July 17, 2015

**Windows DUT Porting Guidelines**

**Version 1.0 Beta**

**11n/WMM/WPA2, VHT, WMM-PS, P2P, PMF, TDLS, VE, WMM-AC**



** *2013 Wi-Fi Alliance*

*All Rights Reserved*

*10900-B Stonelake Boulevard, Suite 126*

*Austin, TX 78759*

*Phone: 512.498.9434 • Fax: 512.498.9435 • Email:* [*certifications@wi-fi.org*](mailto:certifications@wi-fi.org)

[*www.wi-fi.org*](http://www.wi-fi.org)

This document contains confidential information intended solely for Wi-Fi Alliance staff. The contents of this document are subject to changes at the later time.

**Change History**

|  |  |  |
| --- | --- | --- |
| **Version** | **Date released** | **Summary** |
| 1.0 Beta | 07/17/2015 | Beta release |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Table of Contents

[1. Scope 4](#_Toc424916010)

[2. Overview of DUT agent architecture 4](#_Toc424916011)

[3. Working with the source code 5](#_Toc424916012)

[4. Overview of porting process 7](#_Toc424916013)

[5. API implementation for new command 9](#_Toc424916014)

[6. Program specific build options 9](#_Toc424916015)

# Scope

This document provides guidance for Wi-Fi Alliance members and their software engineers who are responsible for implementing DUT code on Windows platform so that their devices can be tested and certified for Wi-Fi certification programs. It is also reckon as a baseline for open source community to extend functionality and improve quality.

The DUT agent consists of several sub-agents, i.e., control agent, configuration agent, and traffic agent. It supports Ethernet connection between the device and test console. Other interfaces can be incorporated as part of future enhancement.

# Overview of DUT agent architecture

CAPI over socket

**Control PC**

Control agent

Control agent

CAPI over socket

TLV over socket

Control module

DUT agent

TLV over socket

Config module

Traffic module

Native API or CLI

Hardware

**Device under test**

The Wi-Fi Alliance provides the sample Windows DUT agent source code which includes:

1. Control agent: The code receives the CAPI command from test console and converts it to struct which is wrapped into TLV format and sent to subsequent agent. The control agent can reside on either a control PC which is separate from underlying device or actually in the device. The interface in the sample code is TCP. Other interfaces can be developed to support a variety of connections such as USB, serial or wireless and so on.
2. Config module in DUT agent: The agent receives data packet in TLV format from network over Ethernet interface. It will then invoke the function corresponding to the decoded command. The function will in turn either the low level API or CLI batch to execute the command.
3. Traffic module in DUT agent: The agent will generate the UDP based traffic and send/receive it to/from the peers. The data packets generated are either fixed rate or maximized one which the underlying OS can support.

# Working with the source code

The overall source code structure comprises the following four subdirectories:

|  |  |
| --- | --- |
| **Directory name** | **Description** |
| /win\_ca | Contains files related to control agent communication with DUT agent and test console |
| /win\_dut | Contains files relate to DUT agent initialization and communication with control agent |
| /win\_inc | Contains header files shared by control and dut agent |
| /win\_lib | Contains various library files shared by control and dut agent |

The win\_ca and win\_dut directory contains the visual studio project file for building win\_ca.exe and win\_dut.exe binary, respectively.

The win\_ca directory includes the following source files:

|  |  |
| --- | --- |
| **Source file name** | **Description** |
| /win\_ca/win\_ca.h | Defines the global structures used by control agent. |
| /win\_ca/win\_ca.c | The main entry of control agent. |

The win\_dut directory includes the following source files:

|  |  |
| --- | --- |
| **Source file name** | **Description** |
| /win\_dut/win\_dut.h | Defines the global structures and variables used by dut agent. |
| /win\_dut/win\_dut.c | The main entry of dut agent. |
| /win\_dut/win\_dut\_init.c | Various buffer initializations. |

The win\_inc directory includes the following source files

|  |  |
| --- | --- |
| **Source file name** | **Description** |
| /iphdr.h | Various protocol header definitions used by the raw socket. |
| /resolve.h | Contains common name resolution and name printing routines. |
| /wfa\_ca\_resp.h | Contains the declaration of functions which return command execution results to test console. |
| /wfa\_cmds.h | The definition of command types. |
| /wfa\_debug.h | The declaration of functions for debugging and logging. |
| /wfa\_main.h | Contains the common headers used by control agent and dut agent. |
| /wfa\_miscs.h | Contains the declaration of miscellaneous functions. |
| /wfa\_rsp.h | Defines the data structures which store the command response information. |
| /wfa\_sock.h | The socket function declaration. |
| /wfa\_tg.h | Defines the macros and structs used for traffic generation. |
| /wfa\_tlv.h | The definition of TLV tag and its encoding/decoding routines. |
| /wfa\_types.h | Defines the string and buffer length macros. |
| /wfa\_ver.h | Defines the system version of dut agent. |
| /wfa\_wmmps.h | Macros and structs definition, and function declaration for WMM-PS |

The win\_lib directory contains the following source files:

|  |  |
| --- | --- |
| **Source file name** | **Description** |
| /wfa\_ca\_resp.c | Contains the functions which return results to test console |
| /wfa\_cmdproc.c | The functions which parse CAPI commands |
| /wfa\_cmdtbl.c | The declaration of functions for implementing CAPI commands. |
| /wfa\_cs.c | The function implementations of CAPI commands. |
| /wfa\_miscs.c | Contains the miscellaneous utility functions. |
| /wfa\_sock.c | C wrappers of winsock. |
| /wfa\_tg.c | Contains traffic-related CAPI command processing functions |
| /wfa\_thr.c | Thread function for sending/receiving packets based on the priority of packets and the affiliated functions which create QoS flow using TC APIs. |
| /wfa\_tlv.c | TLV tag encoding and decoding routines. |
| /wfa\_typestr.c | The declaration of CAPI parsing functions in control agent. |
| /wfa\_wmm\_tc.c | Contains the functions used for the programs where the user priority needs to be set for the outgoing packets. |
| /wfa\_wmmps.c | The functions used exclusively for WMM-PS program. |

# Overview of porting process

The Device Under Test (DUT) is a device with the following two network interfaces:

* The Wi-Fi Interface is typically implemented via a Wi-Fi chipset within the DUT which provides 802.11x MAC and PHY functionality together with the driver and other software.
* The control interface is typically an Ethernet link to which the DUT can be connected and receives CAPI commands.

The sample Windows DUT code is designed with easy portability and modularity in mind. Conceptually it consists of the following three functions:

* Communication: The CAPI commands are received from and the response is generated and sent back to test console via the TCP control link.
* Device configuration and action: The command and associated parameters are used to configure the device in preparation for a given test or to execute the specified action.
* Traffic generation: The data packets are generated and sent over the air through the wireless interface as per the parameters specified (payload size, framerate, and IP address and port etc.).

**OS dependent APIs:**

The OS dependency includes the use of socket and thread, and other system APIs with regard to QoS and system time. Although these APIs have been tested on Windows 7, 8 and 10 platforms, when porting to other Windows platforms, it should be aware of the existence of dependency. The following table shows the distribution of these APIs in different source files.

|  |  |
| --- | --- |
| **Windows API** | **Files** |
| socket | Mainly in wfa\_sock.c, select() is used in wfa\_dut.c, wfa\_thr.c and wfa\_wmmps.c |
| Thread/mutex/condition variable | wfa\_dut.c, wfa\_thr.c, wfa\_tg.c, wfa\_wmm\_tc.c, and wfa\_wmmps.c |
| QoS TC API | wfa\_thr.c |
| System time | wfa\_dut.c and wfa\_miscs.c |
| Registry | wfa\_cs.c |

**Configuration and action APIs:**

These APIs are mainly located in the wfa\_cs.c file and invoked by the main code to configure the device and execute the Wi-Fi specific actions.

These APIs can in turn invoke vendor’s low level APIs which will manage and control the device via the device driver. Each function in the wfa\_cs.c file corresponds to a CAPI command. Some implementations are achieved through Windows netsh utility.

Another way of implementing CAPI command is through CLI. Here are the steps to follow for the implementation:

1. Add the CAPI command to be implemented to the wfa\_cli.txt file by following the format:

* <CAPI command>-TRUE,
* <CAPI command>-FALSE,

Note: TRUE means the command needs to return the result to test console. FALSE means return null.

1. Create a batch file with exactly the same file name as CAPI command, provide the implementation in it and write the result back to windows DUT using setx. Each CLI must return an error code status through the environment variable - WFA\_CLI\_STATUS.

WFA\_CLI\_STATUS Code Definition:

* WFA\_CLI\_COMPLETE(0)
* WFA\_CLI\_ERROR(1)
* WFA\_CLI\_INVALID(2)

If there is any return value from CLI, it should be return through the environment variable WFA\_CLI\_RETURN (String). The return value format is <var1\_name, var1\_value, var2\_name, var2\_value>. For example, if CLI “STA\_P2P\_START\_GROUP\_FORMATION” returns two values “result” and “groupid” then enviorment variable WFA\_CLI\_RETURN should be set to “result,client,groupid,00:11:22:33:44:55 DIRECT-S2”.

The setx or other approaches can be used to write the result back to Windows environment variables. For example,

* SETX WFA\_CLI\_STAUS 0
* SETX WFA\_CLI\_RETURN “result,client,groupid,00:11:22:33:44:55 DIRECT-S2”

# API implementation for new command

When implementing a new CAPI command, a simpler approach is to follow the steps above and do it in the CLI way. If using API is preferred, the following is the procedure:

1. Define the new command’s TLV tag in wfa\_tlv.h.
2. Declare the response function in wfaf\_ca\_resp.h and add the function implementation in wfa\_ca\_resp.c.
3. Add a command parsing handling function to the function array of nameStr in wfa\_typestr.c and implement it in wfa\_cmdproc.c.
4. Define new struct for the command in wfa\_cmds.h if it is different from dutCommand\_t and a response struct in wfa\_rsp.h if it is not a generic one as dutCmdResponse\_t.
5. Declare the function in gWfaCmdFuncTbl of wfaf\_cmdtbl.c and Implement it in wfa\_cs.c.

# Program specific build options

The source code defines a few program specific compiling switches for building the executable for either individual program or all the programs. The relationship among the switches is illustrated as below:

|  |  |
| --- | --- |
| **Build options** | **Programs to build** |
| WFA\_WMM\_WPA2 | 11n, VHT, TDLS |
| WFA\_WMM\_PS | WMM-PS |
| WFA\_WMM\_AC | WMM-AC |
| WFA\_P2P | P2P |
| WFA\_WMM\_WPA2 and WFA\_VOICE\_EXT | Voice Enterprise |
| Not specified | PMF, WFD, WFDS, HS2 |
| All specified | All the programs as state above |