
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

Table of Contents

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

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.

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

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_DSCPINFO_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-FiC hannel 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_APINFO**, **NETAPP_CB_FETCHED_APINFO**, **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_DSCPINFO_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 DHCPCD.

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 depreciated 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	*pcString*	Numbers-and-dots notation of IPv4 address
out	*tHwAddr*	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	*tHandle*	NetApp handle
out	*pIface*	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	*pSettings*	- 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	*pSettings*	- NetApp Settings structure

Returns:

NETAPP_RETCODE

See also:

NetAppClose

NETAPP_RETCODE NetAppGetIfaceInfo (NETAPP_HANDLE *tHandle*, NETAPP_IFACE_INFO ** *pIfaceInfo*, uint32_t * *pListLength*)

Return the systems interface information.

Parameters:

in	*tHandle*	NetApp handle
in	*pIfaceInfo*	Pointer to contain the list of interface information
out	*pListLength*	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 *tIface*, 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	*tHandle*	NetApp handle
in	*tIface*	Interface to fetch the name
out	*pString*	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 *tIface*, 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	*tHandle*	NetApp handle
in	*tIface*	The interface
out	*pLink*	Current link state

Returns:

NETAPP_RETCODE

NETAPP_RETCODE NetAppGetNetworkSettings (NETAPP_HANDLE *tHandle*, NETAPP_IFACE *tIface*, 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	*tHandle*	NetApp handle
in	*tIface*	The interface
out	*pSettings*	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	*tHandle*	NetApp handle
out	*pSettings*	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	*pHidVoiceInfo*	Bluetooth HID Voice info structure
in	*pLanguage*	Langauge string represented in ISO 639-1 Code

See also:

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	*tHwAddr*	NETAPP_HW_ADDR to convert
in	*pcString*	Pointer to a buffer to store the string
in	*ulLength*	Length of the buffer pcString

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	*ulAddress*	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	*tHandle*	NetApp handler
in	*ulPeriodMs*	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	*tHandle*	Returned handle to the NetApp API
in	*pOpenSettings*	Open settings to set callbacks. Can be NULL
in	*pInitSettings*	Initialization Settings called on the first call to this API. Must be NULL on subsequent calls to this API
in	*pSettings*	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 with call the callback when either the pin 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	*tHandle*	NetApp handle
in	*lTimeoutMs*	Time to wait for a response
in	*pcAddress*	Server name to ping

Returns:

NETAPP_RETCODE

NETAPP_RETCODE NetAppSetIfaceUp (NETAPP_HANDLE *tHandle*, NETAPP_IFACE *tIface*, 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	*tHandle*	NetApp handle
in	*tIface*	Interface to bring up/down
in	*bUp*	TRUE for up, false for DOWN

Returns:

NETAPP_RETCODE

NETAPP_RETCODE NetAppSetMacAddress (NETAPP_HANDLE *tHandle*, NETAPP_IFACE *tIface*, char * *pMacAddress*)

Set the MAC address.

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

Parameters:

in	*tHandle*	NetApp handle
in	*tIface*	The interface
in	*pMacAddress*	New MAC Address

Returns:

NETAPP_RETCODE

NETAPP_RETCODE NetAppSetNetworkSettings (NETAPP_HANDLE *tHandle*, NETAPP_IFACE *tIface*, 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 dhcpcd 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 my 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	*tHandle*	NetApp handle
in	*tIface*	The interface
in	*tMode*	IP Mode
in	*pSettings*	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	*tHandle*	NetApp handle
in	*tSettings*	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 NetAppWiFiIsEnabled** (**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	*tHandle*	NetApp handle
in	*pApInfo*	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	*tHandle*	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	*tHandle*	NetApp handle
in	*pSsid*	The name of the router connect too (or null to do a search for APs).
in	*ulPin*	Autoconfiguration pin code
in	*bEnrollee*	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	*tHandle*	NetApp handle

Returns:

NETAPP_RETCODE

NETAPP_RETCODE NetAppWiFiGenerateWPSPin (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	*tHandle*	NetApp handle
out	*pApInfo*	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	*tHandle*	NetApp handle
out	*pApInfo*	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	*tHandle*	NetApp handle
out	*pApInfoList*	Pointer to the list of APs scanned.
out	*pulScanCount*	The number of scanned APs.

Returns:

NETAPP_RETCODE

NETAPP_RETCODE NetAppWiFiIsConnected (NETAPP_HANDLE *tHandle*, bool * *pIsConnected*)

Check if Wi-Fi is connected or not.

Parameters:

in	*tHandle*	NetApp handle
out	*pIsConnected*	True if connected, otherwise false

Returns:

NETAPP_RETCODE

NETAPP_RETCODE NetAppWiFiIsEnabled (NETAPP_HANDLE *tHandle*, bool * *pIsEnabled*)

Check if Wi-Fi is enabled.

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	*tHandle*	NetApp handle
out	*pIsEnabled*	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	*tHandle*	NetApp handle
in	*lTickMs*	How often the scan and results should be done and sent to the application.
in	*lScanTimeMs*	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 NetAppWiFiInviteStart** (**NETAPP_HANDLE** tHandle)
  Start Wi-Fi Invite.
- **NETAPP_RETCODE NetAppWiFiInviteStop** (**NETAPP_HANDLE** tHandle)
  Stop Wi-Fi Invite.
- **NETAPP_RETCODE NetAppWiFiInviteAccept** (**NETAPP_HANDLE** tHandle, char *pBSSID)
  Accept a Wi-Fi Invite Request.
- **NETAPP_RETCODE NetAppWiFiInviteReject** (**NETAPP_HANDLE** tHandle, char *pBSSID)
  Reject a Wi-Fi Invite Request.

Function Documentation

NETAPP_RETCODE NetAppWiFiInviteAccept (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	*tHandle*	NetApp handle
in	*pBSSID*	The BSSID of the AP we want to accept or reject.

Returns:

NETAPP_RETCODE

See also:

NetAppWiFiInviteReject

NETAPP_RETCODE NetAppWiFiInviteReject (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	*tHandle*	NetApp handle
in	*pBSSID*	The BSSID of the access point to reject

Returns:

NETAPP_RETCODE

See also:

NetAppWiFiInviteAccept

NETAPP_RETCODE NetAppWiFiInviteStart (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	*tHandle*	NetApp handle

Returns:

NETAPP_RETCODE Will return a failure if the AP is currently connected to an access point.

NETAPP_RETCODE NetAppWiFiInviteStop (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	*tHandle*	NetApp handle
in	*pName*	the P2P Device Name
in	*ulTimeoutSec*	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	*tHandle*	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	*tHandle*	NetApp handle
in	*pParams*	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	*tHandle*	NetApp handle
out	*pBuf*	Pointer to a string buffer to store the SSID
in	*ulBufSize*	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	*tHandle*	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_FOUND_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	*tHandle*	NetApp handle
in	*pType*	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 pInfo 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	*tHandle*	NetApp handle
in	*pName*	Service name that we want to fetch the info for
out	*pInfo*	Pointer to the SERVICE_INFO 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 NETAP_SETTINGS.pDeviceName is used for the HostName where any spaces are converted to underscores.

Parameters:

in	*tHandle*	NetApp handle
in	*pName*	The Service Name
in	*pType*	The Service Type (e.g., _http._tcp)
in	*ulPort*	The Port for the service
in	*pTxtRecord*	Pointer to a buffer containing the TXT record (if any)
in	*ulTxtLength*	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 headest.
- **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	*tHandle*	NetApp handle
in	*pBtAudioFormat*	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	*tHandle*	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	*tHandle*	NetApp handle
in	*bSynchronous*	We will be feeding AV synchronously (ex. Nexus audio capture) or asynchronously (readying from a file).
in	*pBtAudioFormat*	Pointer to BT audio format parameters

Returns:

NETAPP_RETCODE

NETAPP_RETCODE NetAppBluetoothAvStop (NETAPP_HANDLE *tHandle*)

Stop AV (Audio Source) streaming to Bluetooth headest.

Parameters:

in	*tHandle*	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	*tHandle*	NetApp handle.
in	*pBtDevInfo*	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	*tHandle*	NetApp handle.
in	*pBtDevInfo*	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	*tHandle*	NetApp handle
in	*tServices*	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	*tHandle*	NetApp handle,
out	*pBtDevInfo*	Pointer to an array of discovered Bluetooth devices.
out	*pulCount*	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	*tHandle*	NetApp handle
in	*pBuf*	Pointer to buffer containing audio data
in	*ulLength*	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	*tHandle*	NetApp handle
in	*bAccept*	True to accept, false to reject
in	*pDevInfo*	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	*tHandle*	NetApp handle
in	*pDevInfo*	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 Bluetotth device list. You must free the list once you are finished with it.

Parameters:

in	*tHandle*	NetApp handle
out	*pDevInfoList*	Saved Bluetooth device list
out	*pulCount*	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	*tHandle*	NetApp handle
in	*pApInfo*	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	*tHandle*	NetApp handle
out	*pApInfoList*	Saved AP list
out	*pulCount*	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 **lRssi**
  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 form 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 * **WiFiIfacePrefix**
- 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::WiFiIfacePrefix

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 **lTimeoutSec**
  How long do we discover for (sec) -1 is forever.
- int32_t **lScanTimeMs**
- uint32_t **ulServices**
- uint32_t **ulSocialCh**

Field Documentation

int32_t NETAPP_P2P_DISCOVER_PARAMS::lScanTimeMs

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

int32_t NETAPP_P2P_DISCOVER_PARAMS::lTimeoutSec

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 **bIsGO**
  the peer is group owner
- **NETAPP_IPV4_ADDR tIpAddress**
  IP address.

Field Documentation

bool NETAPP_P2P_PEER_INFO::bIsGO

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::tIpAddress

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::bIsSoftAp

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::tWoWLSettings

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 tIpAddress**
  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::tIpAddress

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 **lRSSI**
  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 **lRate**
  Calculated Speed/Rate in 500 Kbps .units.
- int32_t **lPhyNoise**
  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::lPhyNoise

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 date 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

Connecting
everything®

BROADCOM®

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
Web: www.broadcom.com

NetApp-PG101-R June 28, 2012