Corporate Technology Group

Communications Technology Lab

Simple Config Reference Implementation Software Design Specification

DRAFT Revision 0.4

November 29, 2005

Approval

Role	Date

Revision History

Rev.	Date	Author(s)	Reason for Changes
0.1	05/26/05	HH	Created the first draft, Simple config split from overall design doc.
0.3	10/26/05	VL	Updated to reflect new software architecture
0.4	11/29/05	HH	Added block diagrams in Appendix A.

Notice

Table of Contents

1. Introduction

1.1. Purpose

This document describes the architecture of Intel's Wi-Fi Simple Config reference implementation. The primary goals of the reference implementation are to accelerate the WFA certification process and encourage industry support for Simple Config by providing a working source code implementation of the specification. This reference implementation is not intended to be a development kit for actual products. It has not been optimized for performance code space, and it will not be subjected to rigorous testing. Improvements of this source code in these areas may be undertaken by others who are interested in developing and supporting product development tools.

1.2. Scope

This document describes the major functional blocks of the implementation and the APIs between the functional blocks.

1.3. Related Documents

Related documents and locations are listed in the table below.

Document	**Location**
Wi-Fi Simple Configuration Specification [WIFISPEC]	

1.4. Definitions and Acronyms

Unique terms, acronyms, and associated definitions are listed in the table below.

Term	**Definition**

2. Overview

The Wi-Fi Simple Config Specification [WIFISPEC] describes an architecture and set of protocols intended to simplify the security setup and management of Wi-Fi networks. The reference implementation described in this document supports all of the major features of Simple Config. Each functional component in the implementation is described along with a high-level description of the interfaces and interactions between components.

3. Design Requirements

3.1. Goals and Objectives

The purpose of this reference implementation is to provide a starting point for product development and assist in interoperability testing. The following list enumerates some sub-goals and objectives of the reference implementation.

- Support all device roles (Registrar, AP, and Enrollee) of the Simple Config spec to facilitate interoperability testing
- Provide an implementation for Linux environments
- Enable extensibility to support additional out-of-band channels
- Minimize dependencies on proprietary libraries that may be subject to licensing fees or licensing obligations

3.2. Environment

Hardware Platforms

- Personal Computer (both Desktop and Mobile)

Operating Systems

- Linux (Fedora Core 2)

Nothing in the code or licensing explicitly limits its usability to the above hardware and OS platforms, but some amount of porting will be needed to target other platforms.

3.3. Assumptions and Constraints

<TBD>

4. Simple Config Architectural Overview

This section presents a high-level description of the Simple Config architecture. Much of the material is taken directly from the Simple Config specification.

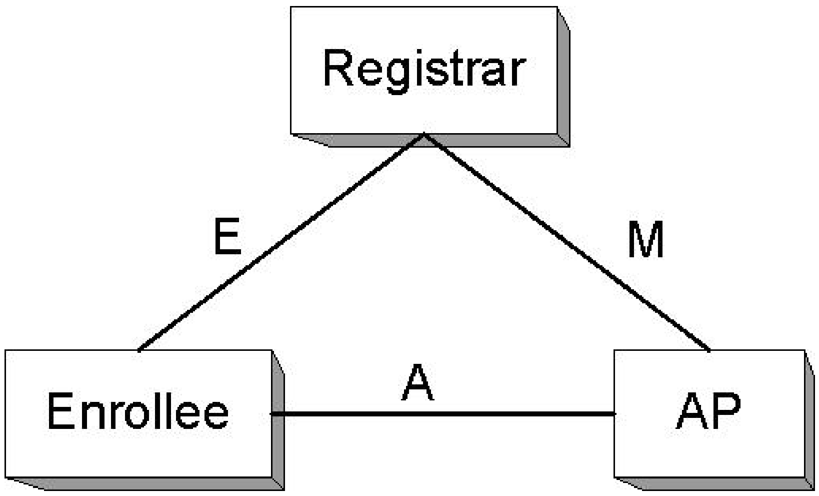


Figure 1: Components and Interfaces

Figure 1 depicts the major components and their interfaces as defined by the Wi-Fi Simple Config Spec. There are three logical components involved in Wi-Fi Simple Config: the Registrar, the AP, and the Enrollee.

4.1. Interface E

This interface is logically located between the Enrollee and the Registrar (physically, the AP can work as a proxy to convey the messages). The functionality of Interface E is to enable the Registrar to discover and issue WLAN Credentials to the Enrollee. Interface E may include only WLAN communication or it may also include communication across an out-of-band channel.

4.1.1. Enrollee

The Enrollee implements Interface E by:

1. Including a Simple Config IE in 802.11 probe messages.
2. Including a device password on a display or printed label for in-band configuration.
3. Optionally supporting one or more out-of-band configuration channels.
4. Implementing the “Device” part of the Registration Protocol.
5. Optionally receiving ad-hoc probe-responses from wireless Registrars.

4.1.2. Registrar

The Registrar implements Interface E by:

1. Processing Enrollee (device or AP) Discovery data in Probe messages (for wireless Registrars) and/or UPnP (for Ethernet-based Registrars).
2. Implementing the "Registrar" part of the Registration Protocol.
3. Optionally supporting one or more out-of-band configuration channels.
4. Configuring the AP with the Enrollee's MAC address and Credential using Interface M.
5. Optionally respond to Enrollee Probe-Requests via an ad-hoc Probe-Response.

4.2.Interface M

This interface is between the AP and the Registrar. Interface M enables an external Registrar to manage a Wi-Fi Simple Config AP.

4.2.1. AP

The AP implements Interface M by:

1. Acting as the Enrollee in the Registration Protocol. This includes sending its own Discovery message across both 802.11 and Ethernet. Support for at least three external Registrars is required.
2. Implementing the Management Interface described in the WFADevice and WFAWLANConfig Service documents. This requires the AP to be a UPnP device that includes support for the Wi-Fi Simple Config proxy service.
3. Monitoring 802.11 probe request and EAP messages from Enrollees and converting them to UPnP Event messages according to the proxy function described in the WFAWLANConfig Service document.

4.2.2. Registrar

The Registrar implements Interface M by:

1. Processing AP Discovery messages across 802.11 and/or Ethernet.
2. Receiving and processing Enrollee Discovery and Registration messages forwarded by the AP
3. Optionally receiving and processing Enrollee Discovery and Registration messages sent in ad hoc mode.
4. Implementing the Registrar side of the Registration Protocol to gain management rights over the AP or to issue WLAN credentials to Enrollees
5. Configuring the AP with the MAC address and/or per-device Credential of the Enrollee.
6. Implementing the Management Interface described in the WFADevice and WFAWLANConfig Service documents. This requires the Registrar to function as a UPnP control point.

4.3. Interface A

This interface is between the Enrollee and the AP. The function of Interface A is to enable discovery of the Simple Config WLAN and to enable communication between the Enrollee and Ethernet-only Registrars.

4.3.1. AP

The AP implements Interface A by:

1. Sending out 802.11 beacons indicating support for Simple Config and generating Probe Response messages containing a description of the AP.
2. Implementing an 802.1X authenticator and the Simple Config Registration Protocol EAP method.
3. Proxying 802.11 probe request and EAP messages between Enrollees and external Registrars as described in the WFADevice and WFAWLANConfig Service documents.

4.3.2. Enrollee

The Enrollee implements Interface A by:

1. Discovering a Simple Config AP and/or wireless external Registrar and sending it 802.11 probe requests including the Enrollee Discovery data.
2. Implementing an 802.1X supplicant and the Simple Config Registration Protocol EAP method.

5. Simple Config Software Design

The software required to implement the roles of APs, Enrollees, and Registrars is not exactly same, but there is considerable commonality. Intel's Simple Config reference design combines all three device roles into a single implementation. The software is configurable at runtime so that it can function in whatever role the user chooses (but only one role at a time). Besides maximizing code reuse, this approach also may be useful to implementers who are interested in developing APs with built-in Registrar functionality.

Figure 2 depicts the major software components in the reference implementation. Each of these software components and their interactions is described below.

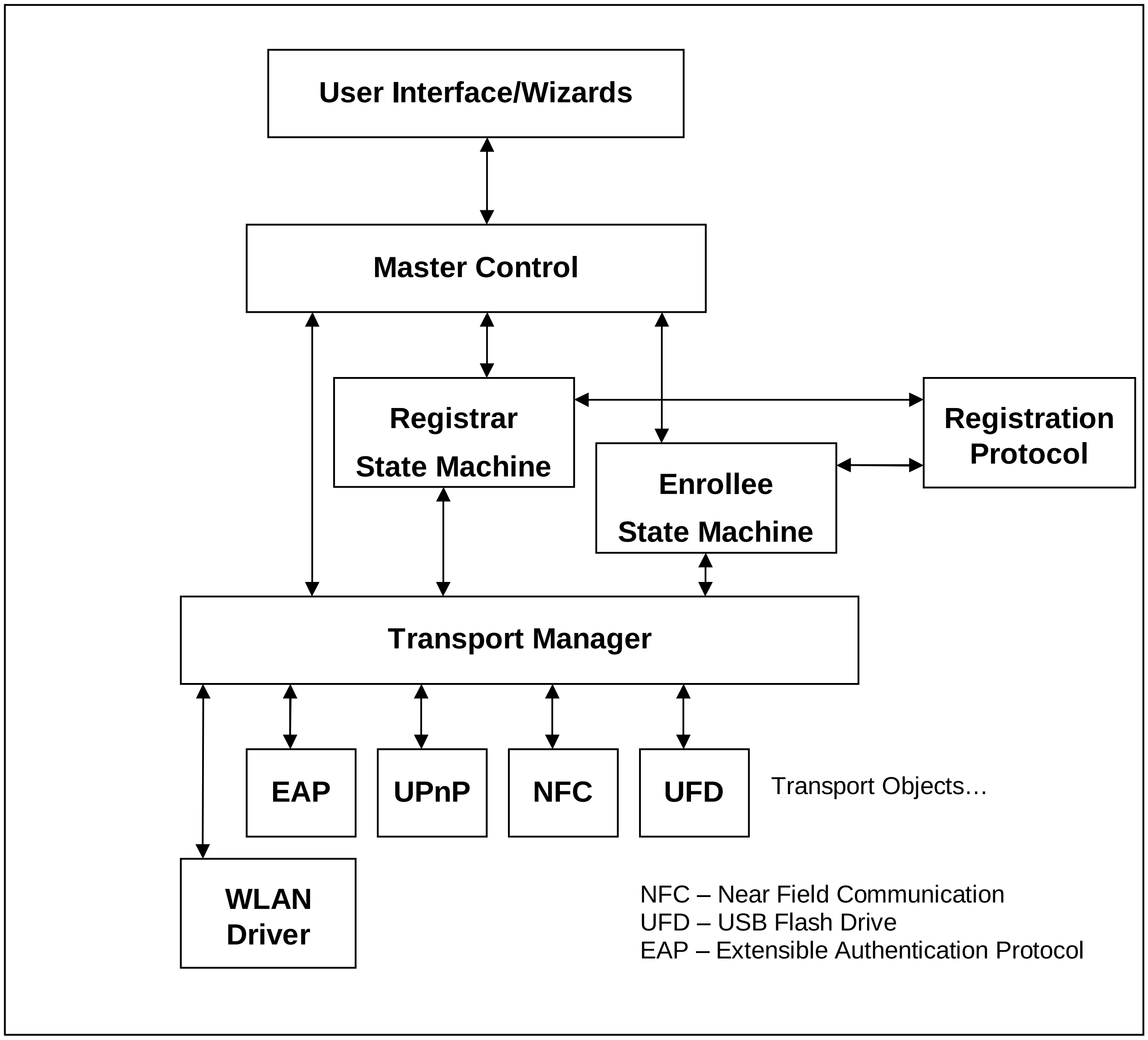


Figure 2 Functional Block Diagram

The software architecture is composed of three layers: transport, control, and UI. The transport layer is composed of the Message Transport component and a set of media-specific interface components. The control layer contains the state machines and implementation of the Simple Config protocols. The UI layer manages the user interface. Only a rudimentary UI is included in the reference implementation.

The design of the layered implementation architecture is intended to be easily extensible to support arbitrary underlying transport mechanisms. The core message processing logic is independent of the communications channel over which the messages are sent. Therefore, the Message Transport component defines a generic transport interface for sending and receiving Simple Config messages. The upper layers in the architecture can discover which transport is being used, but they typically do not need this information. Each transport can be individually enabled or disabled, according to the software configuration.

5.1. Data Processing Model

The processing flow between these components is structured as a combination of method calls and callbacks. The callbacks are asynchronous calls that enqueue a data structure corresponding to a message or task to perform and return immediately to the caller. Each callback queue is serviced by a separate thread associated with the component that owns the queue. This multithreaded design enables efficient processing of multiple concurrent instances of the Registration Protocol, which is important in contexts like rekeying, where several devices may be running the protocol at the same time.

5.2. Registration Protocol

The Registration Protocol component provides a set of functions to process Registration Protocol messages. This component also takes care of performing all cryptographic operations.

Functions for Message Building and Processing:

1. Build Message X – The Registration Protocol component includes a set of functions to build each message of the Registration Protocol. These functions are named BuildMessageM1, BuildMessageM2 and so on. The Build Message functions take the current state of the Registration Protocol instance and from that construct a message that is ready to deliver to the appropriate Transport object.
2. Process Message X – The Registration Protocol component includes a set of functions to extract each message of the Registration Protocol. These functions are named ProcessMessageM1, ProcessMessageM2 and so on. Data structures are defined for each message. The functions first deserialize the messages and then process them by setting the fields of the data structures.

5.3. Registrar/Enrollee State Machine

The Registrar State Machine and Enrollee State Machine components contain the processing logic that layers on top of the Registration Protocol message processing functions and the Message Transport. Each time a new instance of the Registration Protocol is needed, the Master Control component determines which mode the software is in (Registrar or Enrollee). Depending upon this context, the Registrar or Enrollee State Machine component is asked to create a new object that maintains the state associated with that instance of the protocol. After the state machine object is constructed, the Master Control object calls its Start() method to activate a new Registration Protocol instance.

The State Machine objects receive messages from the underlying Transport layer, process those messages using the Registration Protocol component, and send messages via the Transport layer. If user input is required, the State Machine object passes a request for input up to the Master Control layer.

5.4. Transport Manager

The Simple Config architecture is designed to support an extensible set of message transports. These transports include the in-band 802.11 channel and one or more out-of-band (OOB) channel. The Transport Manager component abstracts all of these channels and provides a unified interface to the upper layers so that the state machine components need not deal with transport-specific details.

Transport Manager Operations:

1. Init – Initialize this component
2. Deinit – Shutdown this component
3. StartMonitor – Monitor the specified transport for any status change such as link up, link down, media insertion, media removal and new data ready.
4. StopMonitor – Stop monitoring the specified transport for any status change.
5. TrRead – Read or Receive message from a specified transport.
6. TrWrite – Write or Send a message to a specified transport.
7. EnumChannels –Enumerate what transports (both OOB and in-band) are available on the device.
8. StaticCallbackProc – Callback function for notifying the Transport Manager of events or queuing work for its worker thread.

Following communication or data transfer channels are supported.

- **EAP** – 802.1X and EAP (Extensible Authentication Protocol) are used to transport registration protocol messages. There are no changes to the existing 802.1X implementation; however, new WSC EAP method is plugged into EAP framework.
- **UPnP** – UPnP (Universal Plug and Play) is used by the external registrar and wireless access point to transport registration protocol messages.
- NFC – Near Field Communications channel is used as physical OOB channel
- UFD – USB Flash Drive is used as physical OOB channel

5.5. Master Control

The Master Control component is responsible for initializing and managing the high-level operation of the software. It will run as a Windows service or Linux daemon. The upper edge of Master Control interfaces with user interface components. The lower edge interfaces with the transport and state machine layers of the software.

Master Control implements functions for the following tasks.

- Reading and updating a file containing configuration and options for the Simple Config software.
- Starting up underlying components such as the Transport Manager and State Machine component.
- Initiating and completing enrollment operations, according to the modes specified in the configuration file.
- Interacting with user interface components to obtain input and display status.
- Receiving callbacks from lower layers to process certain messages or events.

- Manipulating lists of Registrars and Enrollees.

5.6. User Interface

The User Interface component allows the user to control functions, solicits input when needed, and provides feedback regarding the state of the system and its operations. The initial User Interface component will support only a simple, command-line interface. However, the software design allows the initial UI to be easily replaced by a more sophisticated User Interface.

5.7. New IEs in WLAN Driver

802.11 Beacon, Probe Request and Probe Response messages include new information elements (IEs) for discovering Wi-Fi Simple Config support and conveying attributes such as device descriptions, public key hashes, and status information. Rather than adding knowledge of the Simple Config IEs directly to the WLAN driver layer, the Reference Implementation exposes a generic interface that allows upper-layer software to pass arbitrary IEs down to the driver. Likewise, the WLAN Driver will pass all IEs that are not already handled by the driver up to the software layers above. The communication channel used to transport these IE payloads is a sockets-based interface. Therefore, this mechanism is not appropriate for real-time probe request processing. Instead, the layers above pass down IEs that should be used by the Driver in probe requests or responses until further notice (i.e., until the IEs are removed explicitly).

6. API Reference

6.1. Module TransportBase

Filename: TransportBase.h

This module declares the abstract base class (interface) for Transport classes. Each transport object implements functions for startup/shutdown, monitoring its particular media, and for data read and write operations. It also sends asynchronous events to a callback message queue serviced by the Transport Manger module.

6.1.1. CTransportBase Class

class CTransportBase

Pure virtual base class for Transport objects

Defined in: TransportBase.h

Class Members

protected

S_CALLBACK_INFO m_trCallbackInfo

Used for callbacks to the Transport Manager object. S_CALLBACK_INFO contains a callback method pointer and a "cookie" that is passed back to the callback method. The cookie is typically used to pass an object pointer so that a static callback method can be dispatched to an instance of the class receiving the callback.

bool m_initialized

Indicates if this Transport object has been initialized.

public

CTransportBase() : m_initialized(false)

Constructor.

virtual ~CTransportBase()

Destructor. Calls Deinit() if it has not already been called.

uint32 SetTrCallback(IN CALLBACK_FN p_trCallbackFn, IN void* cookie)

Sets the callback pointer and cookie in m_trCallbackInfo.

virtual uint32 Init()

Initializes the transport object, preparing it for use. This method should be called immediately after the Constructor.

virtual uint32 StartMonitor()

Causes the transport object to begin monitoring its media. StartMonitor and StopMonitor turn on and off, respectively, incoming data and connection notifications that are passed up to the Transport manager via the callback.

virtual uint32 StopMonitor()

Causes the transport object to stop monitoring its media. StartMonitor and StopMonitor turn on and off, respectively, incoming data and connection notifications that are passed up to the Transport manager via the callback.

virtual uint32 WriteData(char * dataBuffer, uint32 dataLen)

Sends data to the transport media. It is permitted to call WriteData even after StopMonitor has been called.

virtual uint32 ReadData(char * dataBuffer, uint32 * dataLen)

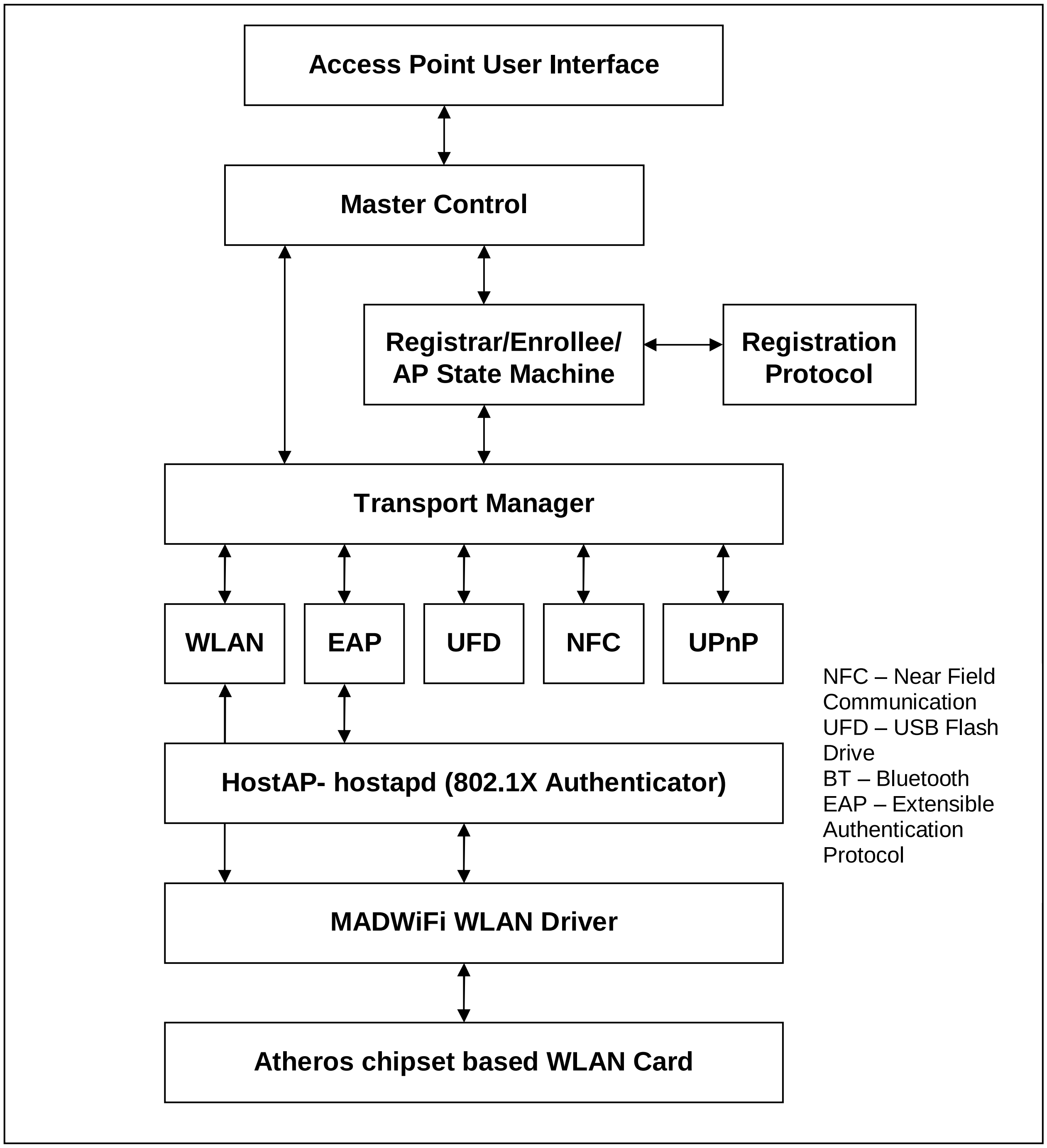
Blocking read on data from the transport media. This function is not used much, because most transports deliver incoming data via an asynchronous callback to the Transport manager. It is permitted to call ReadData even after StopMonitor has been called.

virtual uint32 Deinit()

Shuts down the transport object. This method is automatically called by the Destructor if it has not already been called.

7. Appendix A: Block Diagrams of Linux Access Point and Linux Enrollee/Registrar

7.1. Block Diagram of a Linux Access Point Implementation



Access Point User Interface
Master Control
Registrar/Enrollee/ AP State Machine
Registration Protocol
Transport Manager
WLAN
EAP
UFD
NFC
UPnP
HostAP- hostapd (802.1X Authenticator)
MADWiFi WLAN Driver
Atheros chipset based WLAN Card
NFC – Near Field Communication
UFD – USB Flash Drive
BT – Bluetooth
EAP – Extensible Authentication Protocol

Figure 1. Linux Access Point with HostAP-hostapd and MADWiFi Driver

7.2.Block Diagram of a Linux Station (Enrollee/Registrar) Implementation

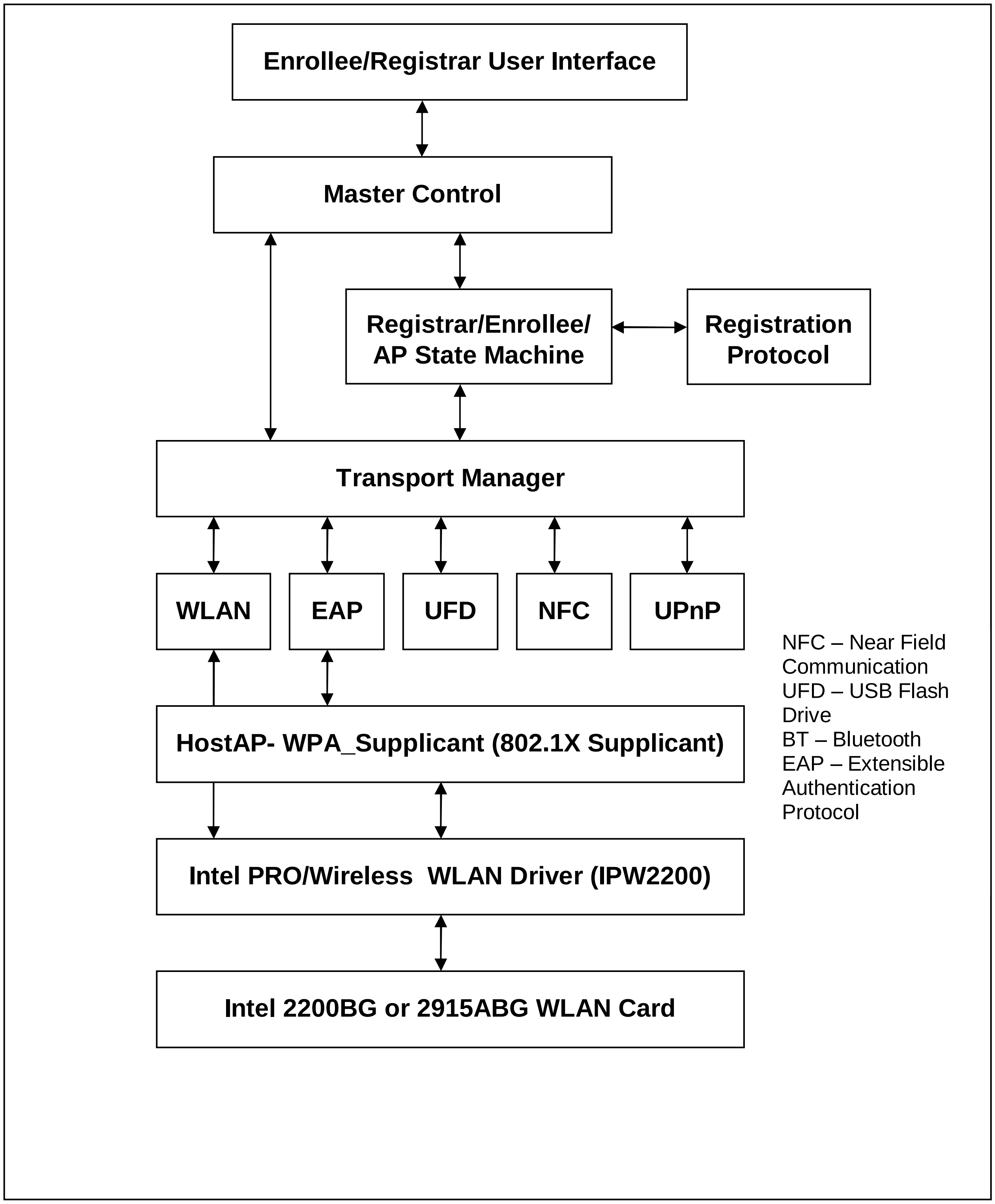

Enrollee/Registrar User Interface
Master Control
Registrar/Enrollee/ AP State Machine
Registration Protocol
Transport Manager
WLAN
EAP
UFD
NFC
UPnP
NFC – Near Field Communication
UFD – USB Flash Drive
BT – Bluetooth
EAP – Extensible Authentication Protocol
HostAP- WPA_Supplicant (802.1X Supplicant)
Intel PRO/Wireless WLAN Driver (IPW2200)
Intel 2200BG or 2915ABG WLAN Card

Figure 2. Linux Enrollee or Registrar with HostAP-WPA_Supplicant and IPW2200 Driver