

Support Wireless USB: Architecture Design

Revision 1.5
May 26, 2008

WUSB Project Team

Colin.Zou@Sun.COM

Sophia.li@Sun.COM

Raymond.Chen@Sun.COM

usb-team@Sun.COM

Sun Microsystems, Inc.

Table of Contents

1	Introduction.....	3
1.1	Wireless USB Industry Standard.....	3
1.2	References.....	3
1.3	Acronyms and Abbreviations.....	4
1.4	Document Scope.....	4
2	WUSB & UWB Modules.....	4
2.1	Solaris USB Software Stack and WUSB Modules Overview.....	4
2.2	Device Drivers and Administration Utility.....	5
2.3	Module list.....	6
2.4	Updates to usba module.....	7
2.5	Hotplug Events Handle for WUSB devices.....	7
3	WUSB Security.....	7
3.1	Association Process.....	7
3.1.1	Association Overview.....	7
3.1.2	Cable Association Process.....	8
3.1.3	Numeric Association Process.....	9
3.1.4	CC Management.....	9
3.2	Connection Process.....	10
3.2.1	Overview.....	10
3.2.2	Authentication.....	10
4	RBAC Model.....	11
4.1	Authorizations.....	11
4.2	Rights Profile.....	11
5	SMF service.....	11
6	Audit.....	12
7	Boot.....	12
8	Issues During System Installation.....	12
9	Driver IOCTLs.....	13
9.1	Radio Controller Driver IOCTLs.....	13
9.2	Host Controller Driver IOCTLs.....	13
9.3	Cable Association Driver IOCTLs.....	14
9.4	IOCTLs for Numeric Association.....	15

1 Introduction

1.1 Wireless USB Industry Standard

Wireless USB (WUSB) is a new standard from usb.org. The “Wireless Universal Serial Bus Specification” Revision 1.0 was released at May 12, 2005. WUSB chooses UWB (Ultra WideBand, another industry standard from wimedia.org) as its radio platform.

According to WUSB specification (see section 1.2 references), “Wireless USB is a logical evolution of USB. The goal is that end users view it as the same as wired USB, just without the wires.”

In Figure 1-1, the grey blocks are wired USB host/devices, the rest are WUSB hosts/devices.

There are two kinds of WUSB hosts:

- WHCI is the on-board host controller usually directly connected to PCI bus;
- HWA is a USB dongle which can be plugged to a USB2.0 port and works as a WUSB host controller.

Either a WHCI or a HWA host can talk to WUSB devices, or, DWA devices which can be used as a wireless hub to connect wired USB devices.

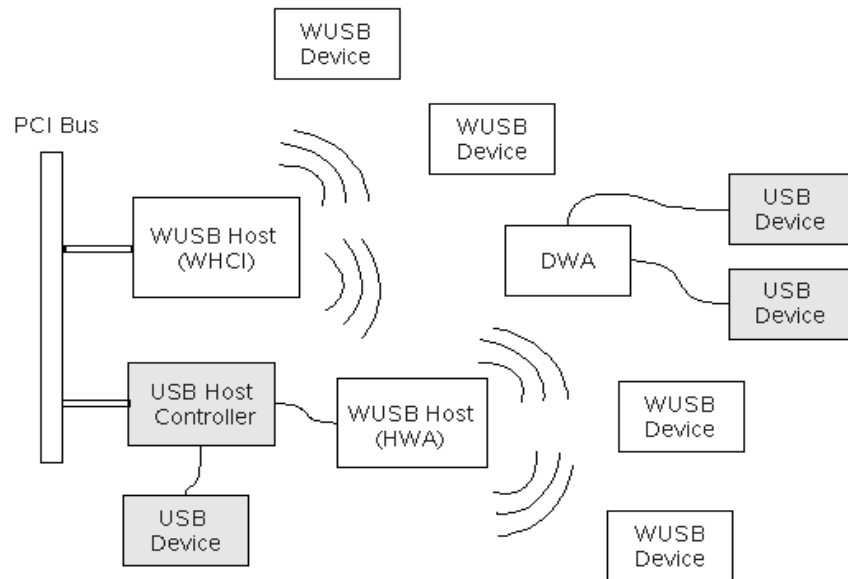


Figure 1-1 Wireless USB Devices

According to WUSB specification, the maximum number of devices that a WUSB host controller can connect at the same time is 127. This design document follows the WUSB specification.

WUSB specification tries to be compliant with USB2.0 specifications. To support WUSB devices, Solaris needs not to have another group of USB client drivers for specific USB devices, instead, just needs to have two new host controller drivers, UWB radio controller drivers and some other utilities for administration and association. WUSB host drivers are developed within the existing usba(7D) framework.

1.2 References

The following specifications or documents are related:

- [WUSB1] Wireless Universal Serial Bus Specification
- [WHCI1] Wireless Host Controller Interface (WHCI) Specification
- [AM] Association Models Supplement to the WUSB Specification
- [AMFAQ] Association Models Supplement FAQ

1.3 Acronyms and Abbreviations

USB	Universal Serial Bus
WUSB	Wireless Universal Serial Bus
UWB	Ultra Wideband
HWA	Host Wire Adapter
DWA	Device Wire Adapter
WHCI	Wireless Host Controller Interface
USBA	Solaris USB Architecture
EHCI	Enhanced Host Controller Interface
OHCI	Open Host Controller Interface
UHCI	Universal Host Controller Interface
CHID	Connection Host ID
CDID	Connection Device ID
CK	Connection Key
CC	Connection Context. It includes three elements: CHID, CDID, CK

1.4 Document Scope

This document tells the architecture design of supporting WUSB on Solaris. It gives an overview of the newly introduced drivers/modules, utilities, and the relationship between the new stuff and existing USB software stack on Solaris. It also introduces the administration tools and a daemon for WUSB/UWB support.

The goal of this design is the same as the goal of WUSB specification: “The goal is that end users view it as the same as wired USB, just without the wires.”. Any other new features are out of scope of this document.

2 WUSB & UWB Modules

2.1 Solaris USB Software Stack and WUSB Modules Overview

Solaris supports wired USB devices by a kernel software stack which includes a framework `usba(7D)`, three host controller drivers and more than ten client drivers for various device classes (printer, storage, keyboard/mouse, etc.). The module `usba` exports interfaces for USB client drivers, so that all USB client drivers talk to one set of interfaces. Refer to Chapter 19 of “Writing Device Drivers” book for details about the interfaces. They are a sub-set of Solaris DDI.

WUSB host controller drivers and other utility modules will be compliant to the existing USB framework, so that all USB client drivers can just work for WUSB devices. Figure 2-1 tells the big picture. The grey blocks are existing wired USB drivers/modules. The rest are newly introduced WUSB drivers/modules. All the existing USB client drivers are expected to work for corresponding WUSB devices. For example, WUSB printers should be supported by the existing USB printer class driver (`usbprn(7d)`).

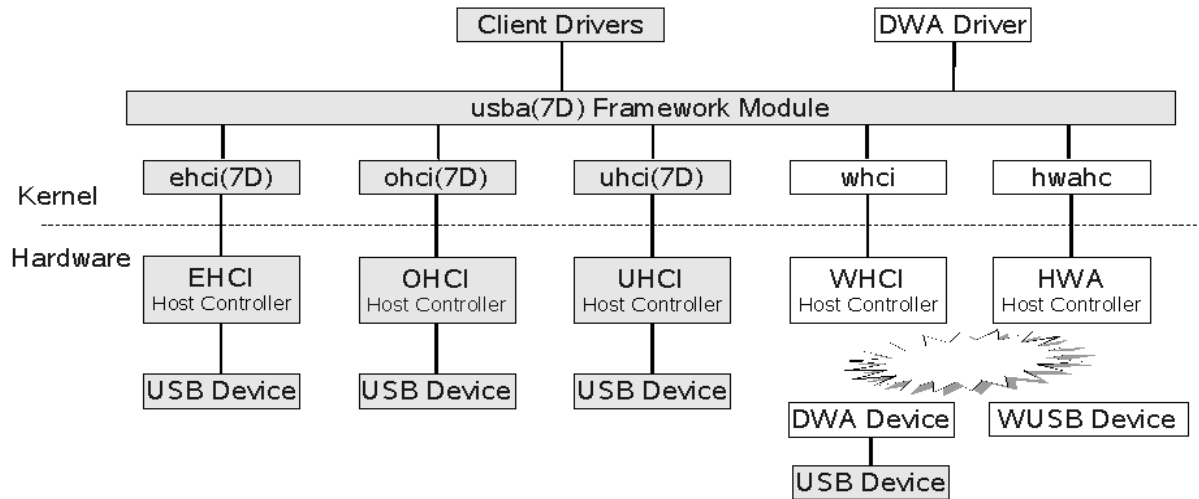


Figure 2-1 Solaris USB kernel software stack with WUSB modules

Figure 2-1 does not cover the userland administration tool and some other kernel modules for UWB radio controller.

2.2 Device Drivers and Administration Utility

Figure 2-2 tells WUSB & UWB device drivers and administration utility.

According to [WHCI1] and [WUSB1], either WHCI or HWA device must include both a radio controller and a host controller. For a HWA device, it must have two hardware interfaces at least, one is for radio controller, another is for host controller. These two interfaces have independent USB class code for driver binding names. What's more, for a USB device, it is up to the manufacturer that whether extra interfaces/functions would be implemented on it (e.g., a USB CD drive may include both storage and audio interfaces, or only a storage one.) Therefore, instead of implementing one driver to bind to the whole HWA device, it is better to have two device drivers (hwarc and hwahc) work for HWA radio and host controllers respectively. For a WHCI device, it is a different case. Both radio and host controllers are implemented in one piece of PCI device, and only one driver (whci) is used for both radio controller and host controller.

uwba is the UWB framework module, it defines the ioctl interfaces implemented by radio controller drivers. Via the WUSB administration interfaces, the ioctl commands can be sent to radio controller drivers to perform radio operations, namely, scan, beacon, channel change, etc.

WUSB administration utility talks to WUSB host drivers (whci, hwahc and hwarc) and cable association driver (wusb_ca). They work together to perform WUSB administration jobs, namely, associate new WUSB devices with hosts, manage connection contexts, etc. Refer to man page wusbadm(1M) and wusbcd(1M) for details.

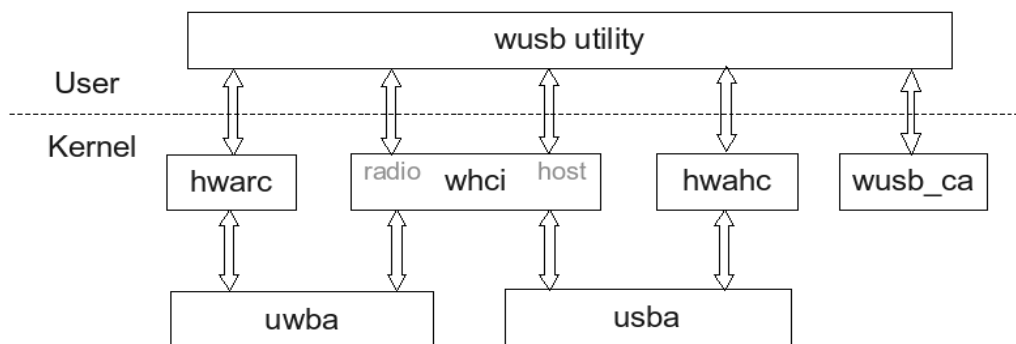


Figure 2-2 WUSB & UWB Device Drivers

2.3 Module list

- **uwba**
UWB Architecture module. It is a misc module. It supports radio controller drivers. Both HWA radio controller driver (hwarc) and whci driver register to uwba during attach.
- **whci**
WHCI driver. It is a PCI device driver. This driver has two components, whci_rc for UWB radio controller and whci_hc for WUSB host controller.
- **hwarc**
HWA radio controller driver. It is a USB client driver which is compliant to Solaris usba(7D) framework. It also registers to uwba during attach. It attaches to the radio controller interface of a HWA device.
- **hwa hc**
HWA host controller driver. It is a USB nexus driver and also works as a USB host controller driver. It is compliant to Solaris usba(7D) framework. It attaches to the host controller interface of a HWA device.
- **wusb_ca**
WUSB cable association driver. It is a USB client driver which is compliant to Solaris usba(7D) framework. It attaches to the cable association interface of a WUSB client device. It is used to associate the WUSB client device with a WUSB host.
- **dwa**
DWA device driver. It is a WUSB nexus driver which supports DWA device as a bridge between a WUSB host and wired USB devices.
- **wusbadm**
WUSB administration tool. A userland tool. By this tool, end users can administer WUSB hosts by issuing various commands to the host drivers. Another main purpose of this tool is to manage the association between host and device. Refer to manpage wusbadm(1M) for details.
- **wusbd**
WUSB daemon. It conforms to smf(5). It implements the wusb service. One of the main tasks of it is to load “Connection Context (CC)” (see WUSB spec for this term) from Solaris file system (refer to section 3.1.4 for CC management) to WUSB host drivers (whci and hwa hc) to prepare for the connection operations. Daemon will make it possible that the associated WUSB devices can be connected without manually running wusbadm tool after system reboot. This feature is critical for WUSB keyboard/mouse devices.

When this daemon starts, it will find all the available wireless hosts and load corresponding CCs to them. For hotpluggable hosts (e.g. A HWA dongle), it listens to device add/remove events. When a WUSB host is plugged, it will detect whether this is a new host to this system. If yes, it generates new host information for it; if not, load the corresponding CC to the driver instance.

For administration requests from user, wusbadm interpret the commands and send requests to wusbd. The latter will talk to drivers to perform the requests.

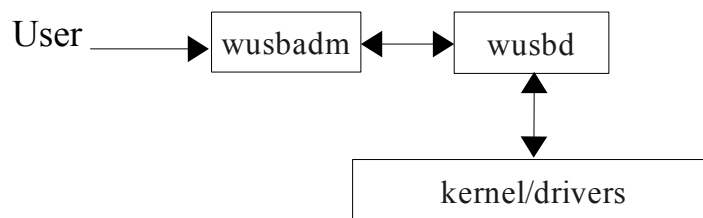


Figure 2-3 WUSB administration tool and daemon

This daemon is managed by SMF in the system category.

svc:/system/wusb:default

See section 5 for details of this service.

2.4 Updates to usba module

All existing USB APIs exported by usba module are untouched. All the updates to usba module in this project are project private implementations because they are for two WUSB host drivers only.

While WUSB supports standard USB 2.0 operations, it adds some new stuffs to support wireless features. The new WUSB descriptors and requests are extensions to USB 2.0 ones. usba module is updated mainly because some functions and data structures are added to it to support common features for two WUSB host controller drivers (whci & hwhc). These functions are called by both whci and hwhc drivers. Besides adding stuffs to usba, we have to update a few usba internal data structures and functions to support WUSB features within usba framework. The updates to usba module mainly include:

- Data structures for WUSB descriptors and requests, implementing what are described in section 7.3 & 7.4 of [WUSB1].
- Authentication functions, implementing the 4-way handshake described in section 6.2.10.9.1 of [WUSB1].
- Functions for connect/disconnect process implementation.
- Functions for CC list operations, such as add/remove CCs.
- Other miscellaneous functions for supporting whci and hwhc drivers.

2.5 Hotplug Events Handle for WUSB devices

The hotplug events of a WUSB device is actually the similar case as a wired USB device. In wired case, USB host controller drivers poll the physical USB ports to detect if there is any devices plugged or unplugged. In wireless case, WUSB host controller drivers listen to the predefined radio signals to know if a wireless device appears or disappears. If users turn on a device radio, then WUSB host controller gets the radio signal and the driver pass the events to usba framework. If a device is turned off or moved out of the range, then the WUSB host controller driver thinks it as a unplug event if it can not get the radio signal for a specific time period. Details see [WUSB1].

3 WUSB Security

The goal of WUSB security is mentioned in section 6.1.1 of “Wireless Universal Serial Bus Specification”. Quoted from the specification: “Wireless implementations of USB are wire-replacement technologies. The wire actually provides two services typically associated with security. It connects the nodes the owner/user specifically wants connected. It also protects all data in transit from casual observation or malicious modification by external agents. The goal of USB Security is to provide this same level of user-confidence for wireless connected USB devices.”

Encryption methods are mentioned in section 6.2.4 of [WUSB1]. The symmetric encryption algorithm “AES-128 Counter with CBC-Mac (CCM)” is used for data encryption. It is implemented in the hardware.

For device authentication in the connection process, WUSB uses public key encryption.

Chapter 6 of [WUSB1] defines WUSB security protocols. [AM] defines the process of association between host and device. To implement them on Solaris, some software components need to be developed in device drivers, the administration tool and the daemon. These software components mainly help on the association and connection processes.

3.1 Association Process

3.1.1 Association Overview

The association must be performed before setup the first time connection between WUSB host and device. Because, in wireless environment, when multiple devices and hosts are put nearby, there must be something to indicate which device want to connect which host. Association will generate a common CC for a pair of host and device. CC is a tuple including CHID (Connection Host ID), CDID (Connection Device ID) and CK (Connection Key). CHID and CDID are public information. CK is a shared secret (a symmetric key) between a pair of host and device which will never be sent

in plain text over air.

There are two kinds of association models:

- Cable association model

A USB cable is used to connect the host with the device for the association purpose. After the WUSB device is plugged to the USB port of the system, as any other USB drivers, wusb_ca driver is attached to the device according to driver binding names. Users then run wusbadm tool to issue the associate command to the host and device. wusb_ca driver is used for passing information between the connected host and device. By following the process described in section 3.1.2 of this document, a CC is created for a pair of host and device. Once the process is completed, the cable is no longer required and user can turn on the device radio to start the connection.

- Numeric association model

No cable is needed. The process is performed over the host and device radio. The wusb_na component in the host controller drivers (whci and hwahc) works with wusbadm tool to create a CC for the host and device.

The CC created by the association process will be saved on the host file system. It will also be saved to the device's non-volatile memory via wusb_ca driver or host controller drivers.

Refer to Figure 3-1 for the module relationship.

Some abbreviations in the Figure 3-1 and 3-2:

CCD: CC database, store CCs.

CCM: CC management function, used for CC management.

PRF: Pseudo-Random Function, used by 4-way handshake authentication.

AM: Association management function, for WUSB cable association and numeric association.

DH: Diffie-Hellman exchange function, used for WUSB numeric association.

PTK: Pairwise Temporal Key

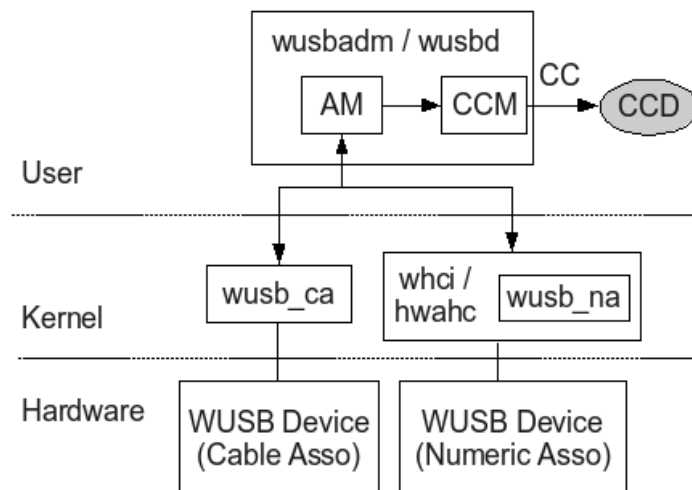


Figure 3-1 WUSB Association

3.1.2 Cable Association Process

The detail cable association process is described in section 4 of [AM]. Below is the process description when it is applied to Solaris.

1. The user connects the WUSB device to the host system's USB port using a USB cable.
2. The host system's usba framework detects the device and attaches wusb_ca driver to the device's cable association interface. (According to section 4.3.1 of [AM], a device with cable association ability must have a USB interface with class number EFh and sub-class number 03h. As any other USB class drivers, wusb_ca driver binds to the device according to the class/sub-class numbers.)

3. The user runs "wusbadm associate -h host-id -c" command to direct the WUSB host to start the cable association process. wusbadm reads the host's CHID from CCD and sends it to device through the attached wusb_ca driver.
4. The device receives the CHID and checks its CC list in its non-volatile memory. If the device has a valid CC with a matching CHID, the device will respond to the host with the CDID from the CC. Or, it responds to the host that it has not such a CC.
5. The wusbadm prompts to user and requires explicit user conditioning to confirm the association, if approved, it will then generate a new CC and send it to the device. Otherwise, abort the association. The new CC must has a freshly generated CK; the CHID and CDID keep the same if there are existing ones.
6. Upon receiving the CC, the device must store the CC in its non-volatile memory, replacing any existing CC with a matching CHID if there is.
7. After successfully sending the CC to device, wusbadm stores the same CC to host CCD.
8. Cable association completed: WUSB host and device now have the same CC.

3.1.3 Numeric Association Process

The detail numeric association process is described in section 5 of [AM]. Quoted from the spec: "The Diffie-Hellman protocol is used to establish a temporary secure channel. To guard against a man-in-the-middle attack, the host and device each display a value that is derived from the Diffie-Hellman keys, and the user is asked to verify that the two values match."

Below is the process description when it is implemented in Solaris.

1. The user runs "wusbadm associate -h host-id -n" command to direct the WUSB host to start the numeric association process. WUSB host controllers then "advertising over the Certified Wireless USB channel that it is accepting "new" device connections". The host's CHID (read from host CCD by wusbadm) is broadcast.
2. The user turns on the device. "Device listens for a host accepting new device connections and initiates a "new" connection when one is found." The device receives the host's CHID and checks its CC list in its non-volatile memory. If the device has a valid CC with a matching CHID, the device will send the CC's CDID to the host. Otherwise, it generates a CDID and sends it to the host.
3. Host and device start the Diffie-Hellman protocol. Details are skipped here because it is exactly the same as what is described in section 5.2 of [AM].
4. A common value are computed by the above steps in both host and device. 2-4 digits of it are shown on both host and device displays. On the host, it is shown by a prompt message of command "wusbadm associate -h host-id -n" performed in step 1. The message also asks user to check if the numbers shown on the host and device are matched. Users type "yes" to confirm or "no" to abort the association process. User also performs some equivalent actions on the device, say, to press "OK" button for "yes" answer.
5. If user approves the association in step 4, then the host and device compute the connection key CK on their own.
6. Numeric association completed: WUSB host and device now have the same CC. (They computed the same new CK and they got CHID/CDID from each other.) wusbadm saves the CC to host CCD; device saves the CC to its non-volatile memory.

Note: All variables used in the numeric process will be erased except for the CC.

3.1.4 CC Management

As mentioned in 3.1.1, CC is a tuple including CHID (Connection Host ID), CDID (Connection Device ID) and CK (Connection Key). CCs are persistent information for host-device association. Once they are setup, they will be repeatedly used for host-device authentication for connection. They are setup by users and should be kept in the host system until be removed by users.

On the device side, [AM] spec requires that device must be able to store at least one CC to its non-volatile memory. If a device has memory space for multiple CCs, it might be able to associate with multiple hosts at the same time. Though the device can be connected with only one host at the same time. The way that the device prioritize or update its multiple records of CCs depends on the manufacturer.

On the host side, one host can have multiple CCs, depends on how many devices are ever associated with the host. After system boot, for each WUSB host, WUSB daemon wusb daemon wusb loads CCs (if there is) from system file system to host

driver instances. The aim is to make host drivers aware the associated devices. Once the devices are turned on, the host driver can then know if the devices are associated with it before. For associated devices, the host will then connect with them without asking users further interference.

CCs are read from the host CCD (CC Database) by daemon wusbd. All CCs are stored in CCD which is a newly introduced file in an existing directory: /etc/usb/wusb_cc.conf

CCs are created by following the process mentioned in section 3.1.2 and 3.1.3. wusbd saves it to CCD. If wusbdadm needs to fetch CC from CCD, it will send the request to wusbd, the latter will read CC for it.

3.2 Connection Process

3.2.1 Overview

After the association process, the connection process can be started because the same CC is now in both host and device. If system reboot, CC will not be lost because it was saved to CCD and can be reloaded by daemon wusbd. If device is powered off, the CC will be kept because it is saved in the non-volatile memory in the device.

An authentication between host and device is necessary because, in wireless environment, when multiple devices and hosts are put nearby, either host or device need to prove to each other that it is the one expected.

The flowchart in Figure 3-2 is explaining the connection process since system boot. When system boots up, the WUSB host drivers (whci or hwahc) are attached and wait until the daemon wusbd comes up. The WUSB daemon wusbd loads the CCs from CCD and passes them to the host driver. The host driver broadcasts the CHID and scans/gets connection request from WUSB devices and performs the 4-way handshake for authentication. After a successful authentication, the host driver brings the device online and starts the encrypted data exchanges. If the device is disconnected for some reasons, the re-connection is performed.

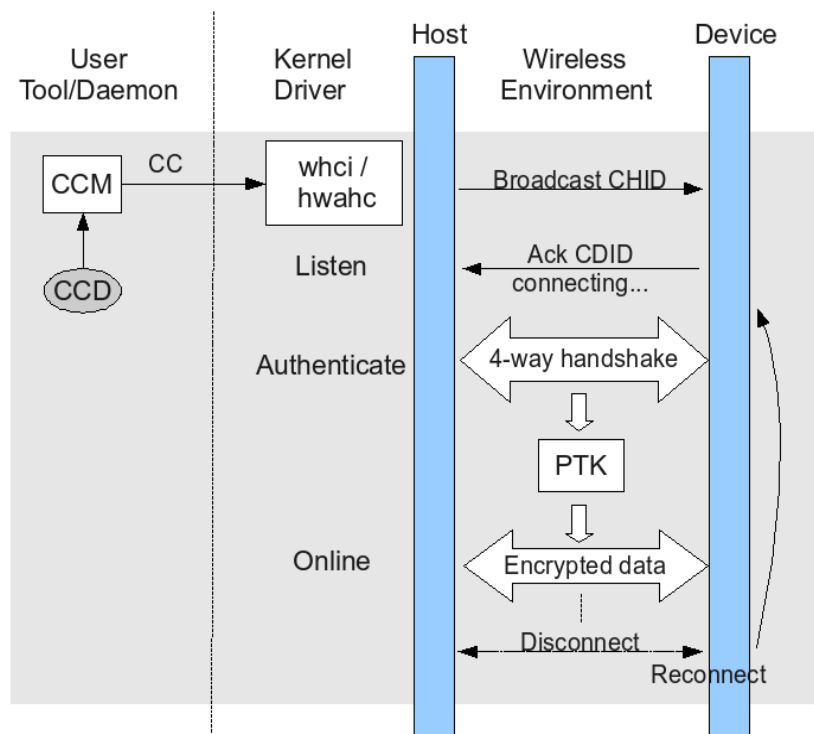


Figure 3-2 WUSB Connection Flowchart

3.2.2 Authentication

In the connection process, a 4-way handshake is performed to derive a symmetric key for data encryption. It is described in section 6.2.10.9.1 of [WUSB1]. Quoted from the specification: “The 4-way handshake combines mutual authentication and temporal key distribution into a single 4-message protocol. Temporal keys are derived in a manner that does not expose the CK, thereby eliminating the need for a unique nonce for every use of the CK.”. As figure 3-2,

PTK (Pairwise Temporal Key) is derived from the handshake, it then be used by the host hardware and device hardware to encrypt data.

Solaris implementation follows the specification. It is implemented in WUSB host controller drivers and the WUSB extension to usba module.

4 RBAC Model

For administer WUSB host and device-host association, a couple of authorizations and a profile are introduced.

4.1 Authorizations

The following new authorizations are introduced:

1) `solaris.admin.wusb.:::Administer Wireless USB::help=WUSBHeader.html`

Authorizations for administer wireless USB host and host-device associations.

2) `solaris.admin.wusb.read.:::Read Wireless USB Host and Device Information::help=WUSBread.html`

Allow users to read WUSB host and device information, including connection status, types and device/host IDs. This authorization is by default granted to Basic Solaris User rights profile. So that all users by default can read the information.

3) `solaris.admin.wusb.modify.:::Add or delete information of Wireless USB Device:::help=WUSBmodify.html`

Allow users to add or delete the information of WUSB devices, including the association information, device ID, and status. By allowing users to add/delete association information, users are allowed to associate a device with a host or dissociate a device with a host. This authorization is by default granted to Console User rights profile.

With this authorization by default granted to Console User profile, the owner of /dev/console (e.g., a user who logged the system via login(1)) is granted the authorizations of associate or dissociate WUSB devices with hosts on the system. Please refer to “PSARC 2008/034 Defining Workstation Owner Infrastructure “ for details about console user.

4) `solaris.admin.wusb.host.:::Manage Wireless USB Host::help=WUSBhost.html`

Allow users to administer the WUSB hosts, including remove, disable and enable the hosts on the system.

5) `solaris.smf.manage.wusb.:::Manage Wireless USB Service::help=SmfWusbStates.html`

Allow users to enable, disable, or restart wireless USB service (`svc:/system/wusb:default`).

4.2 Rights Profile

A new profile is introduced:

```
WUSB Management:::Manage Wireless USB:auths=solaris.admin.wusb.*,
solaris.smf.manage.wusb;help=WUSEmgmt.html
```

This profile is by default granted to System Administrator rights profile.

In order to successfully invoke all of the `wusbadm` subcommands and manage `wusb` service, users must be added to WUSB Management profile (or granted the access to a role with that profile).

5 SMF service

The `wusb` daemon is managed by SMF, it follows the SMF security policy (`smf_security(5)`). A privilege model will be defined for it.

As aforementioned, the smf service name is "svc:/system/wusb:default". To meet the SMF policy, this service is delivered as follows:

- The service is delivered disabled. The service is enabled by user via svcadm(1M) when user starts to use WUSB devices for the first time; and it is disabled by user via svcadm(1M) when user decides to stop using any WUSB devices.

- The service is managed using the action_authorization "solaris.smf.manage.wusb" which is included in the WUSB Management profile.

- The service is local only.

- The service has no service specific properties, therefore no value_authorization.

- The service implements the project private /usr/lib/wusbd.

- The service implements the principle of least privilege by specifying them in the service manifest's method_context/method_credential/limit_privilege property.

Some other implementation details:

- The SMF manifest delivers as:

 - /var/svc/manifest/system/wusb.xml

- The method delivers as:

 - /lib/svc/method/svc-wusb

6 Audit

Because wusbadm uses RBAC, each wusbadm subcommand invocation will generate an audit record. The granting of all the authorizations described in section 4 are audited. The authentications cover modify association information, reading host/device information, enabling/disabling hosts and wusb service managements.

7 Boot

From end user's view, during boot, it is similar as wired USB devices plugged to the system that, associated WUSB devices can be connected with host without user interference. (Assuming the device radio is turned on while booting.) The difference is, it takes a little bit more time for a WUSB device to be connected. The main reason is, after host driver attach, wusb service must load the CCs to host drivers before host drivers can start to find and connect devices. As mentioned in section 5, wusb service is enabled at the first time when user wants to use WUSB devices. User then needs not to manually enable the service after each reboot.

Boot from a WUSB disk is not supported. WUSB specifications do not give any solutions for booting from WUSB disk, and the industry (including system BIOS vendors) does not have a solution yet. The challenging issues start from BIOS support.

8 Issues During System Installation

Unlike wired USB, WUSB devices (including keyboard and mouse) must go through an association process before it is able to connect with a host. And WUSB specifications do not give any explicit solutions about supporting devices during system installation. This brings a problem. Take WUSB keyboard as an example, during system installation, if user wants to use the WUSB keyboard before OS is loaded, system BIOS (or sparc OBP) must offer interfaces for user to associate the device with host and also offer drivers for both the WUSB host and device. However, so far, no system BIOS offers these functions. It is really a tough job to support WUSB in BIOS because of the complexity of association and host drivers.

In later stage of installation process, after the OS image is booted and drivers are attached, it is possible to pre-define some policies and make a WUSB device work (associated and connected) without user running any commands on the system first. The detail behavior is expected as: user plugs the device to the system USB port and wait a couple of seconds for the cable association process to complete, then unplug the device and turn on the device radio to connect it with host. That way, even a WUSB keyboard (as the first user input device for the system) can be associated and connected during system installation. (Note, this case only applies to devices supporting cable association model because it is a must requirement for numeric association process that user must input something to confirm the

association before the device is associated.).

As most usage scenarios for various WUSB devices are expected to be in the cases after system installation, it is out of the scope of this project to discuss the details of supporting WUSB devices during system installation. It can be another project to add the feature in the future.

From the device manufacturer's view, if a WUSB device is expected to be supported as well as the wired ones during system installation, the feasible way (and maybe the easiest way) is to implement both WUSB interfaces and wired USB interfaces on the same device. That is, take the keyboard as the example again, manufacturers can implement both wired and wireless keyboard interfaces in it. If plug it to the system, then wired USB host driver will work for the wired keyboard interface; if unplug it and turn on the device radio (assume the association is done), then WUSB host driver will work for the wireless interface on the device. In either case, the system has a keyboard available to work. It is a usual case in USB domain that one device has multiple interfaces implemented.

9 Driver IOCTLs

All the ioctl interfaces introduced in this project are project private. They are implemented by radio controller drivers, host controller drivers and cable association driver. wusbadm and wusb service call these interfaces to send device commands or get device messages.

9.1 Radio Controller Driver IOCTLs

See header file uwb.h for details.

Radio controller device implements the underlying details for PHY and MAC layers, according to [UWB1], [WUSB1] and [WHCI1]. Radio controller device drivers implement some ioctls for start/stop radio scan, start/stop beacon, get notifications, etc. wusbadm and wusb service call these interfaces to initiate and manage the radio platform related issues. End users do not aware the underlying operations. Two ioctls are introduced:

- **UWB_COMMAND**

Send a command to the device. There is a command result return from the device.

There are a couple of commands, each of them has a command code. See `uwb_rc_cmd_code_t` in `uwb.h`.

- **UWB_GET_NOTIFICATION**

Get the message (notification) from device.

There are a couple of notification types, each of them has a notification code. See `uwb_rc_notif_t` in `uwb.h`.

9.2 Host Controller Driver IOCTLs

See header file `wusb_io.h` for details.

Host controller hardware implements the WUSB protocol, according to [WUSB1] and [WHCI1]. Host controller drivers implement some ioctls for association, enable/disable host controller operations, and get host/device state, etc. All the ioctls for host controller drivers are listed as following:

- **WUSB_HC_GET_DSTATE**

Get the state of an associated device.

- **WUSB_HC_GET_MAC_ADDR**

Get the host's MAC address

- **WUSB_HC_ADD_CC**

Load a CC to host driver and update chid when CC list is null.

- WUSB_HC_REM_CC
Remove a CC from host driver.
- WUSB_HC_SET_CHANNEL
Set host beaconing channel number.
- WUSB_HC_START
Enable the host controller.
- WUSB_HC_STOP
Disable the host controller.
- WUSB_HC_START_NA
Direct the host to start to accept device connections from un-associated devices.
- WUSB_HC_STOP_NA
Direct the host to stop accepting device connections from un-associated devices.
- WUSB_HC_GET_HSTATE
Get the host controller's state.

9.3 Cable Association Driver IOCTLs

See header file `wusb_ca.h` for details.

WUSB cable association driver `wusb_ca` implements some `ioctl`s for cable association process. User runs `wusbadm associate` subcommand to start the process. All the `ioctl`s introduced are listed as following:

- WUSB_CA_IOCTL_GET ASSO_INFO
Get a list of all the association related info of the device.
- WUSB_CA_IOCTL_GET ASSO_REQS
Get the association requests from the device.
- WUSB_CA_IOCTL_SEND_HOST_INFO
Send the host association info to device.
- WUSB_CA_IOCTL_GET_DEVICE_INFO
Get the device association info.
- WUSB_CA_IOCTL_SET_CONNECTION
Mark the current association session a successful one.

- WUSB_CA_IOCTL_SET_FAILURE

Fail the current association session.

9.4 IOCTLs for Numeric Association

A couple of ioctls will be introduced for numeric association, similar as the ones for cable association, see section 9.3. The header file and detail interface definition are not available yet because, the feature for numeric association will be delivered in phase 3 of this project (see question 3 of 20q document) and there are no devices in the market supporting numeric association so far. Numeric association support is a feature in WUSB host controller drivers.