Creating Android Applications for Today’s Bluetooth Devices

First Half

Agenda

- Bluetooth Technology Evolution
- Architectural Overview
- Stack Architecture
  - Physical Layer
  - Link Layer
  - HCI Layer
  - L2CAP Layer
  - Security Manager Protocol
  - Attribute Protocol
  - Generic Attribute Profile
  - Generic Access Profile
  - Applications
- Air Interface Packet Structure

Agenda

- Bluetooth Technology Evolution
- Architectural Overview
- Stack Architecture
  - Physical Layer
  - Link Layer
  - HCI Layer
  - L2CAP Layer
  - Security Manager Protocol
  - Attribute Protocol
  - Generic Attribute Profile
  - Generic Access Profile
  - Applications
- Air Interface Packet Structure

Bluetooth Technology Evolution

What is Bluetooth Technology

- Bluetooth wireless technology
  - short-range communications system intended to replace the cable(s)
  - Operates in 2.4 GHz ISM band
  - Uses FHSS to combat interference
  - Range up to 100 m based on the radio class
  - No line of sight required
- Two Forms
  - BR/EDR – Basic Rate/Enhanced Data Rate
  - Low Energy(LE)
- Both forms include
  - device discovery
  - Connection establishment and management
  - Data Transfer

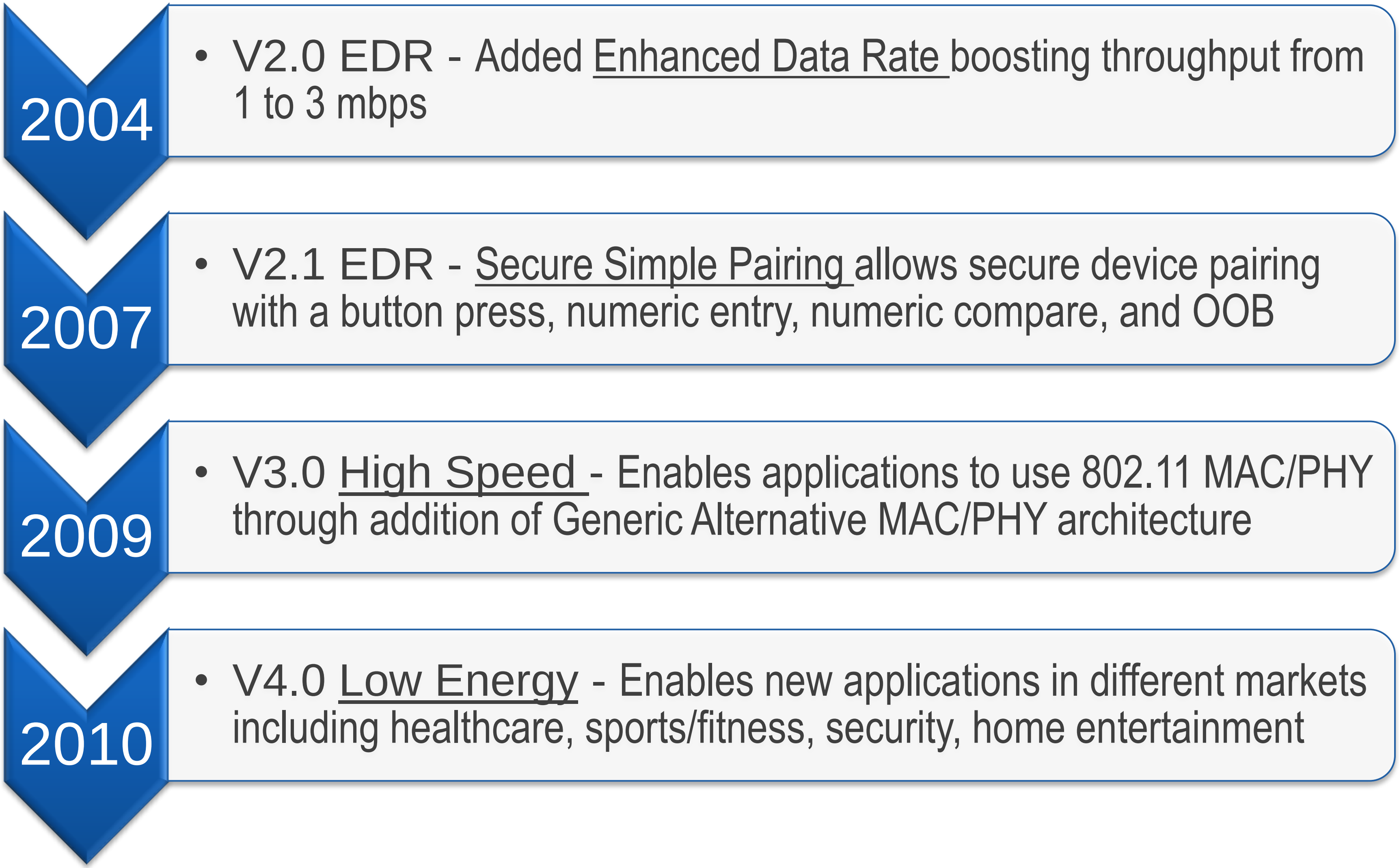
Basic Rate

- SCO & ACL Data
- Data Rates
  - 1 Mbps – FSK
  - 3 Mbps – QPSK
  - 24 Mbps, 802.11 AMP (High Speed)
- Basic Rate extensions
  - Optional Enhanced Data Rate (EDR)
  - Alternate Media Access Control (MAC)

Bluetooth Low Energy

- Focus on ultra-low power consumption
  - Ideal for devices with very low battery capacity
- Fast connections
- Efficient discovery / connection procedures
- very short packets
- client server architecture
- Everything optimized for power consumption
- Designed for use cases that require low data rates

Bluetooth Technology Evolution



2004

- V2.0 EDR - Added Enhanced Data Rate boosting throughput from 1 to 3 mbps

2007

- V2.1 EDR - Secure Simple Pairing allows secure device pairing with a button press, numeric entry, numeric compare, and OOB

2009

- V3.0 High Speed - Enables applications to use 802.11 MAC/PHY through addition of Generic Alternative MAC/PHY architecture

2010

- V4.0 Low Energy - Enables new applications in different markets including healthcare, sports/fitness, security, home entertainment

Bluetooth Ecosystem Today

Bluetooth®
SMART READY

Droid RAZR

Windows 8

iPhone 4s

Mac Mini

MacBook Air

Bluetooth®
SMART

Bluetooth®

4+ Billion *Bluetooth* products today

Bluetooth Technology – What's Next

Apps in *Bluetooth* Smart Ready devices or Cloud turn data into information

SMART READY

Billions of *Bluetooth* Smart Ready devices turning data into information through apps

Mobile Phones

Tablets

Personal Computers

Connected Televisions

Securely Sending Data to applications

SMART

Billions of *Bluetooth* Smart & *Bluetooth* Classic devices securely sending data

Health & Fitness

Automotive

Consumer Electronics

Smart Home/Energy

Better with Bluetooth

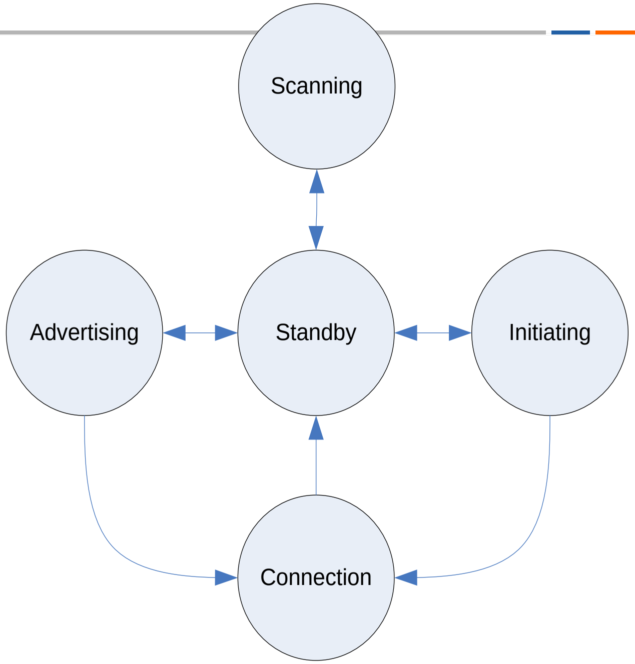
Agenda

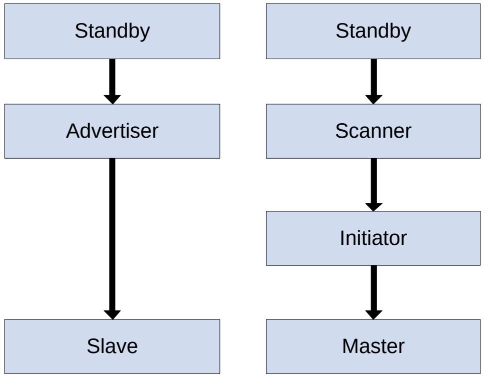
- Bluetooth Technology Evolution
- Architectural Overview
- Stack Architecture
  - Physical Layer
  - Link Layer
  - HCI Layer
  - L2CAP Layer
  - Security Manager Protocol
  - Attribute Protocol
  - Generic Attribute Profile
  - Generic Access Profile
  - Applications
- Air Interface Packet Structure

Operating States and Roles

State		State Description
Standby		Does not transmit or receive packets
Advertising		Broadcasts advertisements in advertising channels
Scanning		Looks for advertisers
Initiating		Initiates connection to advertiser
Connection	Master Role	Communicates with device in the Slave role, defines timings of transmissions
	Slave Role	Communicates with single device in Master Role

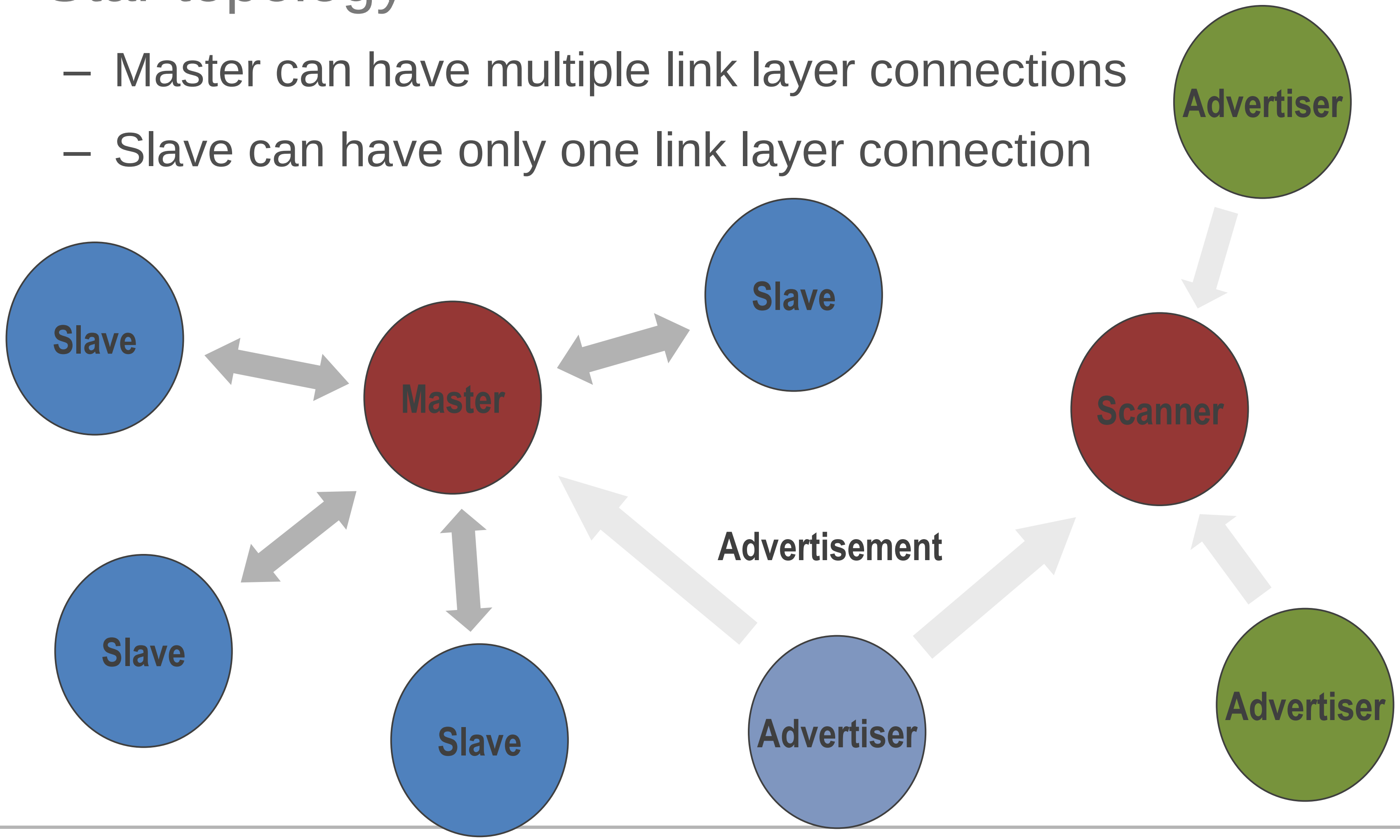
- Master/Slave only
  - No scatter net
  - No role switches





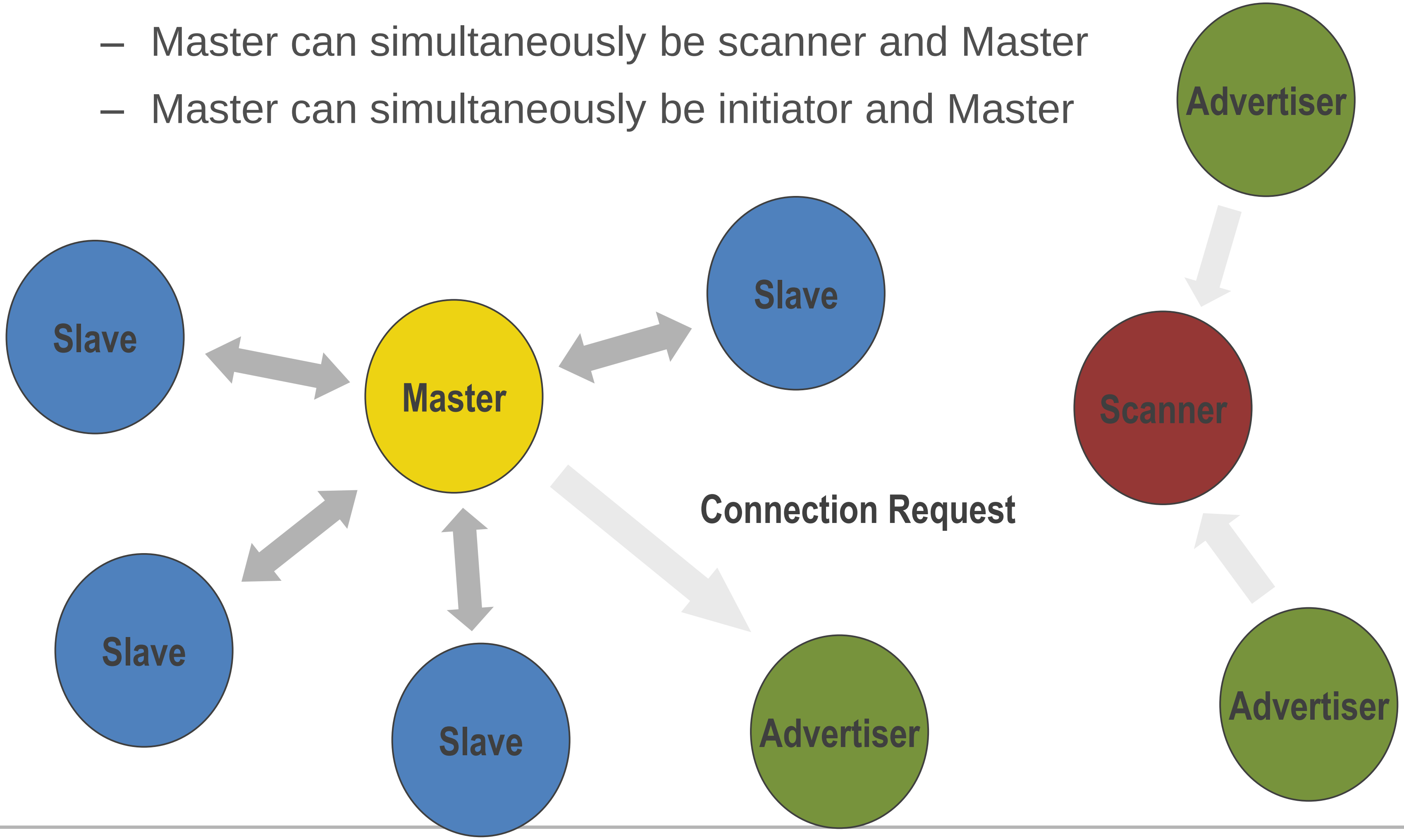
Topology Example

- Star topology
  - Master can have multiple link layer connections
  - Slave can have only one link layer connection



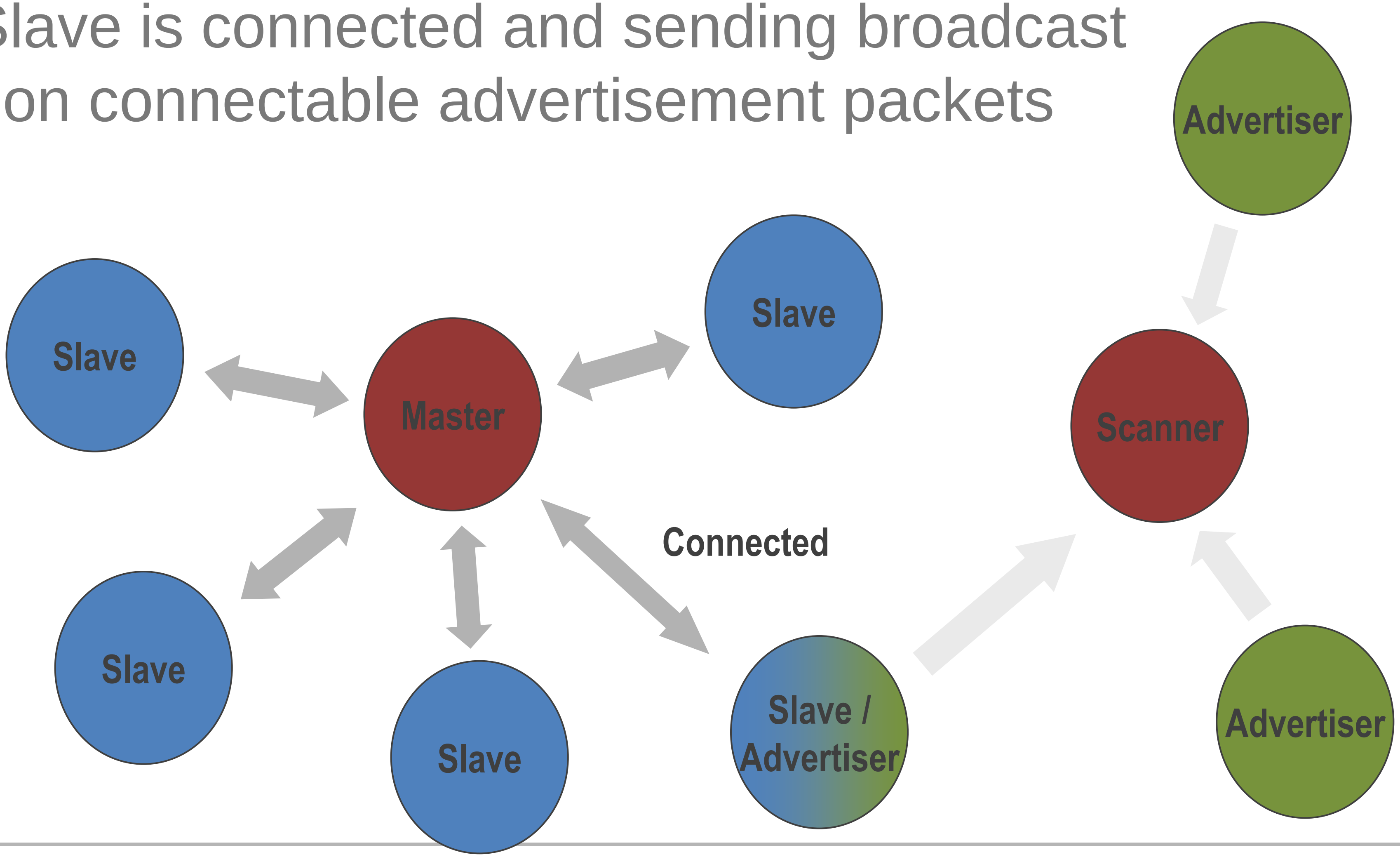
Topology Example

- Initiation of connection requests
  - Master can simultaneously be scanner and Master
  - Master can simultaneously be initiator and Master



Topology Example

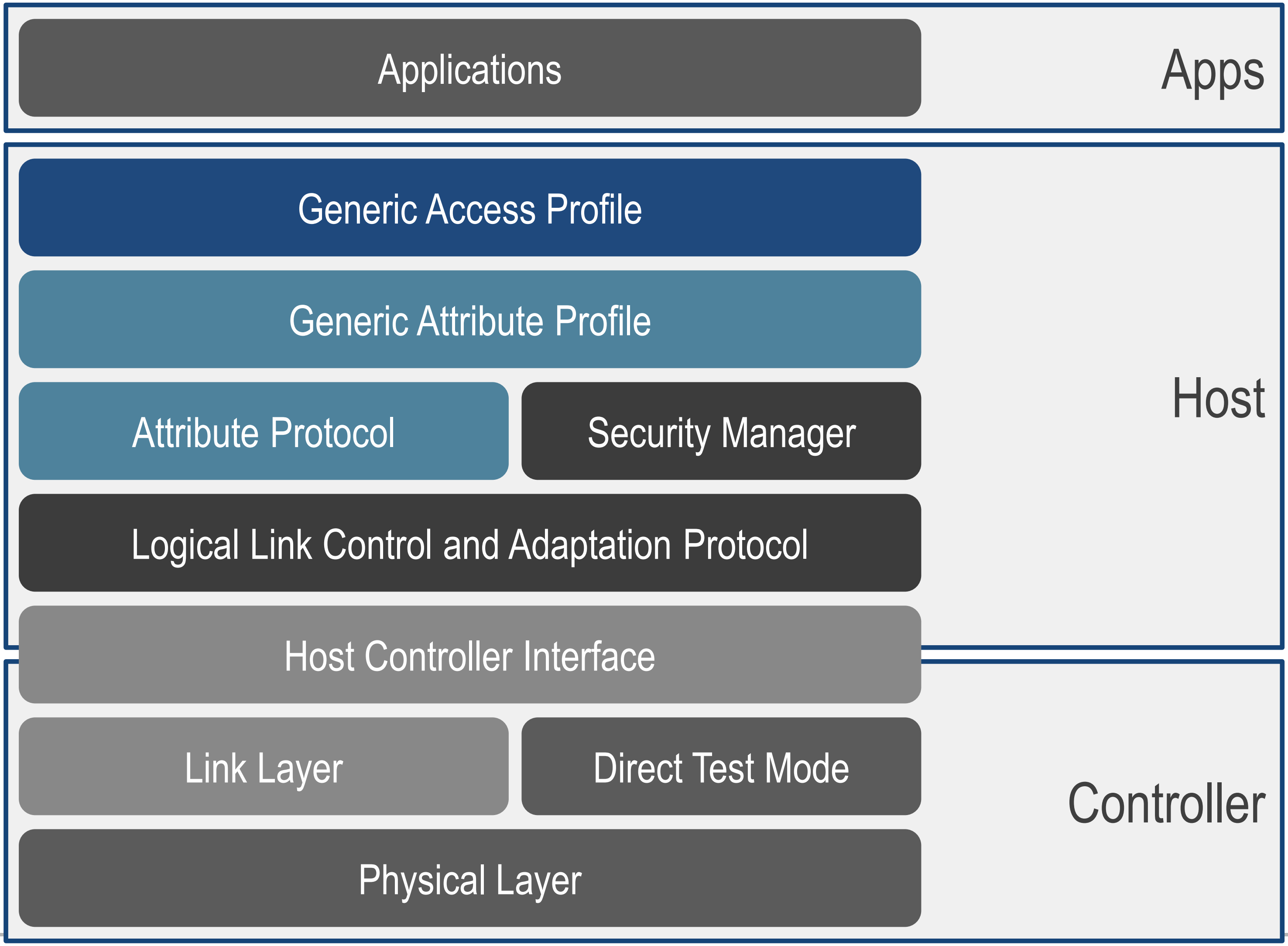
Slave is connected and sending broadcast non connectable advertisement packets



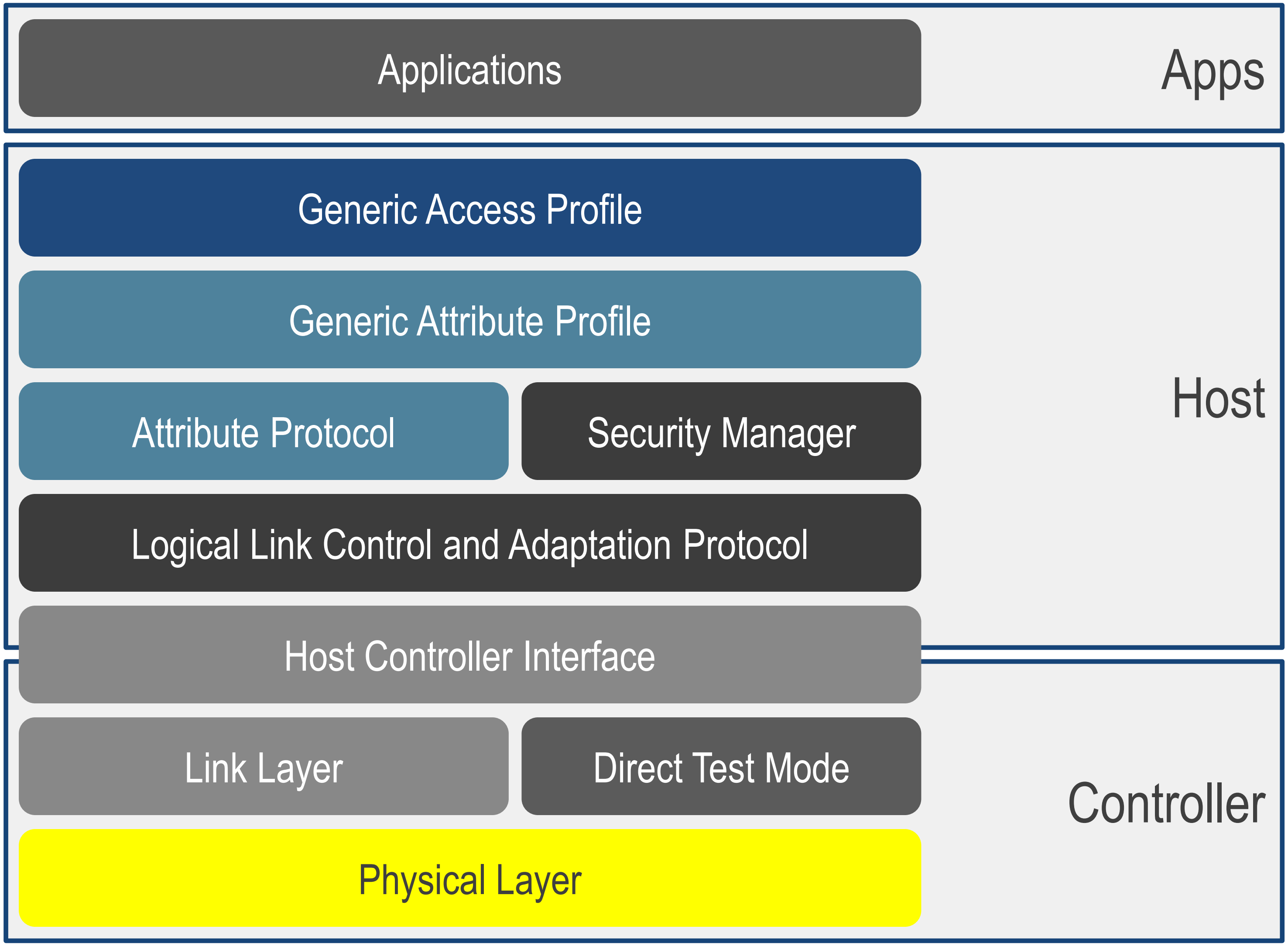
Agenda

- Bluetooth Technology Evolution
- Architectural Overview
- Stack Architecture
  - Physical Layer
  - Link Layer
  - HCI Layer
  - L2CAP Layer
  - Security Manager Protocol
  - Attribute Protocol
  - Generic Attribute Profile
  - Generic Access Profile
  - Applications
- Air Interface Packet Structure

Stack Architecture

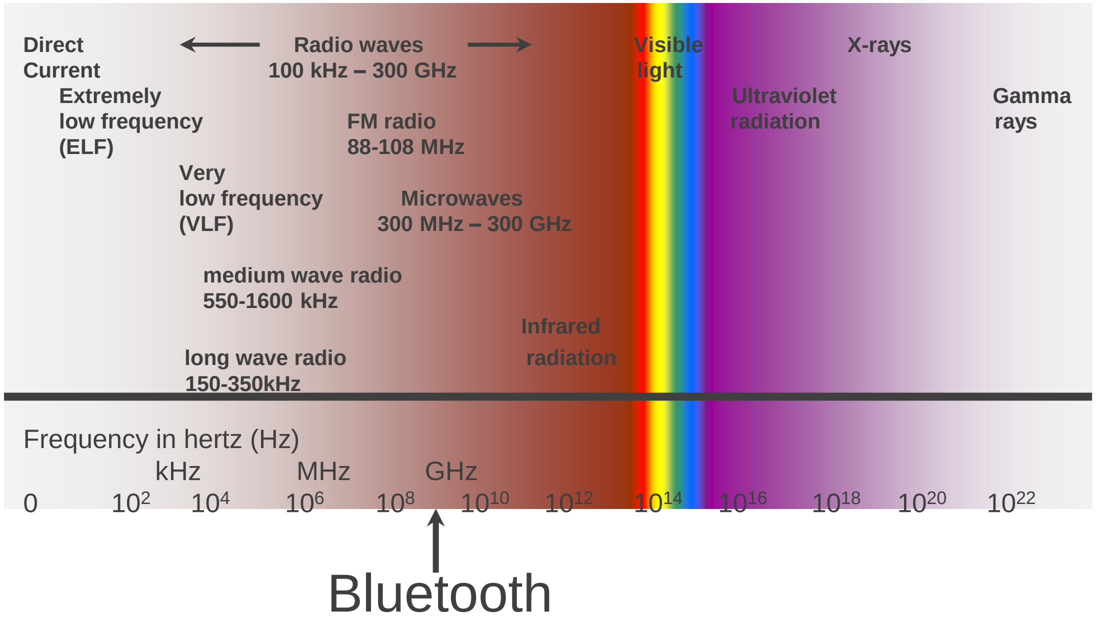


Stack Architecture



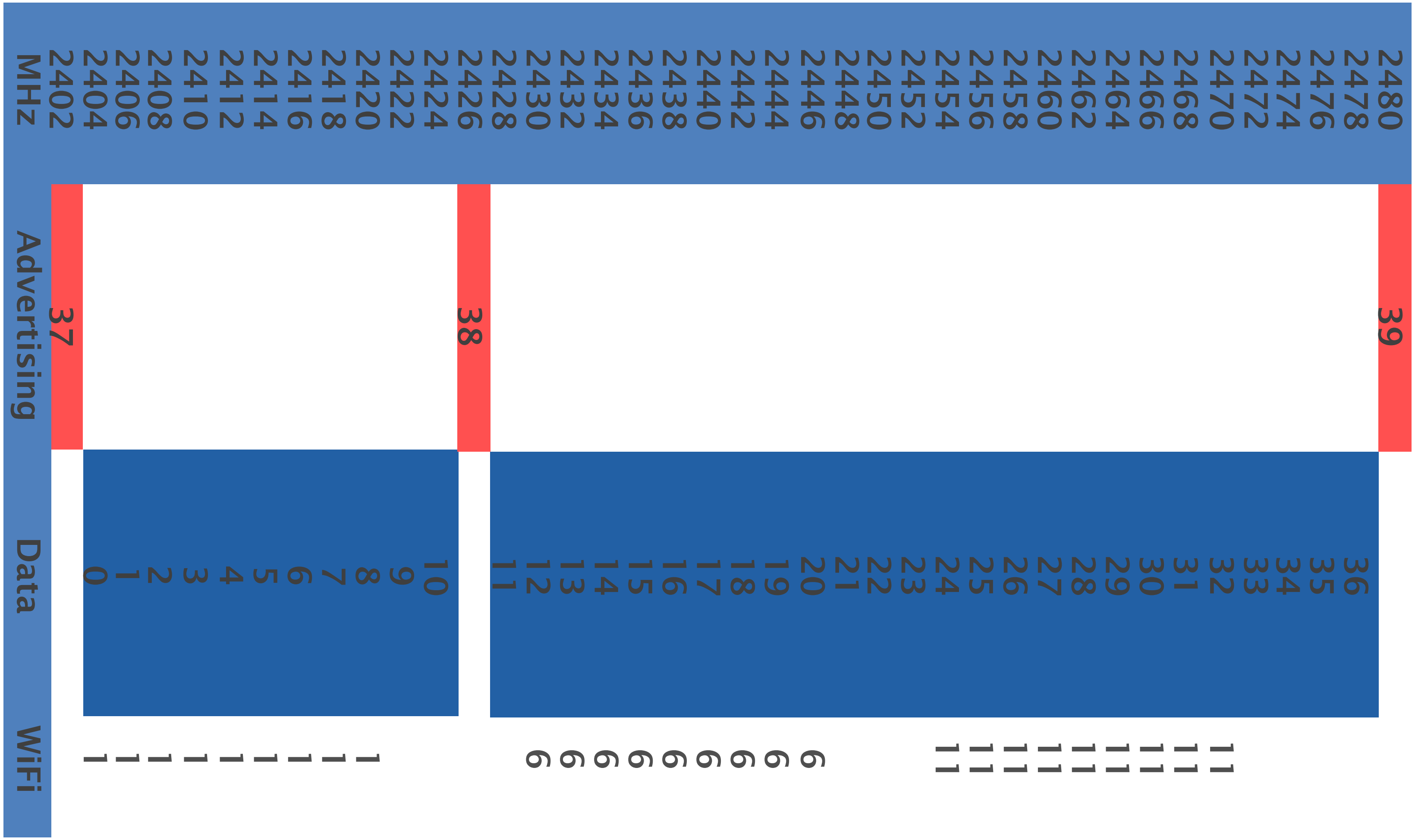
Spectrum Usage

- The 2.4GHz ISM band is a free for all for anyone who wants to use it.



- The 2.4GHz ISM Band is also used by:
  - Microwave Ovens
  - Digital Cordless Phones
  - 802.11b/g

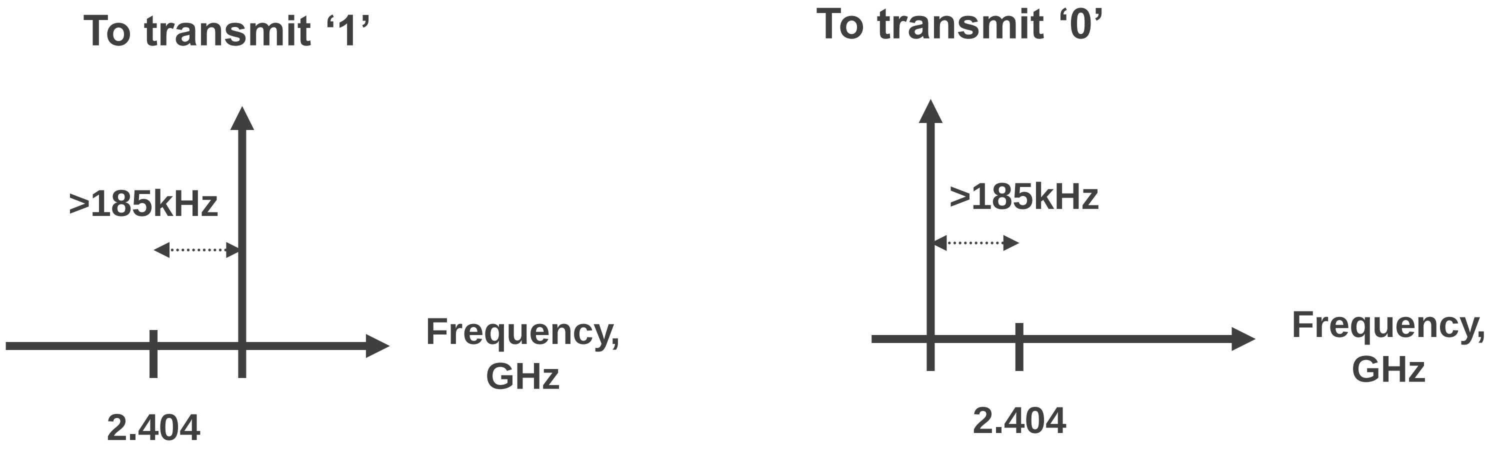
Bluetooth low energy Frequency Plan



Lower guard band of 2MHz, upper guard band of 3.5MHz

Modulation Scheme

- Data is transmitted using Gaussian Frequency Shift Keying, GFSK
- For channel 0 (Frequency 2.402GHz)



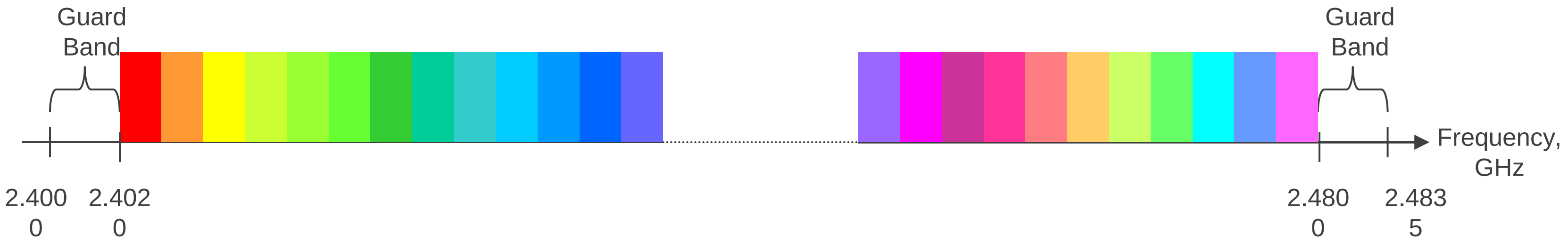
During one time slot the data can change value every 1µs, so the transmit frequency oscillates back and forth around the center channel frequency.

Frequency Hopping Spread Spectrum - FHSS

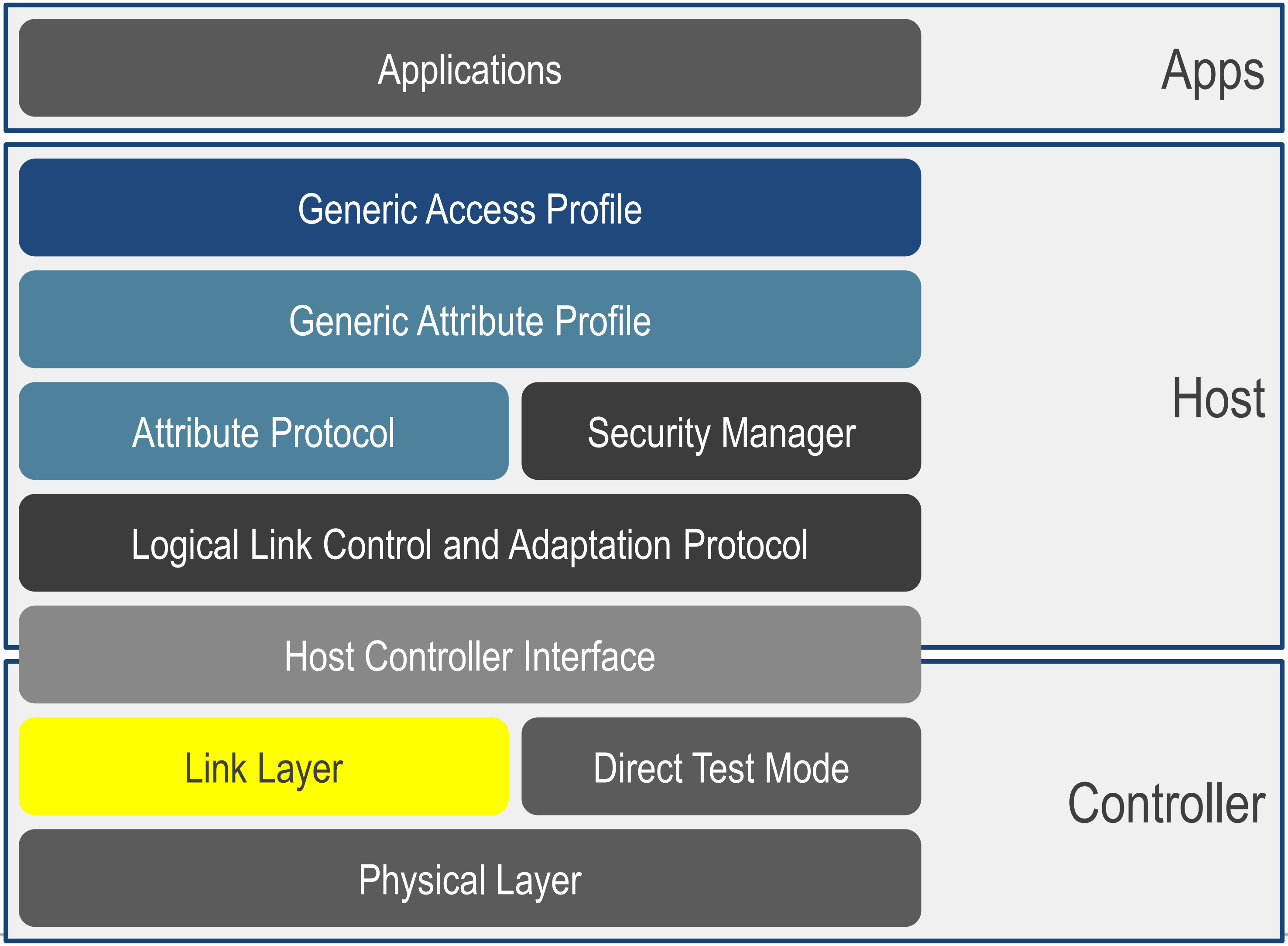
- Bluetooth low energy splits the spectrum up into 37 2MHz wide channels data channels

- FHSS occurs while in a connection

- The frequency hops follow a hop-length that is pseudo-random per connection
  - Communicated in the “Connection Request”
  - Provides instant adaptive frequency hopping capability
  - Can be updated using a channel update message



Stack Architecture



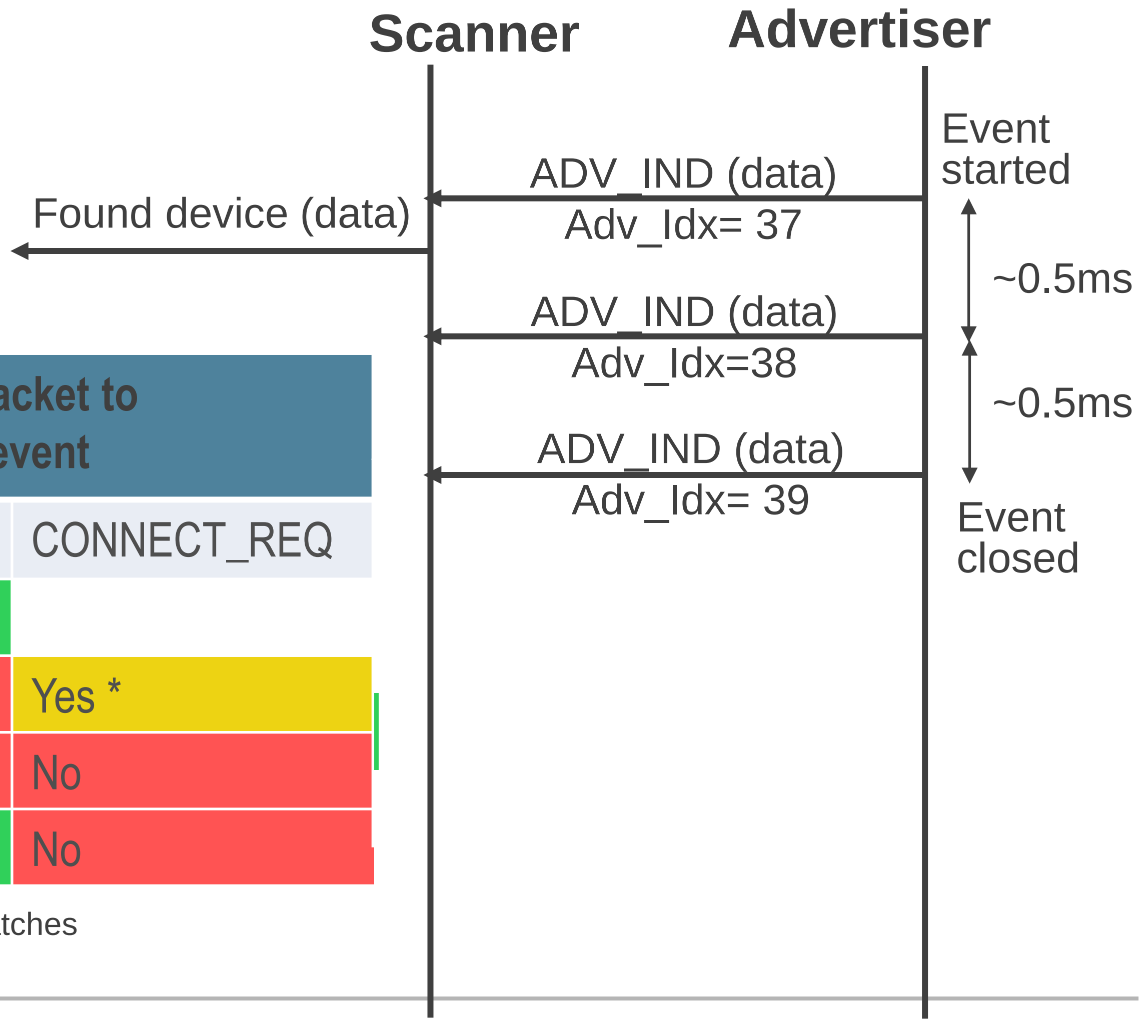
Physical Channels

- ISM band split into 40 channels of two types
  - Advertising Channels
  - Data channels

- Advertising Channels
  - Frequencies
    - 2402 (37), 2426 (38), 2480 (39)
  - Usage
    - Discovering devices
    - Initiating a connection
    - Broadcasting data

- Data Channels
  - Frequencies
    - 2404-2424 (0-10), 2428-2478 (11-36)
  - Usage
    - Communicating between connected devices

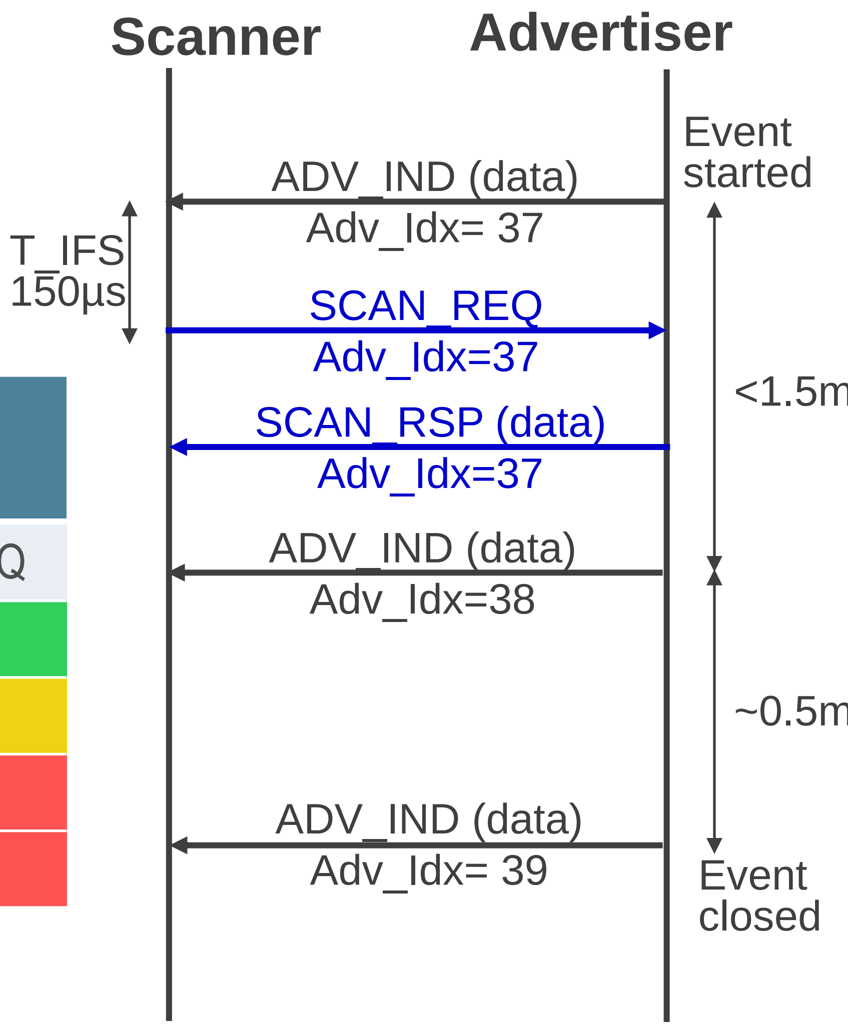
Advertising



Advertising Event	Response packet to advertising event	
	SCAN_REQ	CONNECT_REQ
ADV_IND	Yes	
ADV_DIRECT_IND	No	Yes *
ADV_NONCONN_IND	No	No
ADV_DISCOVER_IND	Yes	No

* If initiator address matches

Active Scanning



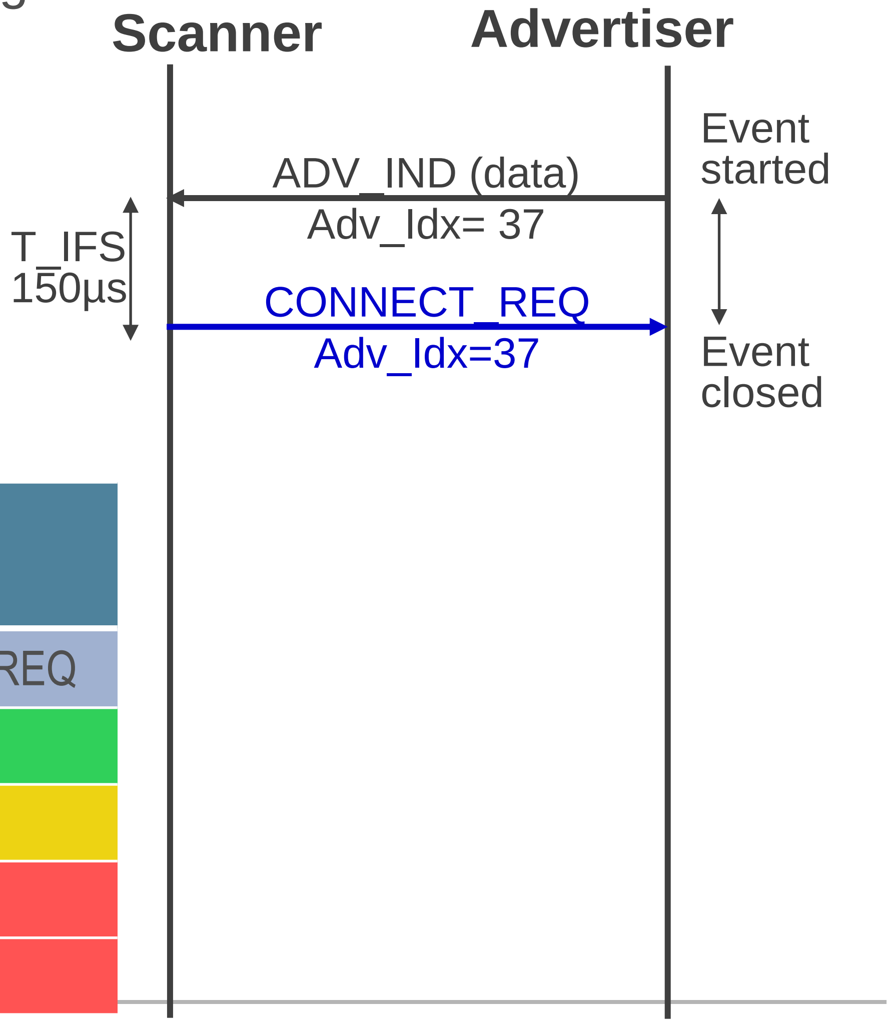
Advertising Event	Response packet to advertising event	
	SCAN_REQ	CONNECT_REQ
ADV_IND	Yes	Yes
ADV_DIRECT_IND	No	Yes *
ADV_NONCONN_IND	No	No
ADV_DISCOVER_IND	Yes	No

* If initiator address matches

Connection

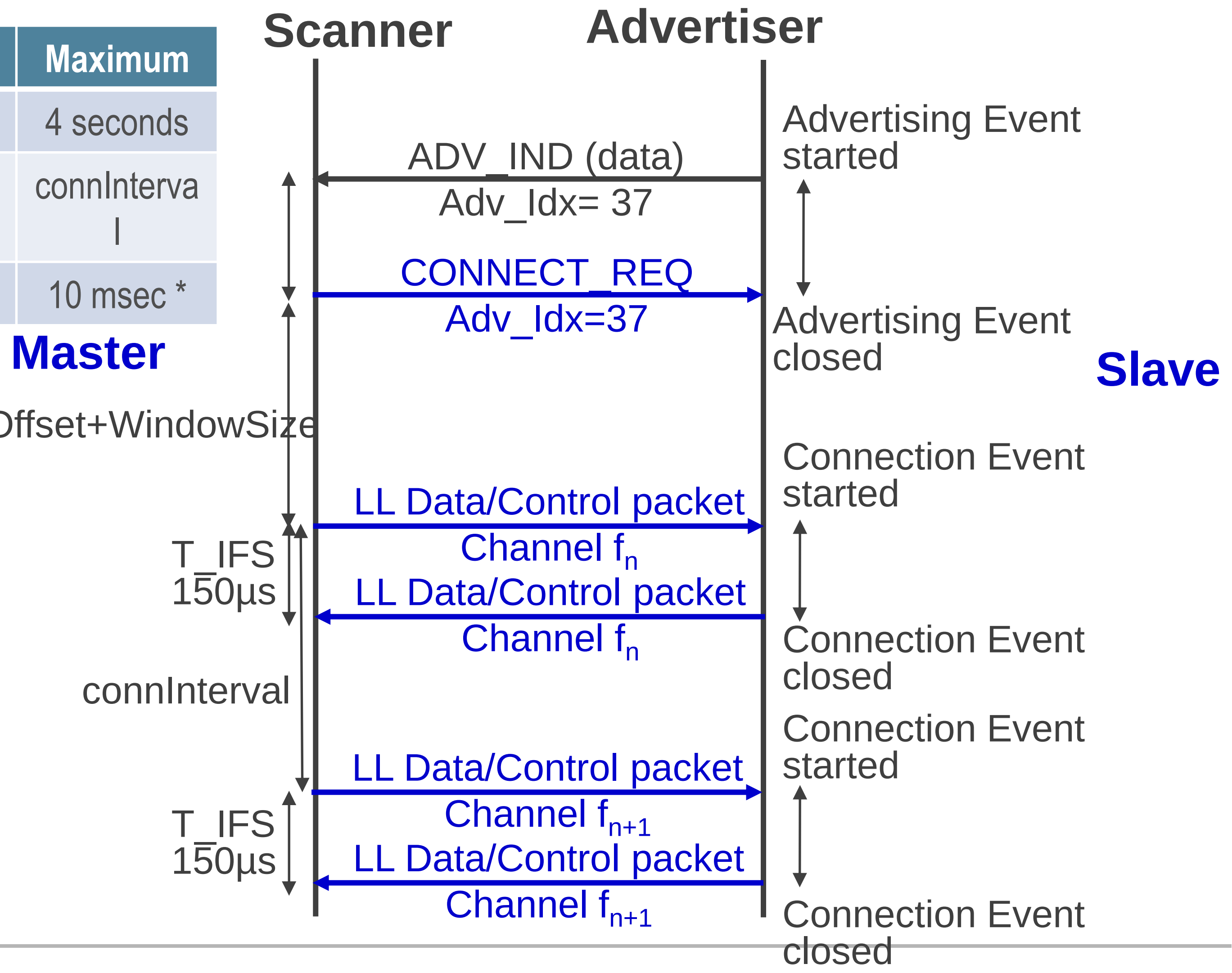
- CONNECT_REQ includes the following data:
  - Transmit window size
  - Transmit window offset
  - Connection interval
  - Slave latency
  - Connection Timeout
  - Hop sequence
  - Channel Map

Advertising Event	Response packet to advertising event	
	SCAN_REQ	CONNECT_REQ
ADV_IND	Yes	Yes
ADV_DIRECT_IND	No	Yes *
ADV_NONCONN_IND	No	No
ADV_DISCOVER_IND	Yes	No



Connection

Parameter	Minimum	Maximum
connInterval	7.5 msec	4 seconds
WindowOffset	0	connInterval
WindowSize	1.25 msec	10 msec *

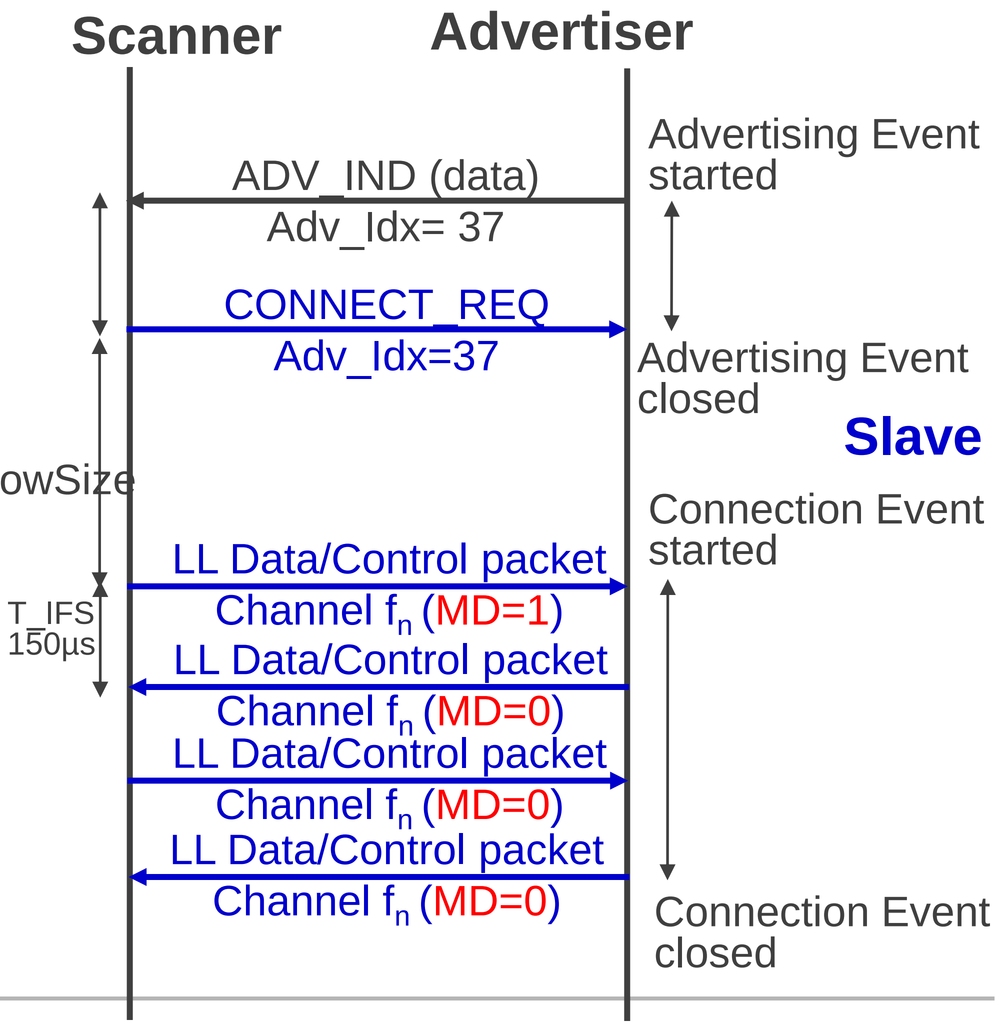


Connection using More Data (MD)

Parameter	Minimum	Maximum
connInterval	7.5 msec	4 seconds
WindowOffset	0	connInterval
WindowSize	1.25 msec	10 msec *

Master

1.25ms < t < WindowOffset+WindowSize



		Master	
		MD=0	MD=1
Slave	MD=0	Master closes connection	Master transmits Slave listens
	MD=1	Master transmits Slave listens	Master transmits Slave listens

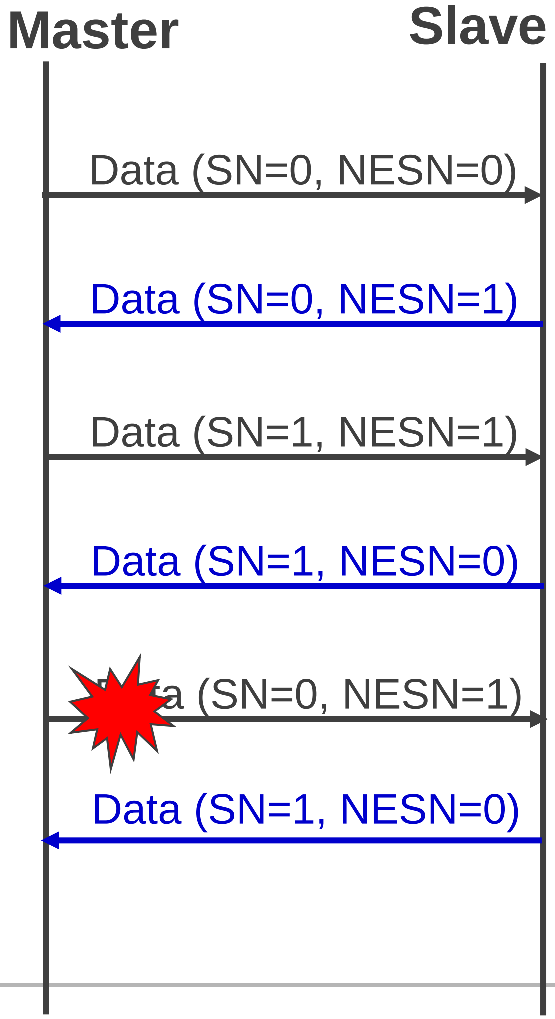
Connection Termination

- Master initiated termination - transmit TERMINATE_IND packets to slave until:
  - Acknowledgement from Slave
  - Slave latency + 6 connection events
- Slave initiated termination – transmit TERMINATE_IND packets to master until
  - Acknowledgement from Master
  - 6 connection events
- Connection supervision timeout

Acknowledgement and Flow Control

- Acknowledgements embedded in header of every Data channel PDU
  - Single bit Sequence Number (SN)
  - Single bit Next Expected Sequence Number (NESN)

- Packet is retransmitted until the NESN is different from the SN value in the sent packet
  - Enables lazy acknowledgement for significant power savings



Bluetooth low energy addresses

- Device addresses
  - Public
  - 48-bit address obtained from IEEE Registration authority
  - BD_ADDR in dual-mode devices

company_assigned [0:23]	company_id [24:47]

  - Private
  - Optional for Bluetooth low energy devices
  - Changes frequently – enables privacy

hash [0:23]	random [24:47]

- Access addresses
  - 32-bit pseudo random access address
  - Changes with each link layer connection

Device Filtering

- Devices maintain a “white list”
  - Storage of device addresses for device filtering

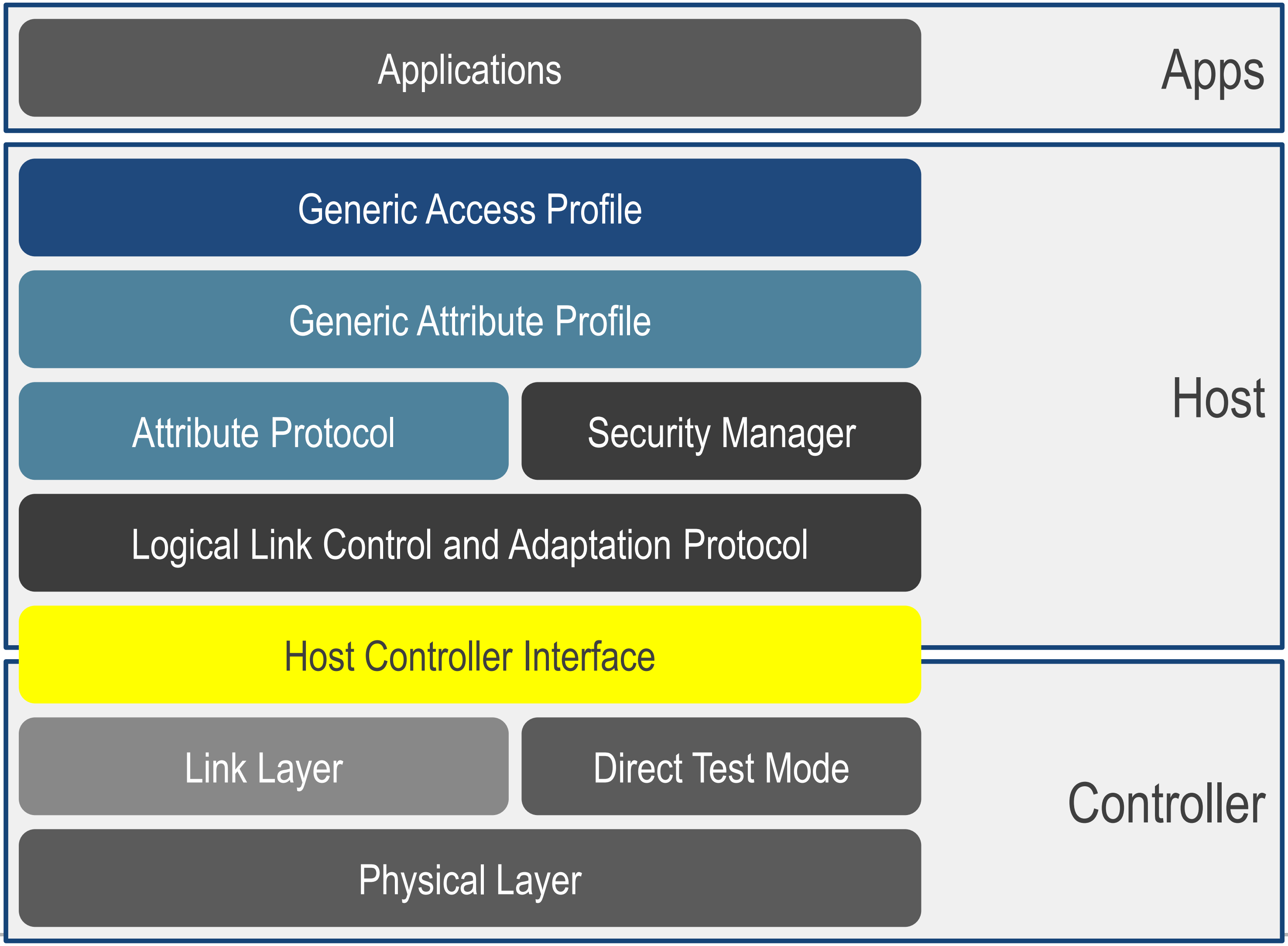
- Filter policy can be set to:
  - Advertiser
    - Process scan/connection requests from devices in white list
    - Process all scan/connection requests (default advertiser filter policy)
    - Process connection requests from all devices but only scan requests in white list
    - Process scan requests from all devices but only connection requests in white list
  - Scanner
    - Process advertising packets from devices in white list
    - Process all advertising packets (default scanner filter policy)
  - Initiator
    - Process connectable advertising events from devices in white list
    - Process connectable events only from single device specified by host

White Lists

- No need to send every advertising packet to Host
  - only send information from devices in white list

- Allows “connect to white list” semantics
  - a master can automatically connect to a set of devices
  - will connect when sees adverts from these devices
  - allows very fast connections from many devices

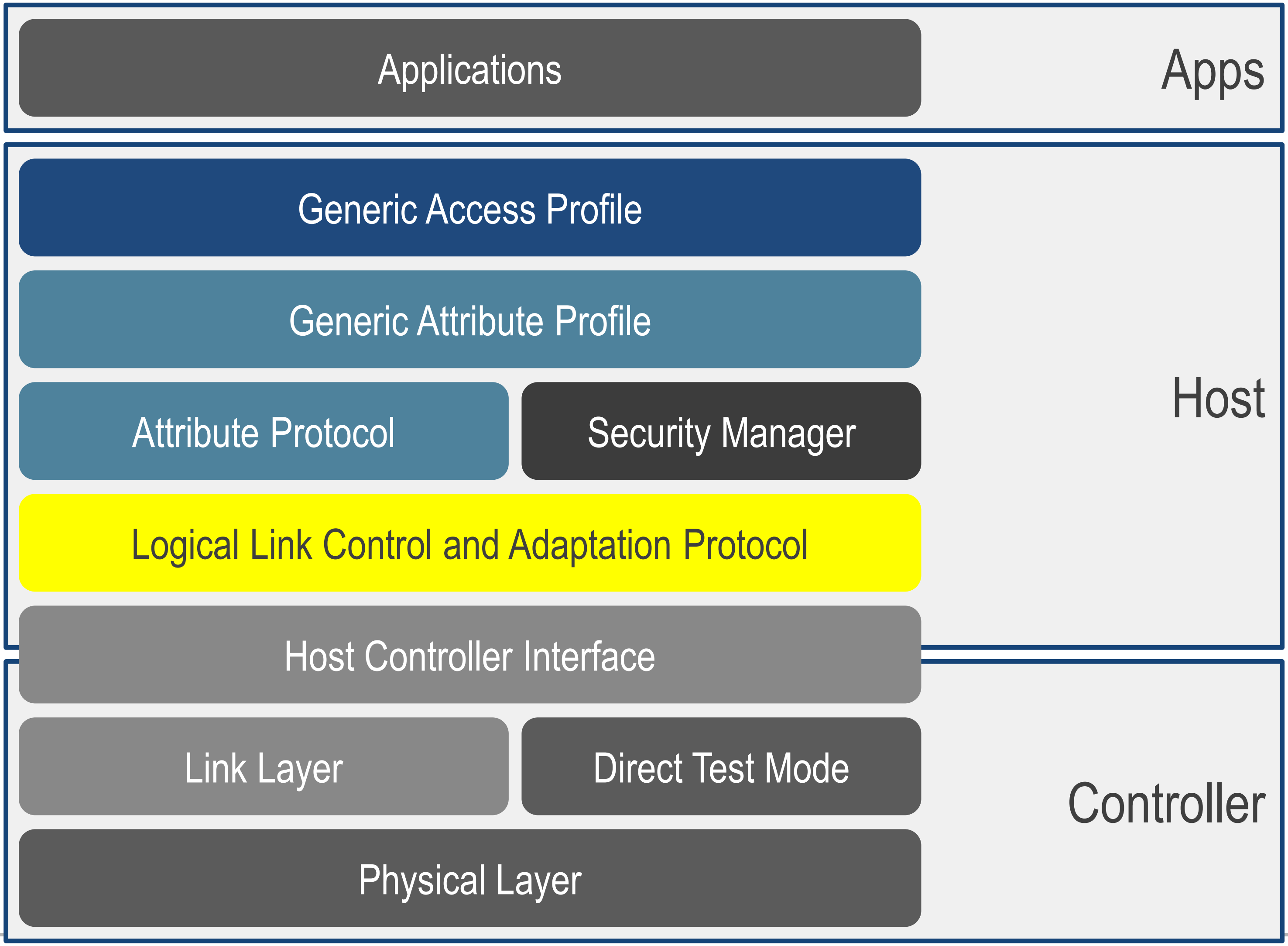
Stack Architecture



HCI Commands and Events

- Defines physical connection between a host (e.g. PC) and a host controller (e.g. Bluetooth module)
  - Interaces UART, USB, SD, 3-wire UART
  - It also defines messages that are passed across the HCI interface.
    - Controller Commands & Events
    - Host Flow Control
    - Device Setup
    - Device Discovery
    - Connection Setup and State
    - Remote Information
    - Link Information - (RSSI, Channel maps)

Stack Architecture

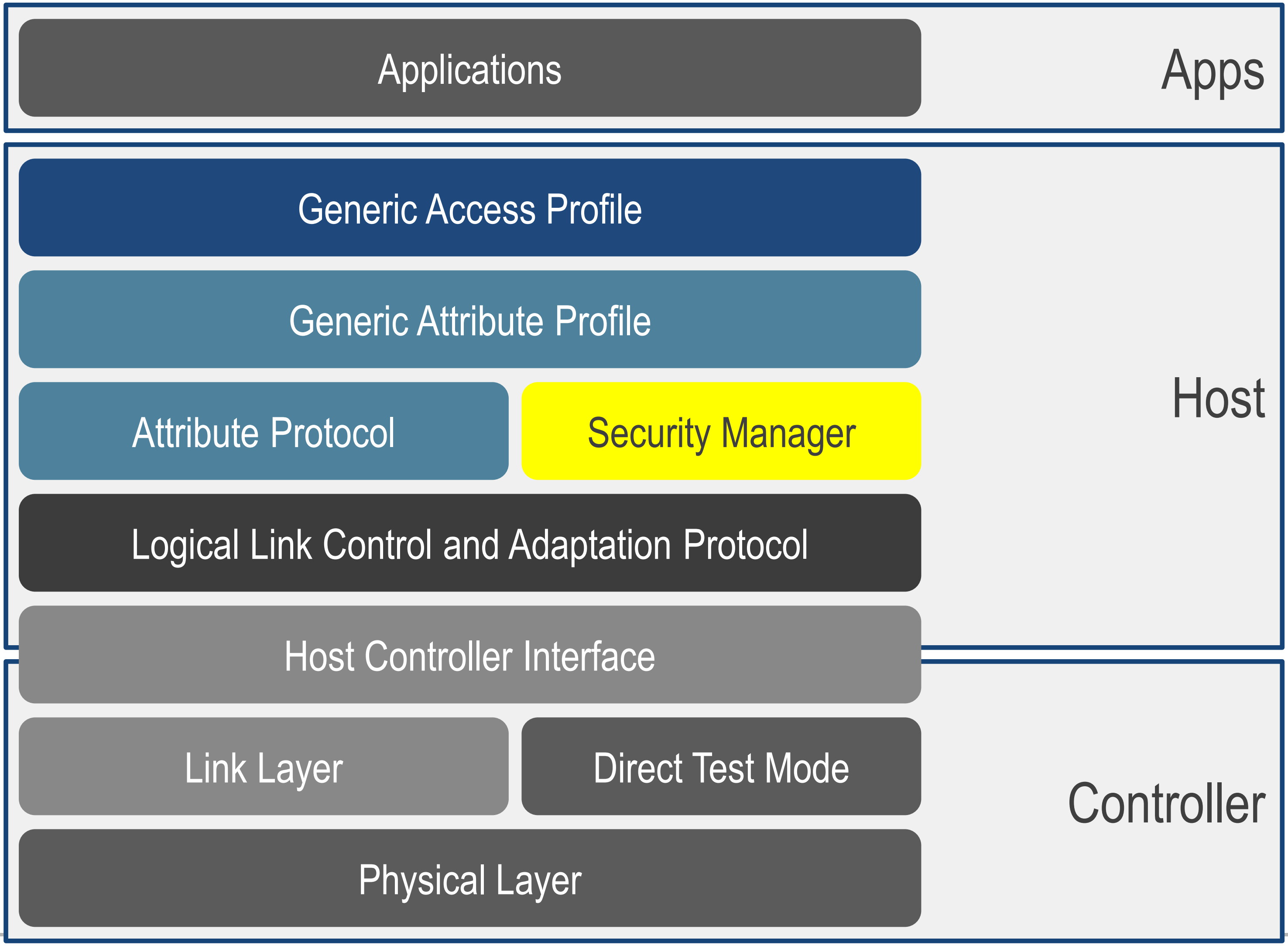


L2CAP Channel Types

- Higher Level Protocol Multiplexing
- Packet Segmentation and reassembly
- L2CAP in Bluetooth low energy operates in Basic Mode
  - Offers only fixed channel types
  - Connection oriented channels are not used in BTle

Channel Type	Local CID (sending)	Remote CID (receiving)
Attribute Protocol	0x0004 (fixed)	0x0004 (fixed)
Signaling	0x0005 (fixed)	0x0005 (fixed)
Security Manager Protocol	0x0006 (fixed)	0x0006 (fixed)

Stack Architecture

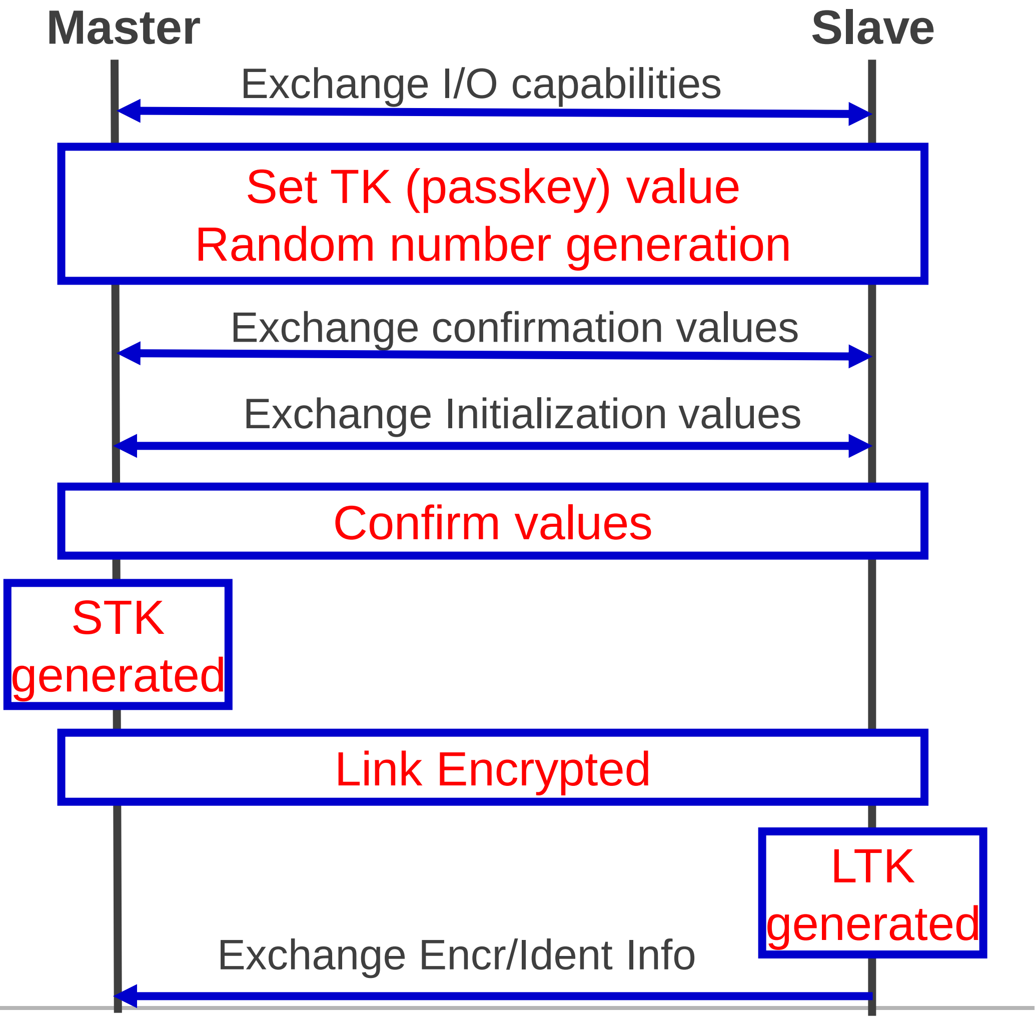


Security Manager Protocol

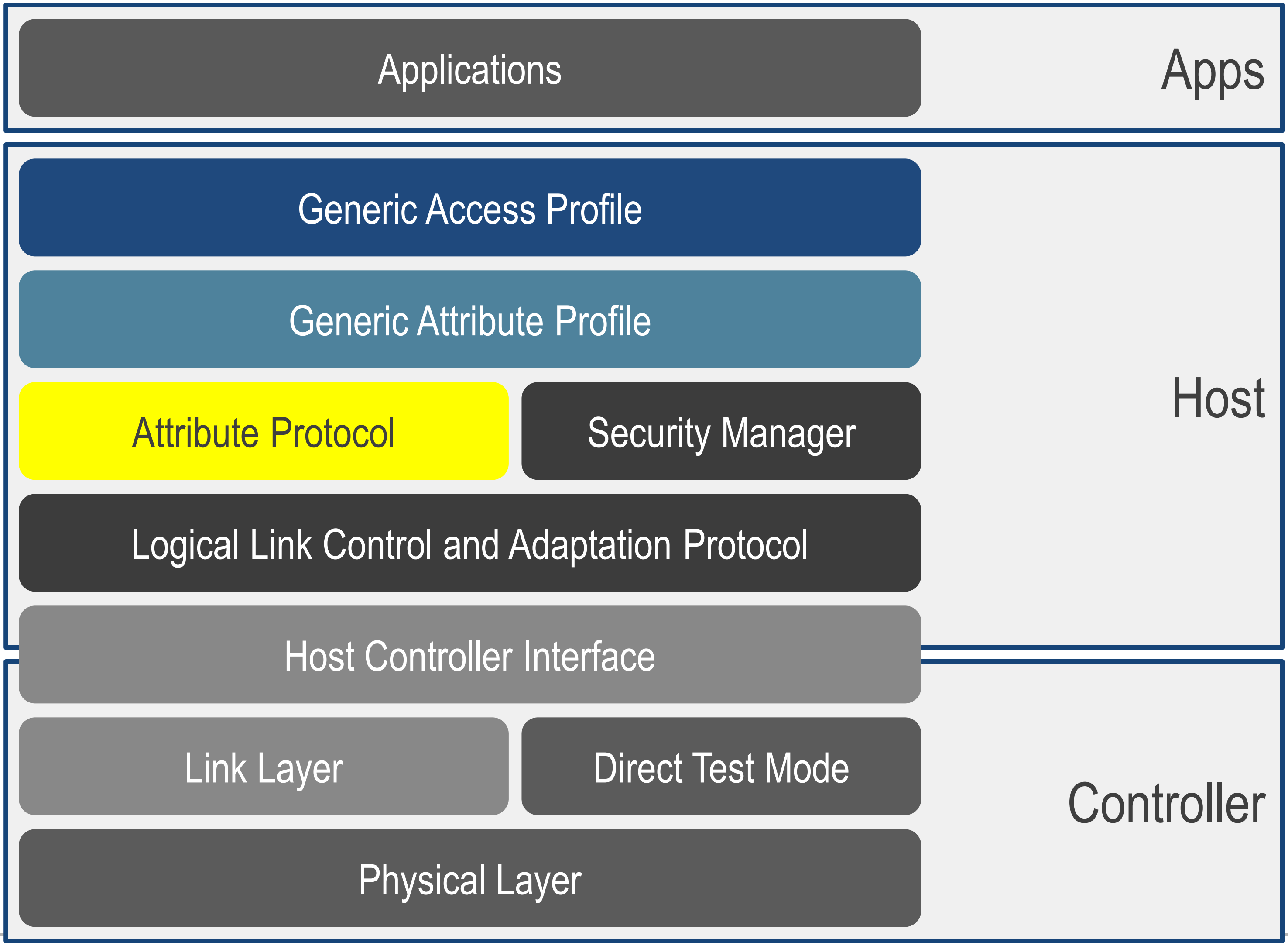
- Defines the protcol and behavior to manage
  - Pairing
  - Authentication and Encryption
- Uses L2CAP fixed channel 0x000
- Distributing key model
  - Slave generates and distributes key information to master
  - Master can use this key information when reconnecting
- Pairing
  - authentication based on capabilities / security requirements
  - side effect is encrypted link / key distribution
- Bonding
  - Devices save keys for bonded devices

Pairing

- Phase 1
  - Pairing request and response
  - Identifies IO capability (keypad, display, none)
  - Authentication requirements
- Phase 2
  - Confirmation values exchanged - based on TK, random number, and master/slave addresses
  - Short term key generated by master encrypts link
- Phase 3
  - Slave generates LTK using DIV
  - Shares key with Master to speed reconnections



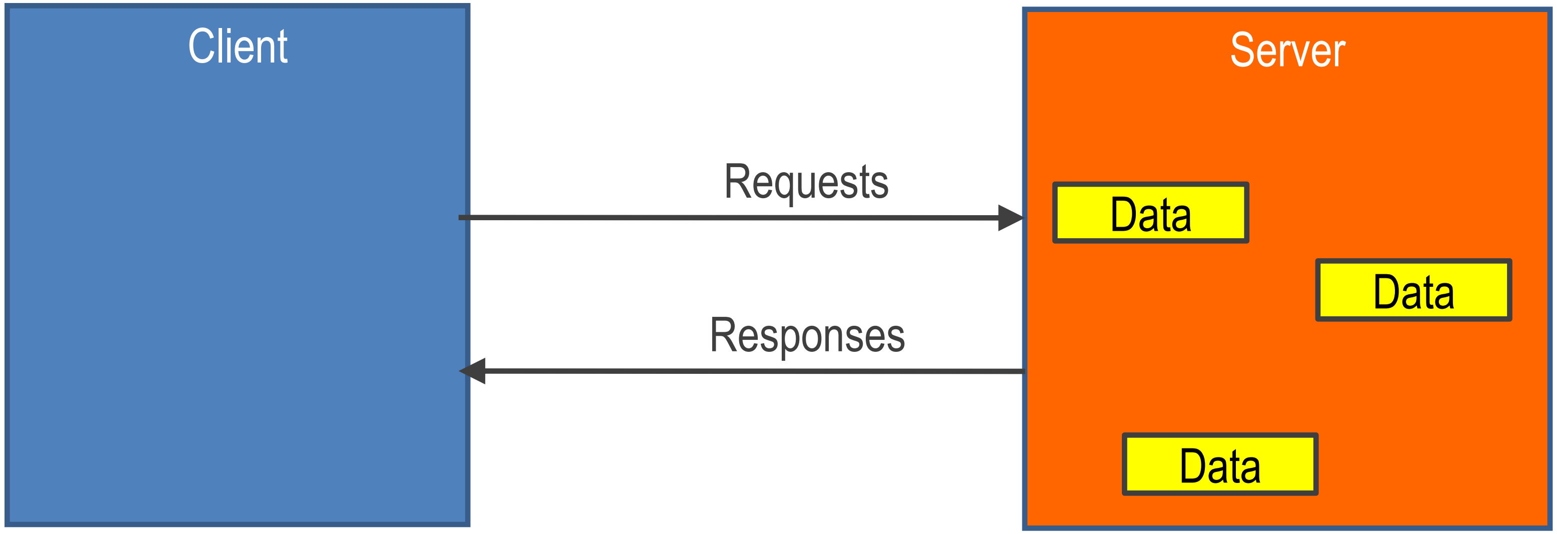
Stack Architecture



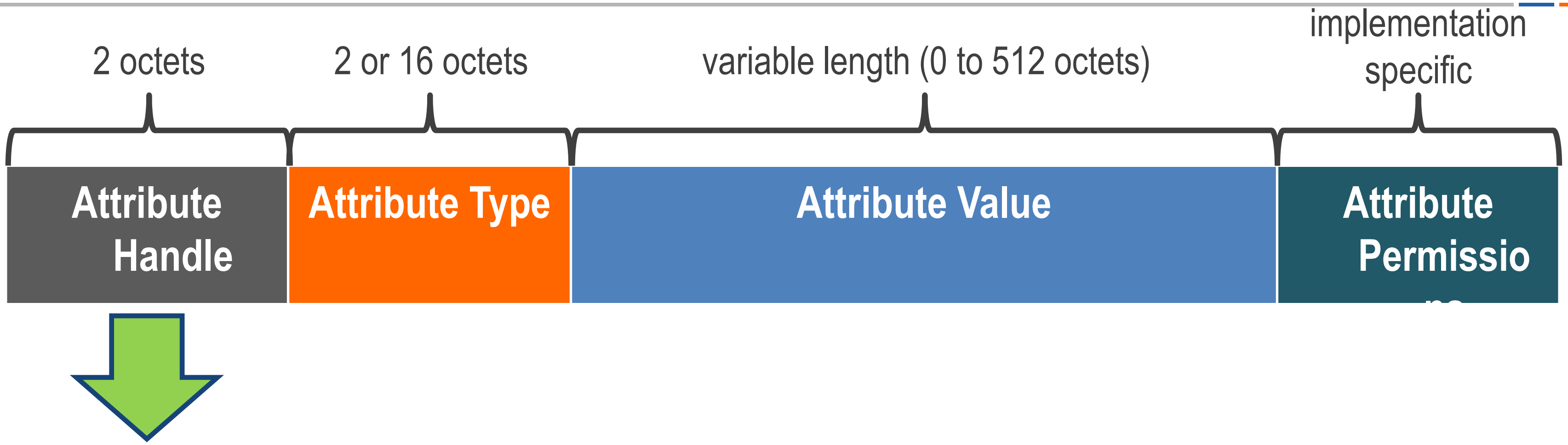
Attribute Protocol (ATT)

- Client Server Architecture
  - servers have data
  - clients request data to/from servers

- Servers expose data using Attributes



Attribute Handle



2 octets

2 or 16 octets

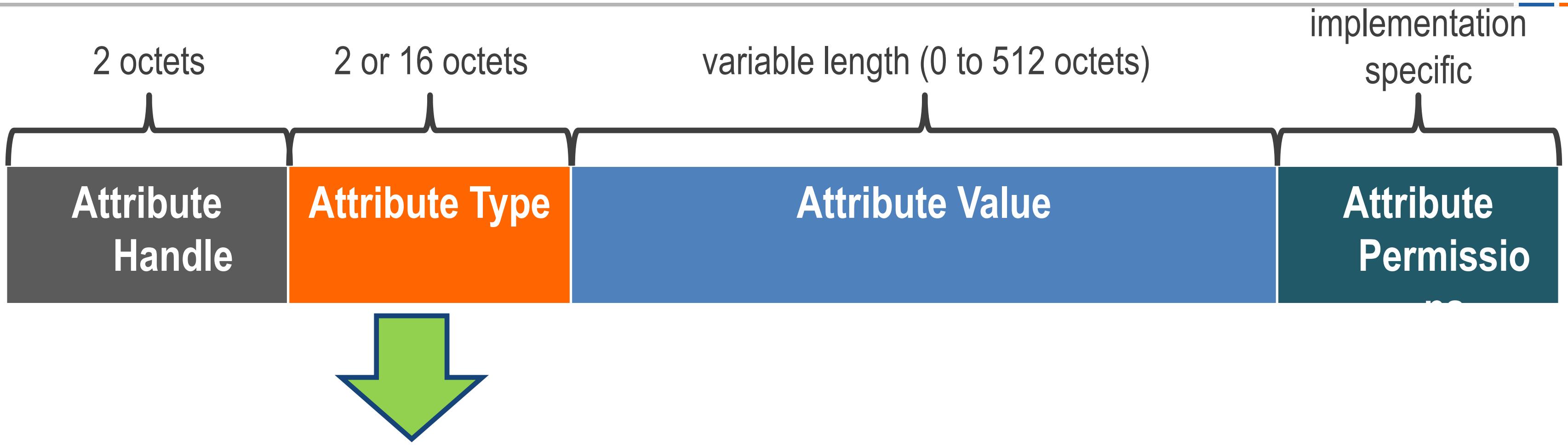
variable length (0 to 512 octets)

implementation specific

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissio ns

- Handle is a 16 bit value
  - 0x0000 is reserved – shall never be used
  - 0x0001 to 0xFFFF can be assigned to any attributes
- Handles are “sequential”
  - 0x0005 is “before” 0x0006
  - 0x0104 is “after” 0x00F8
- Always unique in the table

Attribute Type



2 octets

2 or 16 octets

variable length (0 to 512 octets)

implementation specific

Attribute Handle

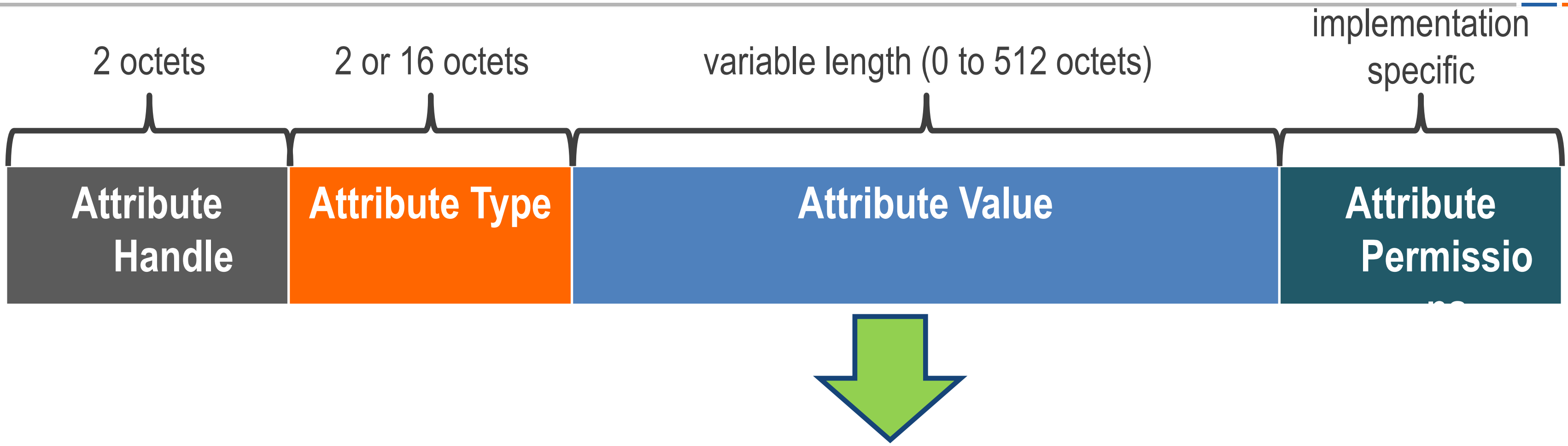
Attribute Type

Attribute Value

Attribute Permissio

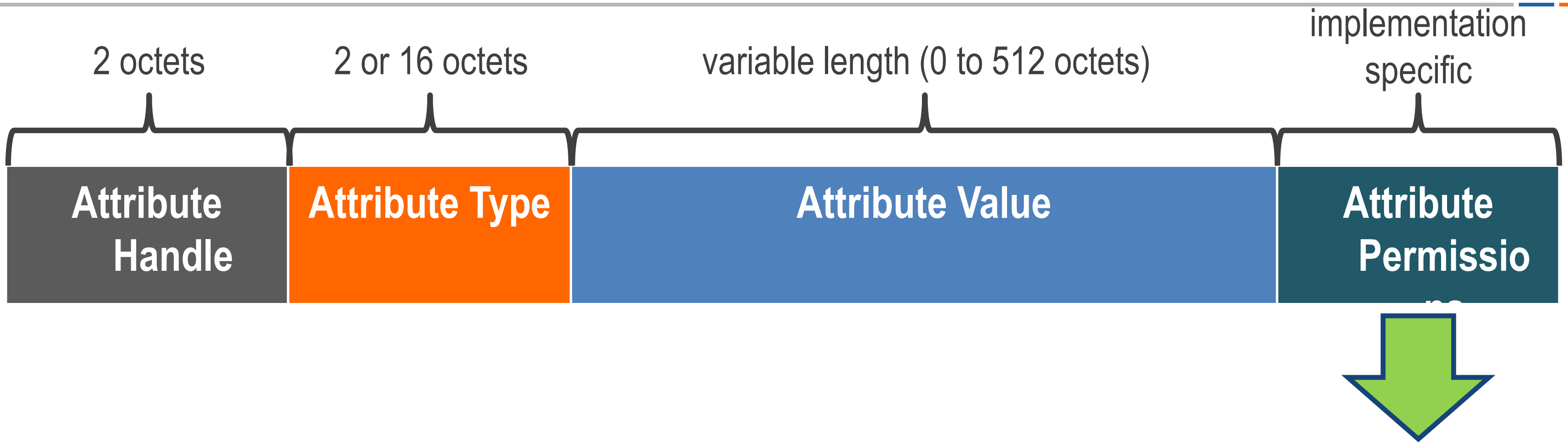
- SIG defined Attribute Types – 16 bits
  - Bluetooth_Base_UUID is: 00000000-0000-1000-8000 00805F9B34FB
  - Declarations - Defined GATT profile attribute types.
  - Descriptors - Defined attributes that describe a characteristic value
  - Numbers assigned to adopted services and characteristics
- Custom Attribute Types – 128 bit
  - Custom Services and characteristics
  - http://www.itu.int/ITU-T/asn1/uuid.html

Attribute Value



- An Attribute value is an array of octets, 0 to 512 octets in length can be fixed or variable length
- Each Attribute type defines the data structure for the attirbute value
  - Example: AttributeType = 0x2800 defines a 16 or 128 bit value
  - Example: Attribute Type = 0x2803 defines the Attribute Value to be {r, «Handle», «UUID»}
  - Example: Attribute Type = AlertLevel(0x2A06) defines Attribute value to be uint8

Attribute Permissions



2 octets | 2 or 16 octets | variable length (0 to 512 octets) | implementation specific

Attribute Handle | Attribute Type | Attribute Value | Attribute Permissio ns

- Attributes values may be:
  - readable / not readable
  - writeable / not writeable
  - readable & writeable / not readable & not writeable
- Attribute values may require:
  - authentication to read / write
  - authorization to read / write
  - encryption / pairing with sufficient strength to read / write

- Permissions not “discoverable” over Attribute Protocol
- If request to read an attribute value that cannot be read - Error Response «Read Not Permitted»
- If request to write an attribute value that requires authentication - Error Response «Insufficient Authentication» - Client must create authenticated connection and then retry
- There is no “pending” state

PROTOCOL METHODS

Protocol PDU Type	Sent by	Description
Request	Client	Client requests something from server – always causes a response
Response	Server	Server sends response to a request from a client
Command	Client	Client commands something to server – no response
Notification	Server	Server notifies client of new value – no confirmation
Indication	Server	Server indicates to client new value – always causes a confirmation
Confirmation	Client	Confirmation to an indication

PROTOCOL IS STATELESS

- After transaction complete
  - no state is stored in protocol

- A transaction is:
  - Request -> Response
  - Command
  - Notification
  - Indication -> Confirmation

SEQUENTIAL PROTOCOL

- Client can only send one request at a time
  - request completes after response received in client
- Server can only send one indication at a time
  - indication completes after confirmation received in server
- Commands and Notifications are no response / confirmation
  - can be sent at any time
  - could be dropped if buffer overflows – consider unreliable

Client Initiated Methods

- Request Method
  - Reading Attributes
    - ReadRequest(handle) ←→ ReadResponse(value)
    - ReadByTypeRequest(startingHandle, endHandle, UUID) ←→ ReadByTypeResponse(list of [handle, value] pair)
  - Writing Attributes
    - WriteRequest(handle, value) ←→ WriteReponse
  - Finding Attributes
    - FindInformation(startingHandle, endHandle, UUID) ←→ FindInformationResponse(format, [Handle, UUID])
- Example
  - Read (0x0022) => 0x04 ; Read (0x0098) => 0x0802

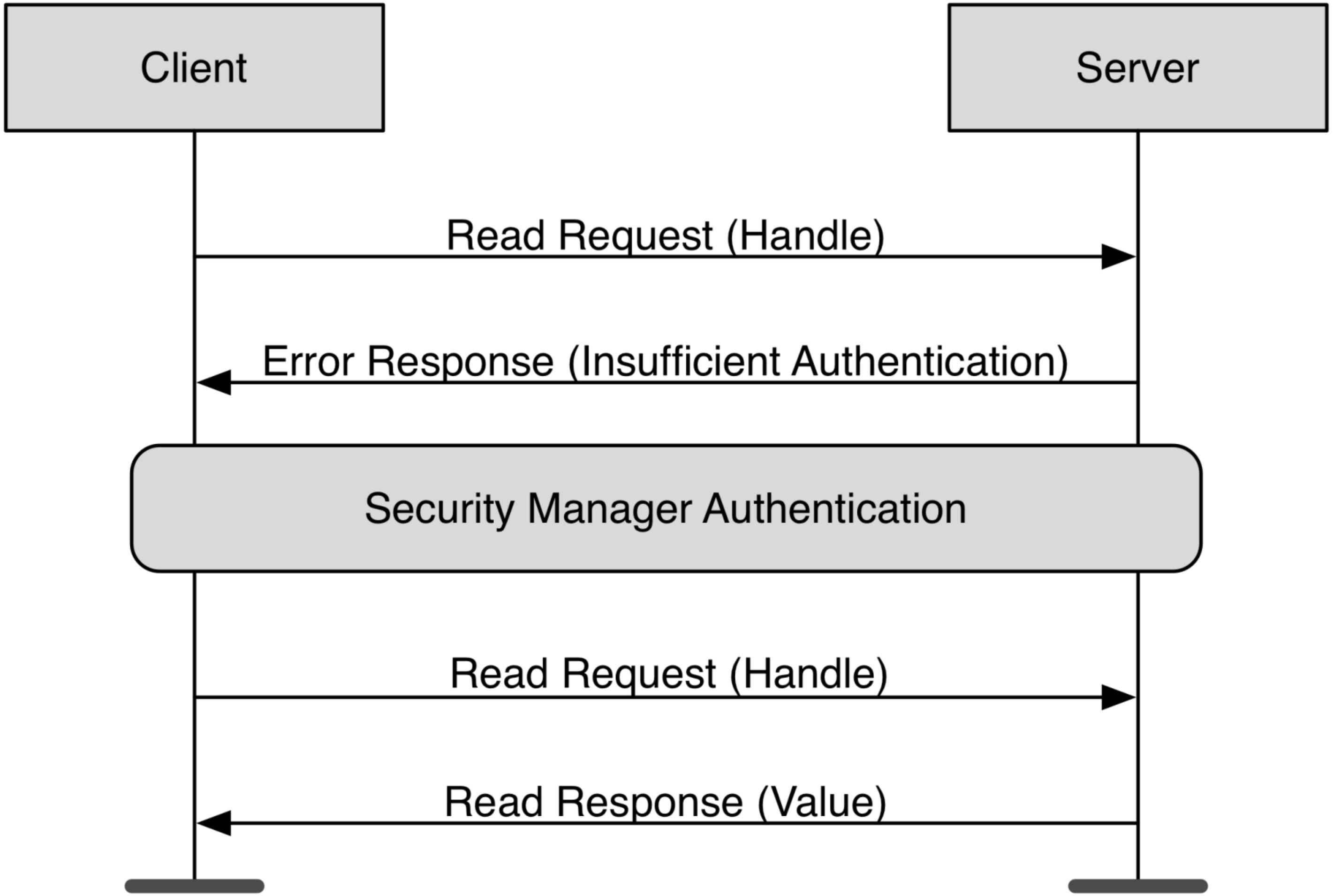
Server Initiated Methods

- `Handle Value Notification (handle, value)`

- `Handle Value Indication(handle, value) => Handle Value Confirmation ()`

Error Response

(any) Request (*) => Error Response (Opcode, Handle, Error Code)

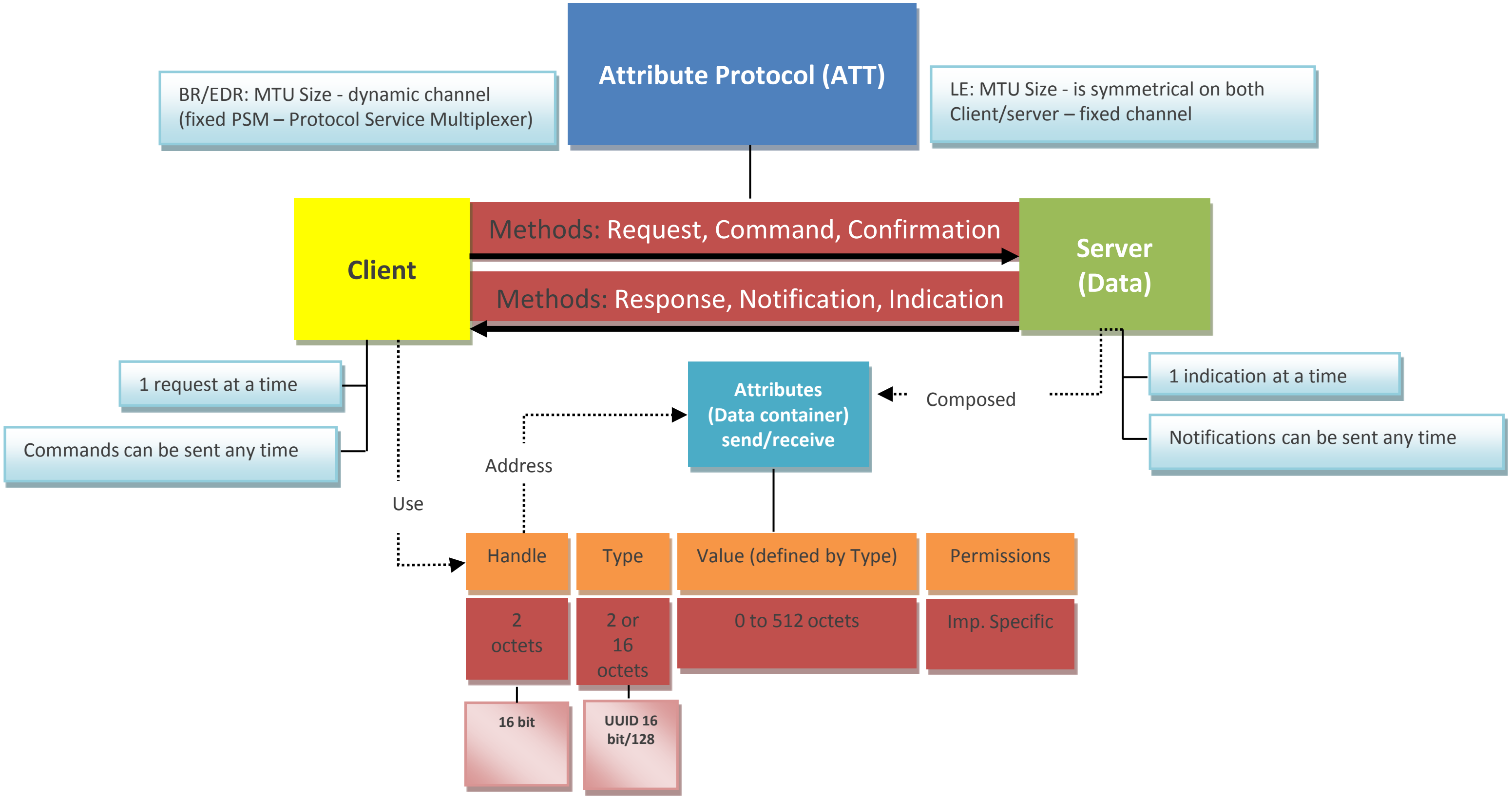

Client
Server
Read Request (Handle)
Error Response (Insufficient Authentication)
Security Manager Authentication
Read Request (Handle)
Read Response (Value)

Example: ATTRIBUTE Table

- Example – ReadRequest(0x0022) - ReadResponse(0x802)
- Example – ReadRequest(0x0004) – ReadResposne({r, 0x0006, <<Appearance>>

Handle	Type	Value	Permissions
0x0001	«Primary Service»	«GAP»	R
0x0002	«Characteristic»	{r, 0x0003, «Device Name»}	R
0x0003	«Device Name»	"Temperature Sensor"	R
0x0004	«Characteristic»	{r, 0x0006, «Appearance»}	R
0x0006	«Appearance»	«Thermometer»	R
0x000F	«Primary Service»	«GATT»	R
0x0010	«Characteristic»	{r, 0x0012, «Attribute Opcodes Supported»}	R
0x0012	«Attribute Opcodes Supported»	0x00003FDF	R
0x0020	«Primary Service»	«Temperature»	R
0x0021	«Characteristic»	{r, 0x0022, «Temperature Celsius»}	R
0x0022	«Temperature Celsius»	0x0802	R*

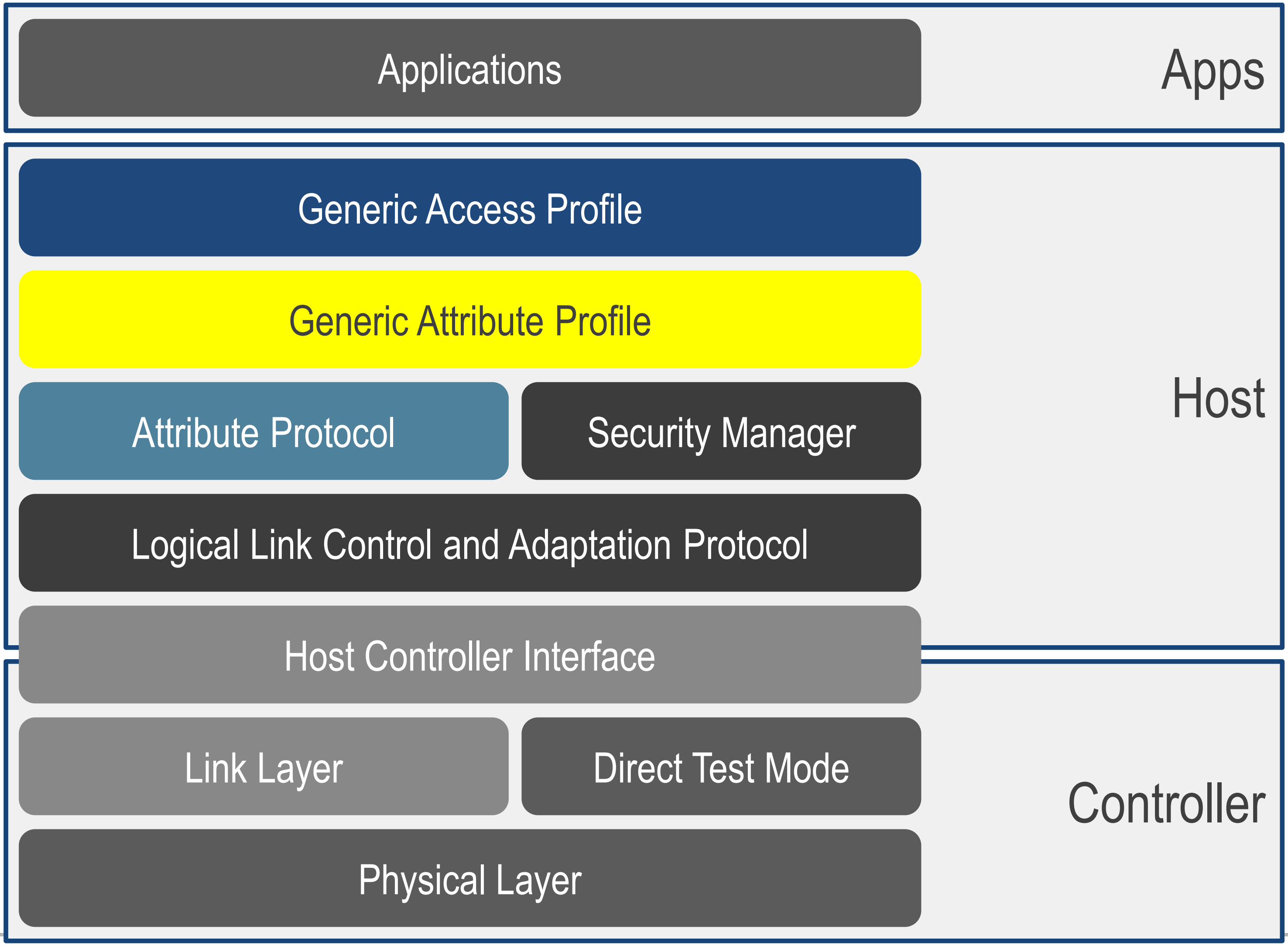
Attribute Protocol (ATT) Summary



Attribute Protocol (ATT)
BR/EDR: MTU Size - dynamic channel (fixed PSM – Protocol Service Multiplexer)
LE: MTU Size - is symmetrical on both Client/server – fixed channel
Client
Methods: Request, Command, Confirmation
Methods: Response, Notification, Indication
Server (Data)
1 request at a time
Commands can be sent any time
Attributes (Data container) send/receive
Composed
1 indication at a time
Notifications can be sent any time
Address
Use
Handle
Type
Value (defined by Type)
Permissions
2 octets
2 or 16 octets
0 to 512 octets
Imp. Specific
16 bit
UUID 16 bit/128

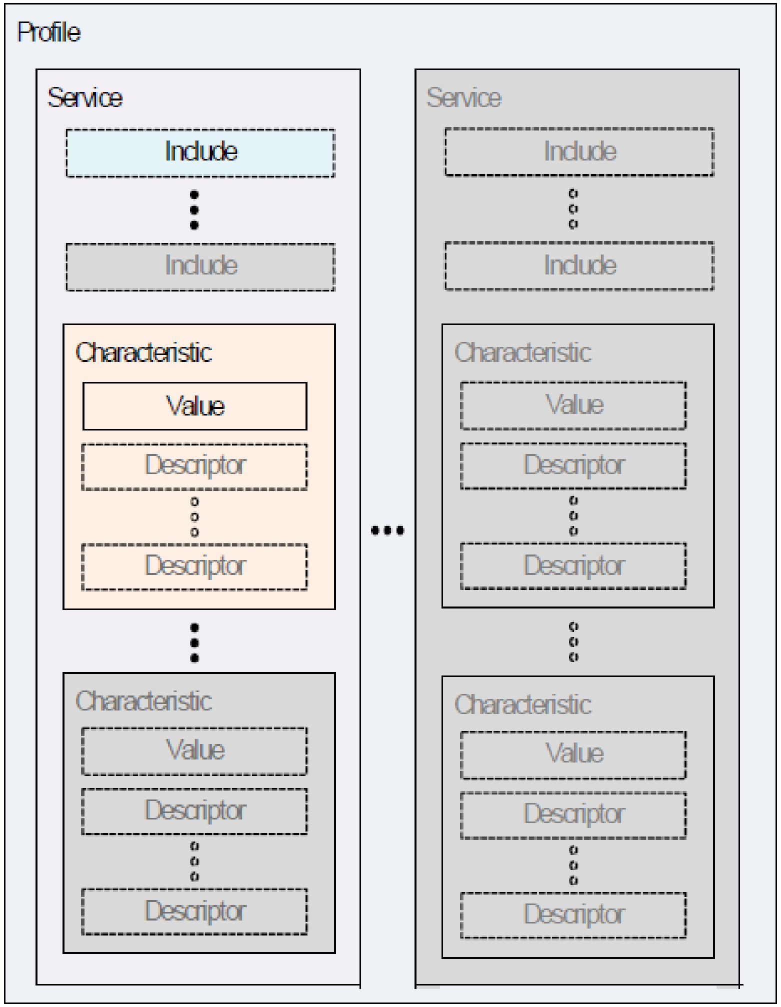
- Exposes Data using Typed, Addressable Attributes: Handle, Type, Value
- Methods for finding, reading, writing attributes by client
- Methods for sending notifications / indications by server

Stack Architecture



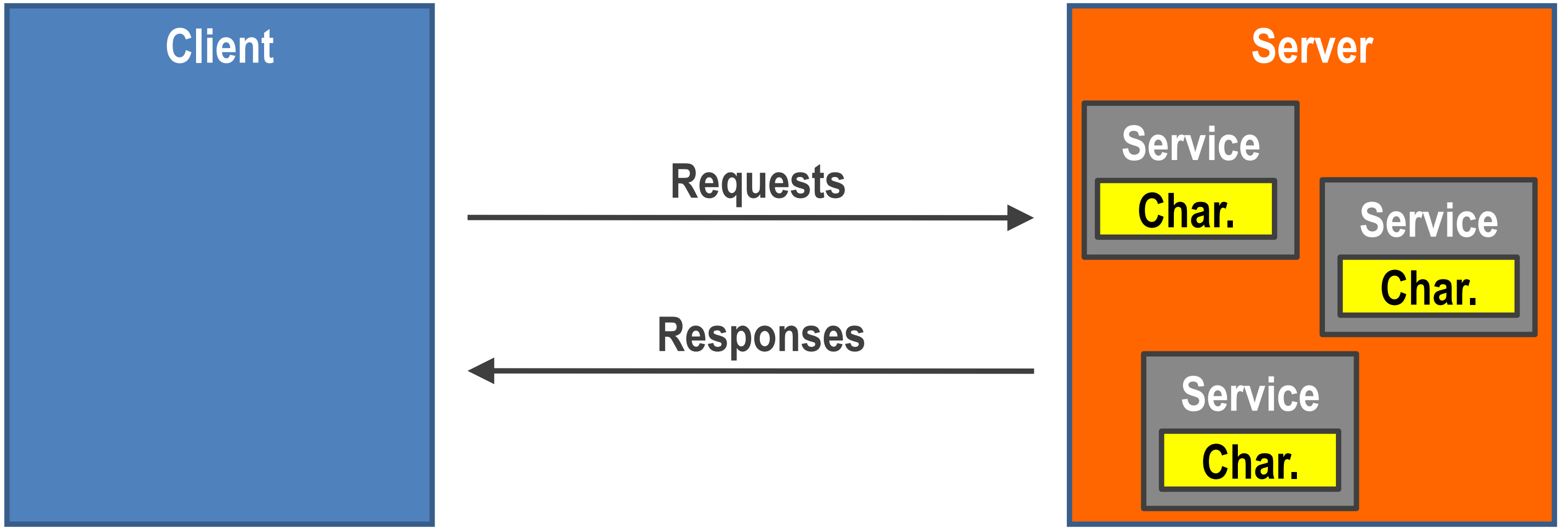
GENERIC ATTRIBUTE PROFILE (GATT) Hierarchy

- Built on top of the ATT
- Provides a framework for developing profiles
- A profile is composed of one or more services.
- A service is composed of characteristics or references to other services.
- Each characteristic contains a value and may contain optional information about the value.



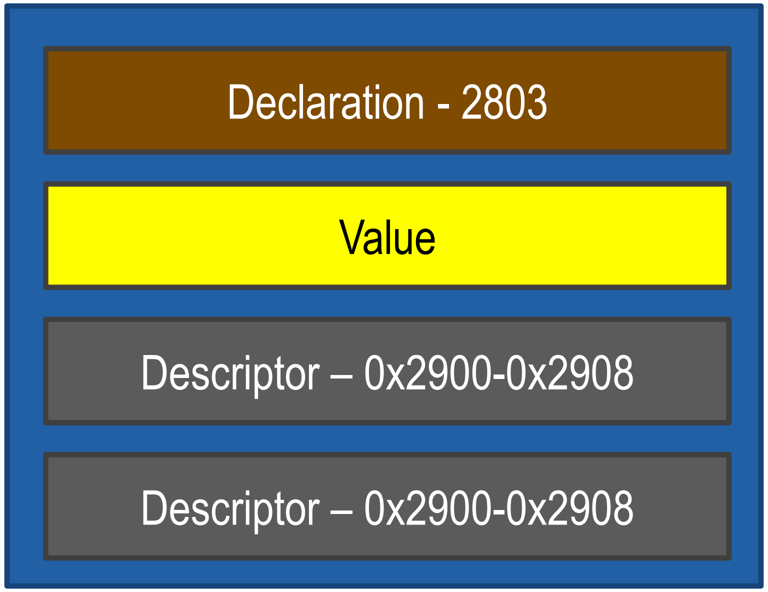
Client Server Architecture

- Same client server architecture as Attribute Protocol
  - except that data is encapsulated in “Services”
  - data is exposed in “Characteristic”



WHAT IS A CHARACTERISTIC?

- Group of attributes to define data
- Characteristics specify
  - Data size, format
  - Permissible Values
  - Permissions
- Represented in Attribute Table as multiple attirbutes
  - Characteristic Declaration
  - Characteristic Value
  - Characteristic Descriptors – 1 : n
- Example – Alert Level
  - Uint8
  - Permissbile values: 0, 1, 2
  - R/W



ATTRIBUTES ARE FLAT

Handle	Type	Value	Permissions
0x0001	«Primary Service»	«GAP»	R
0x0002	«Characteristic»	{r, 0x0003, «Device Name»}	R
0x0003	«Device Name»	“Temperature Sensor”	R
0x0004	«Characteristic»	{r, 0x0006, «Appearance»}	R
0x0006	«Appearance»	«Thermometer»	R
0x000F	«Primary Service»	«GATT»	R
0x0010	«Characteristic»	{r, 0x0012, «Attribute Opcodes Supported»}	R
0x0012	«Attribute Opcodes Supported»	0x00003FDF	R
0x0020	«Primary Service»	«Temperature»	R
0x0021	«Characteristic»	{r, 0x0022, «Temperature Celsius»}	R
0x0022	«Temperature Celsius»	0x0802	R*

GROUPING GIVES STRUCTURE

Handle	Type	Value	Permissions
0x0001	«Primary Service»	«GAP»	R
0x0002	«Characteristic»	{r, 0x0003, «Device Name»}	R
0x0003	«Device Name»	“Temperature Sensor”	R
0x0004	«Characteristic»	{r, 0x0006, «Appearance»}	R
0x0006	«Appearance»	«Thermometer»	R
0x000F	«Primary Service»	«GATT»	R
0x0010	«Characteristic»	{r, 0x0012, «Attribute Opcodes Supported»}	R
0x0012	«Attribute Opcodes Supported»	0x00003FDF	R
0x0020	«Primary Service»	«Temperature»	R
0x0021	«Characteristic»	{r, 0x0022, «Temperature Celsius»}	R
0x0022	«Temperature Celsius»	0x0802	R*

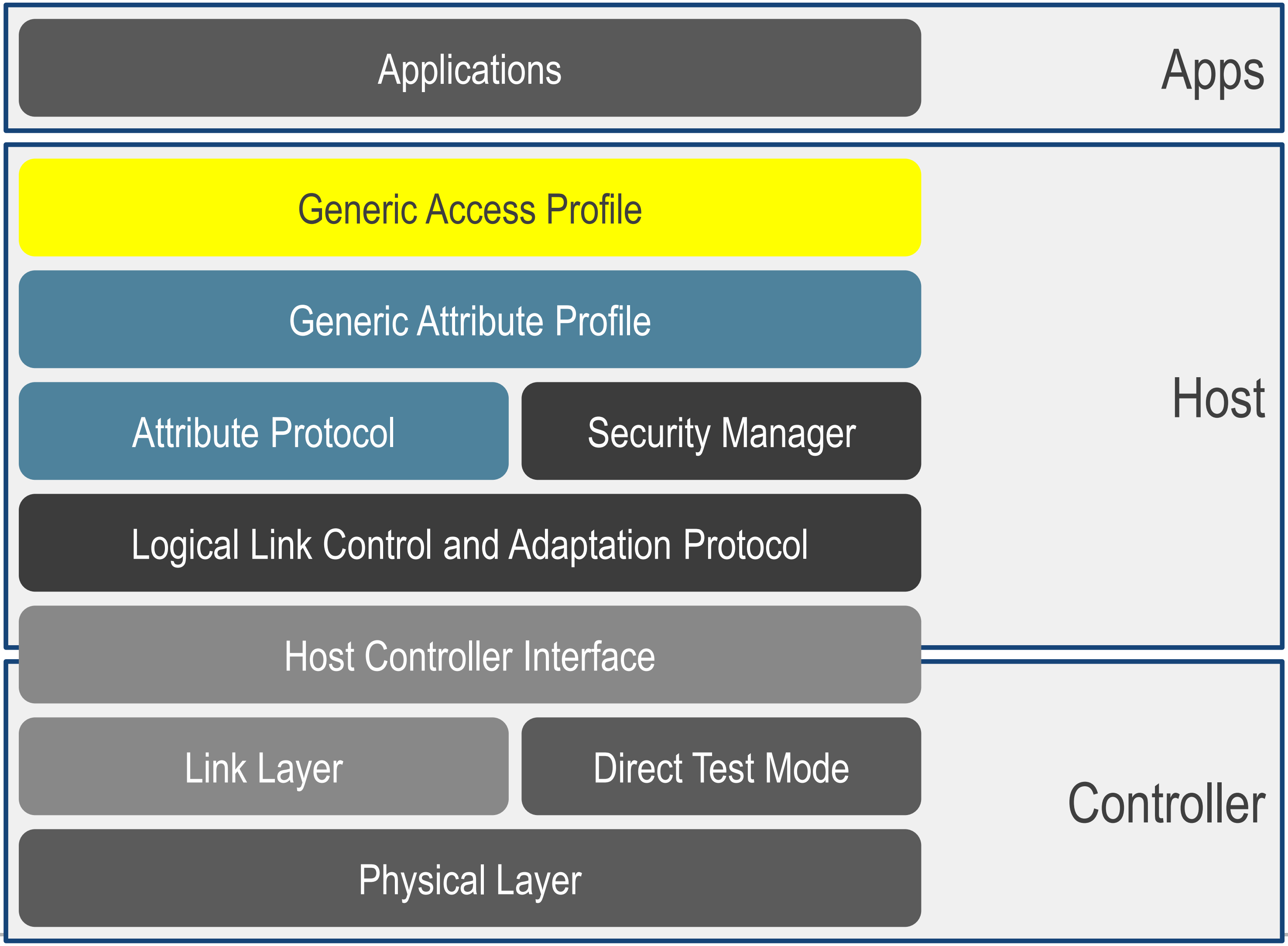
GROUPING GIVES STRUCTURE

Handle	Type	Value	Permissions
0x0001	«Primary Service»	«GAP»	R
0x0002	«Characteristic»	{r, 0x0003, «Device Name»}	R
0x0003	«Device Name»	“Temperature Sensor”	R
0x0004	«Characteristic»	{r, 0x0006, «Appearance»}	R
0x0006	«Appearance»	«Thermometer»	R
0x000F	«Primary Service»	«GATT»	R
0x0010	«Characteristic»	{r, 0x0012, «Attribute Opcodes Supported»}	R
0x0012	«Attribute Opcodes Supported»	0x00003FDF	R
0x0020	«Primary Service»	«Temperature»	R
0x0021	«Characteristic»	{r, 0x0022, «Temperature Celsius»}	R
0x0022	«Temperature Celsius»	0x0802	R*

GROUPING GIVES STRUCTURE

Handle	Type	Value	Permissions
0x0001	«Primary Service»	«GAP»	R
0x0002	«Characteristic»	{r, 0x0003, «Device Name»}	R
0x0003	«Device Name»	“Temperature Sensor”	R
0x0004	«Characteristic»	{r, 0x0006, «Appearance»}	R
0x0006	«Appearance»	«Thermometer»	R
0x000F	«Primary Service»	«GATT»	R
0x0010	«Characteristic»	{r, 0x0012, «Attribute Opcodes Supported»}	R
0x0012	«Attribute Opcodes Supported»	0x00003FDF	R
0x0020	«Primary Service»	«Temperature»	R
0x0021	«Characteristic»	{r, 0x0022, «Temperature Celsius»}	R
0x0022	«Temperature Celsius»	0x0802	R*

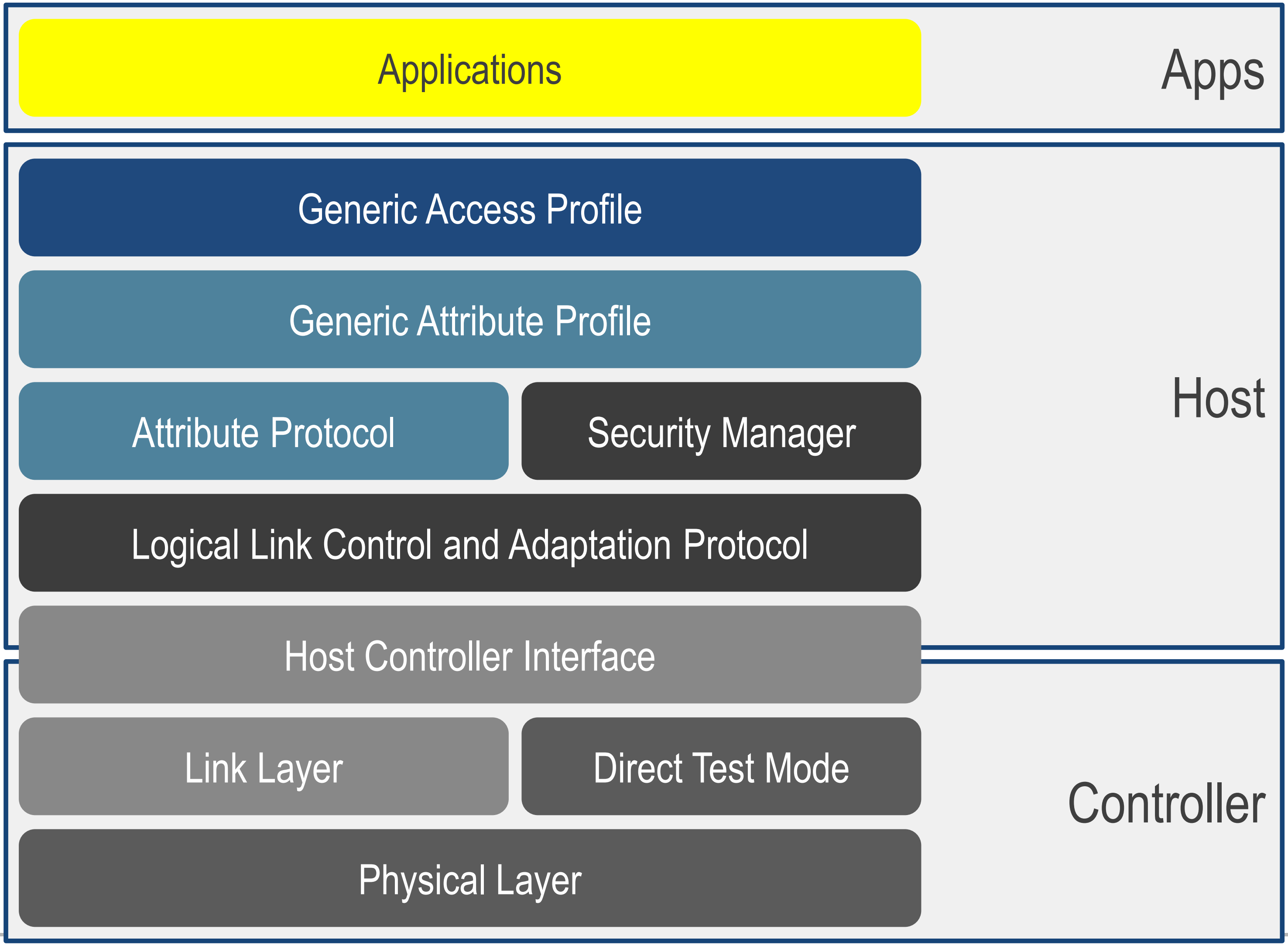
Stack Architecture



Generic Access Profile - GAP

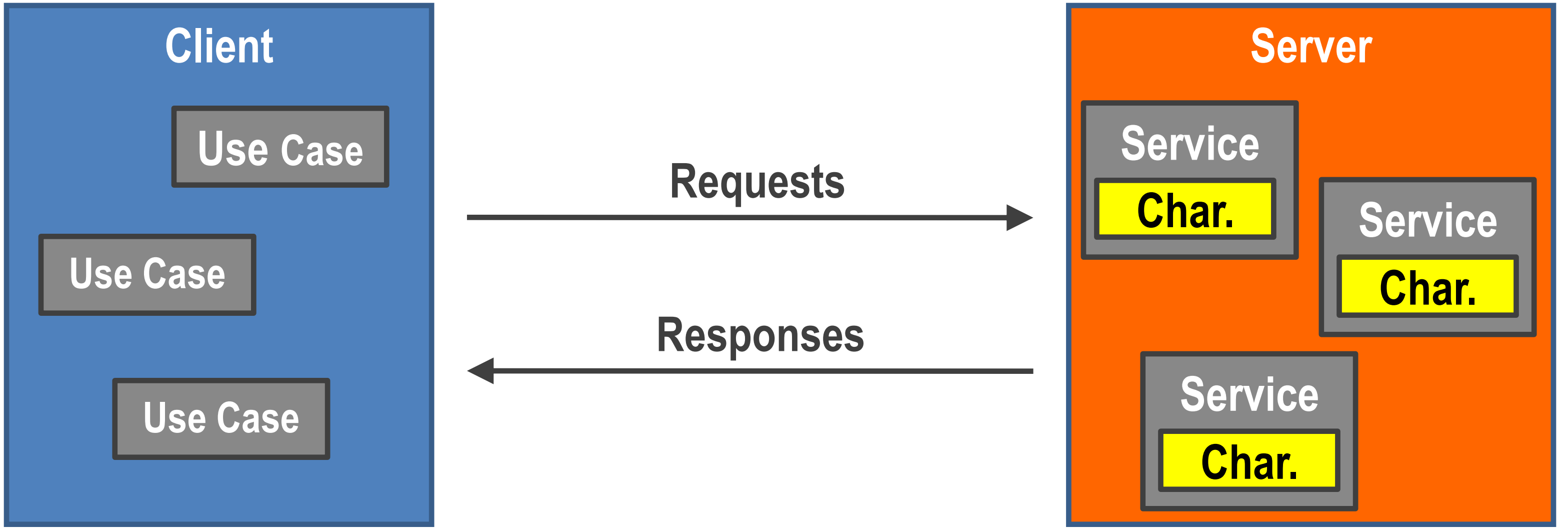
- Defines procedures for:
  - Discovering identities, names, and basic capabilities
  - Creating bonds
  - Exchange of security information
  - Establishing connections
- Defines Advertising and Scan Response Data formats
- All profiles are built upon GAP
- Defines profile roles
  - Broadcaster – sends non-connectable advertisement and never connect
  - Observer – listens to advertisement packets but never connect
  - Peripheral – Always take the role of slave
  - Central – Always take the role of master

Stack Architecture



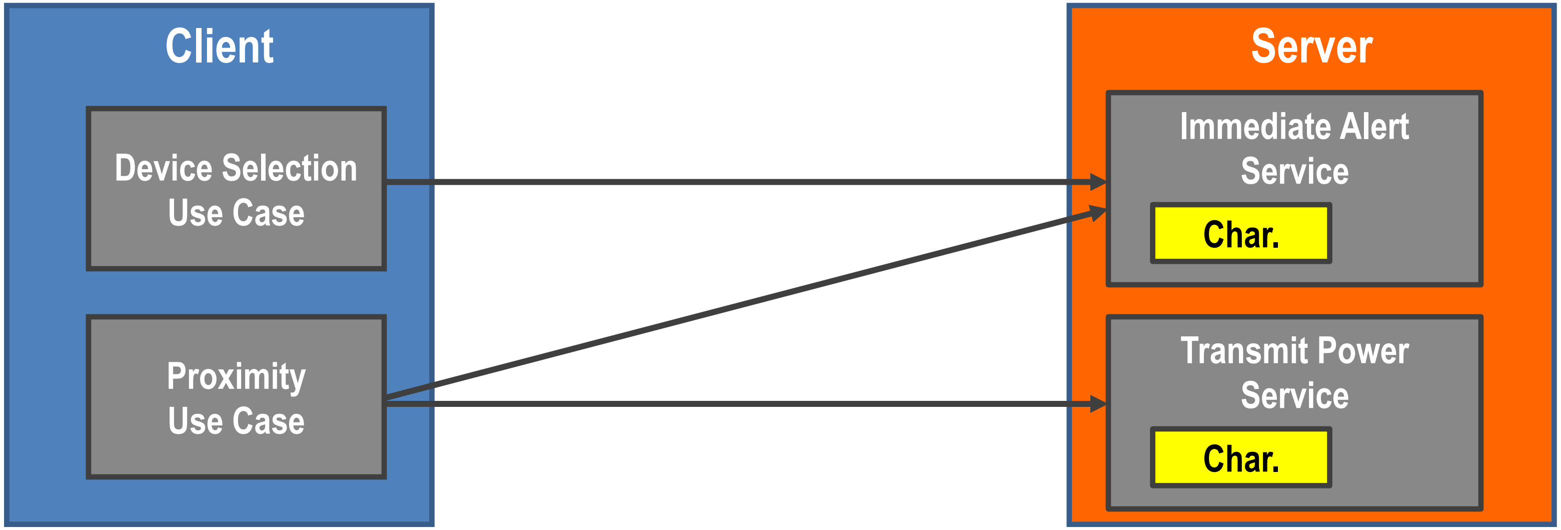
BTle Applications

- Client Server Architecture
  - Services – exposes behavior that have characteristics
  - Use Cases– define how to use services on a peer



Use Cases and Services

- There is not a one-to-one link between services and use cases
- Clients implement use cases, Servers implement services
- Use cases can use multiple services



GATT based Profile Specifications

- Profile specifications
  - Use case
  - Behaviors
  - Discovery Procedures
  - Connection Parameters (slave latency, conn Interval) etc
  - Profile Roles
- Service specifications
  - Characteristics (Mandatory, Optional)
  - Characteristics Properties (Broadcast, Control Point etc)
- Characteristics specifications
  - Specify structure of value – Eg: Alert Level – 1 byte
  - Permissible values – Eg: 0 – No Alert, 1 – Medium Alert, 2 – High Alert
  - Permissions – Read/Write

Heart Rate Profile

- User Scenarios
  - The Heart Rate Profile is used to enable a data collection device to obtain data from a Heart Rate Sensor that exposes the Heart Rate Service
- Roles
  - Heart Rate Sensor
  - Heart Rate Collector
- Heart Rate Sensor Role requirements
  - Heart Rate Service - Mandatory
  - Device Information Service - Manadatory
- Characteristics – Heart Rate Service
  - Heart Rate Measurement - Notify .
  - Heart Rate Measurement - Client Characteristic Configuration descriptor
  - Body Sensor Location - Read - Optional

Proximity Profile

- User Scenarios
  - Leaving a phone behind
  - Leaving keys behind
  - Child straying too far
  - Hospital patient from bed
  - Automatic PC Locking & Unlocking
  - Automatic PC Locking & Authenticated Unlocking
- Roles
  - Proximity Monitor
  - Proximity Reporter
- Proximity Profile
  - Specifies services used
  - Specifies GAP requirements for discoverability/connectability
- Services
  - Link Loss Service
  - Immediate Alert Service
  - Tx Power Service

Attribute Table – Proximity Profile

- How many Attributes?
- How many Services?
- Turn AlertLevel for Primary Service

Handle	Type	Value	Permissions
0x0001	**«Primary Service» 0x2800**	«Link Loss Service» - **0x1803**	R
0x0002	**«Characteristic» 0x2803**	{r, 0x0003, «Alert Level - **0x2A06**»}	R
0x0003	**«Alert Level» 0x2A06**	0, 1 or 2	R, W
0x0004	**«Primary Service» 0x2800**	«Immediate Alert Service» - **0x1802**	R
0x0005	**«Characteristic» 0x2803**	{r, 0x0006, «Alert Level - **0x2A06**»}	R
0x0006	**«Alert Level» 0x2A06**	0 , 1 or 2	N
0x0007	**<<Client_Conf>> - 0x2903**	Bit 0 – Notification – On/ff, Bit 1- Indication On/Off	R/W
0x0010	**«Primary Service» 0x2800**	«TX Power» - **0x1804**	R
0x0011	**«Characteristic» 0x2803**	{r, 0x0006, «TX_Power_Level - **0x2A06**»}	R
0x0012	**«TX_Power_Level>> 0x2A06**	+18 dbm to -18 dbm	R

Attribute Table – Battery Service

- Services – Battery
- Characteristics – Battery Level – 0 to 100

Handle	Type	Value	Permissions
0x0001	«Primary Service» 0x2800	Battery Service UUID	R
0x0002	«Characteristic» 0x2803	{r, 0x0003, Battery Level UUID – 0x2A19}	R
0x0003	BatteryLevelUUID 0x2A19	0 to 100	R

- How many Attributes?
- How many Services?
- How many Characteristics ?
- Can we do a write operation on 0x003 ?

Attribute Table – Find Me Profile

- User Scanario:Defines the behavior when a button is pressed, an immediate alert happens on peer device
- Profile Roles – Find Me Locator, FindMe Target
- Services - FindMe
- Characteristics: Alert Level

Handle	Type	Value	Permissions
0x0001	«Primary Service» 0x2800	«FindMe Service UUID »	R
0x0002	«Characteristic» 0x2803	{r, 0x0003, «Alert Level - **0x2A06**»}	R
0x0003	«Alert Level» 0x2A06	0, 1 or 2	R, W

- How many Attributes?
- How many Services?

2011 Published Profiles

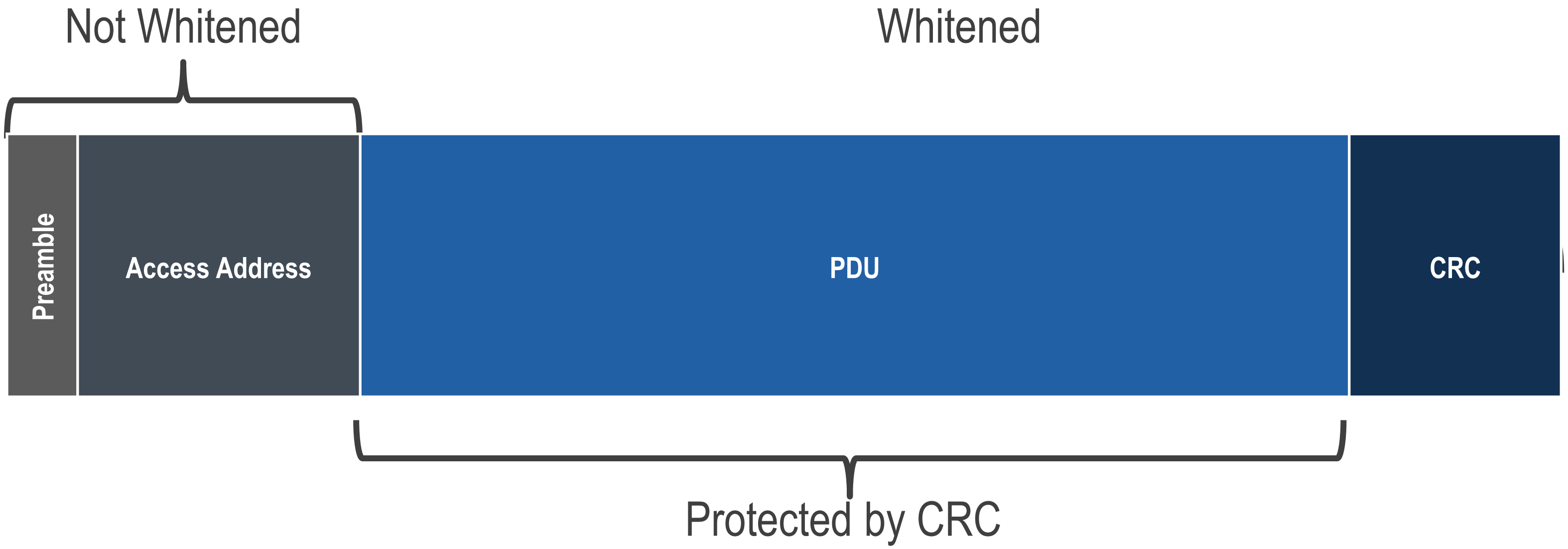
- Alert Notification Profile
- Alert Notification Service
- Blood Pressure Profile
- Blood Pressure Service
- Current Time Service
- Device Information Service
- Find Me Profile
- Health Thermometer Profile
- Health Thermometer Service
- Heart Rate Profile
- Heart Rate Service
- Immediate Alert Service
- Link Loss Service
- Next DST Change Service
- Phone Alert Status Profile
- Phone Alert Status Service
- Proximity Profile
- Reference Time Update Svc
- Time Profile
- Tx Power Service

Agenda

- Bluetooth Technology Evolution
- Architectural Overview
- Stack Architecture
  - Physical Layer
  - Link Layer
  - HCI Layer
  - L2CAP Layer
  - Security Manager Protocol
  - Attribute Protocol
  - Generic Attribute Profile
  - Generic Access Profile
  - Applications
- Air Interface Packet Structure

One Packet Format

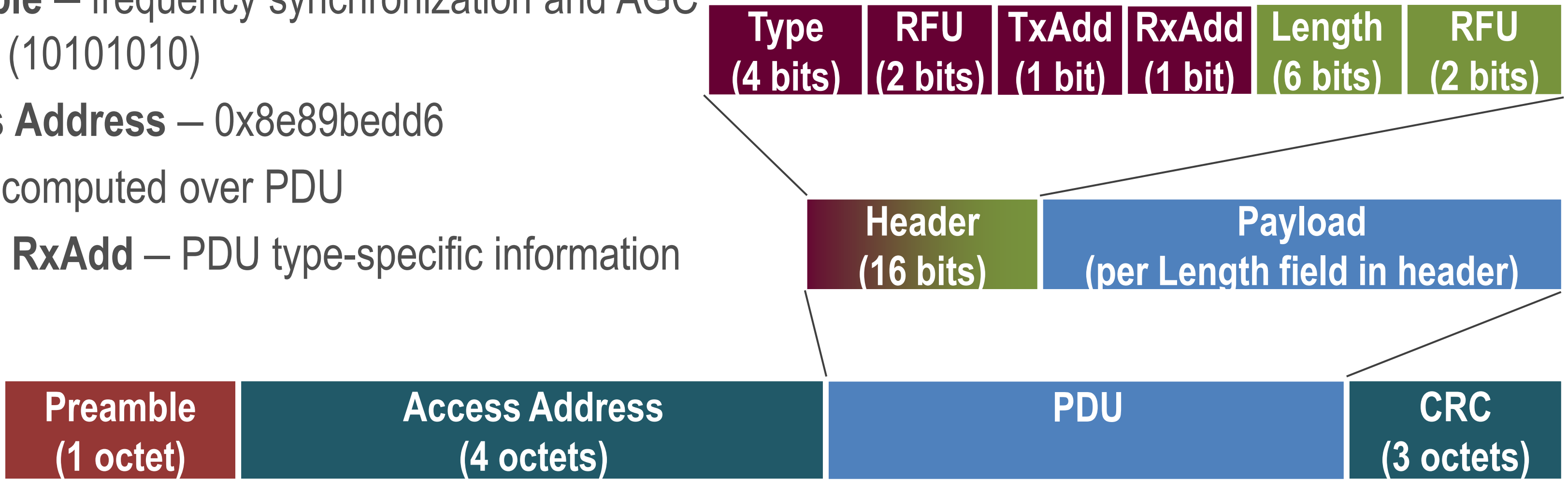
- Used for Advertising and Data Channel Packets
- Preamble (0x55, 0xAA)
  - Frequency synchronization, symbol timing estimation, AGC training
- Access Address
  - Advertising packets – always 0x8e89bed6
  - Data packets – different for each link layer connection
- Packet Data Unit
  - Defined based upon packet types



Air Interface Packets – Advertising Packets

Type	Packet	Usage
0000	ADV_IND	Connectable undirected advertising event
0001	ADV_DIRECT_IND	Connectable directed advertising event
0010	ADV_NONCONN_IND	Non-connectable undirected advertising event
0011	SCAN_REQ	Scan request for further information from advertiser
0100	SCAN_RSP	Response to scan request from scanner
0101	CONNECT_REQ	Connect request by Initiator
0110	ADV_DISCOVER_IND	Discoverable undirected advertising event

- **Preamble** – frequency synchronization and AGC training (10101010)
- **Access Address** – 0x8e89bedd6
- **CRC** – computed over PDU
- **TxAdd, RxAdd** – PDU type-specific information



Type (4 bits) | RFU (2 bits) | TxAdd (1 bit) | RxAdd (1 bit) | Length (6 bits) | RFU (2 bits)

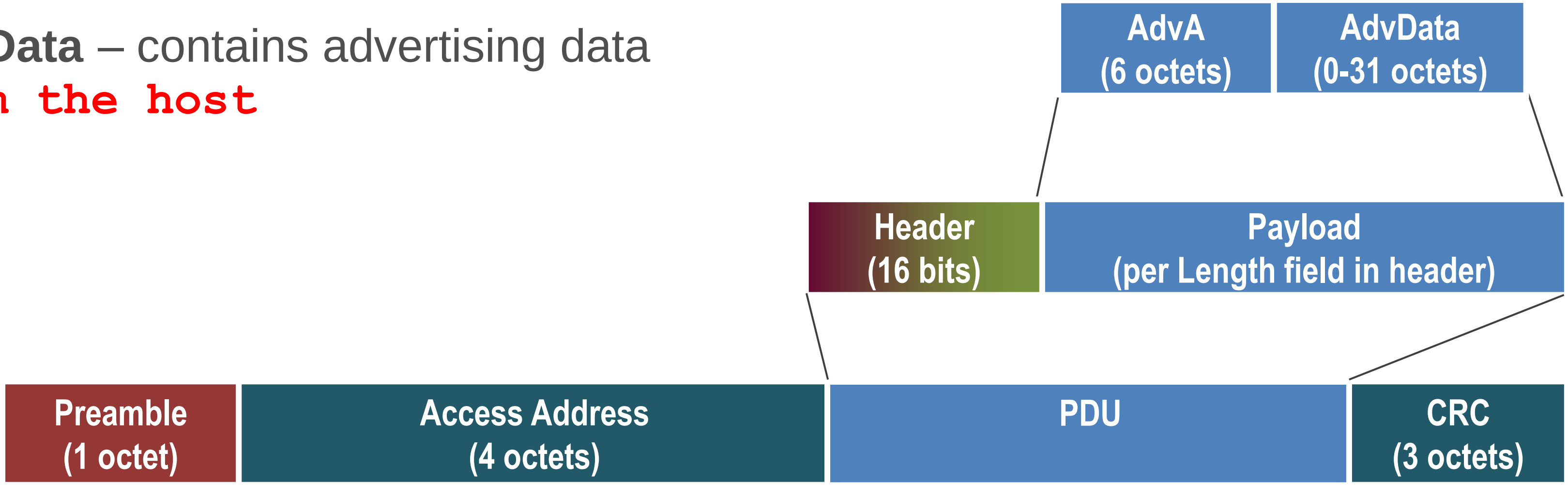
Header (16 bits) | Payload (per Length field in header)

Preamble (1 octet) | Access Address (4 octets) | PDU | CRC (3 octets)

Air Interface Packets – Advertising PDUs (Undirected)

Type	Packet	Usage
0000	ADV_IND	Connectable undirected advertising event
0010	ADV_NONCONN_IND	Non-connectable undirected advertising event
0110	ADV_DISCOVER_IND	Discoverable undirected advertising event

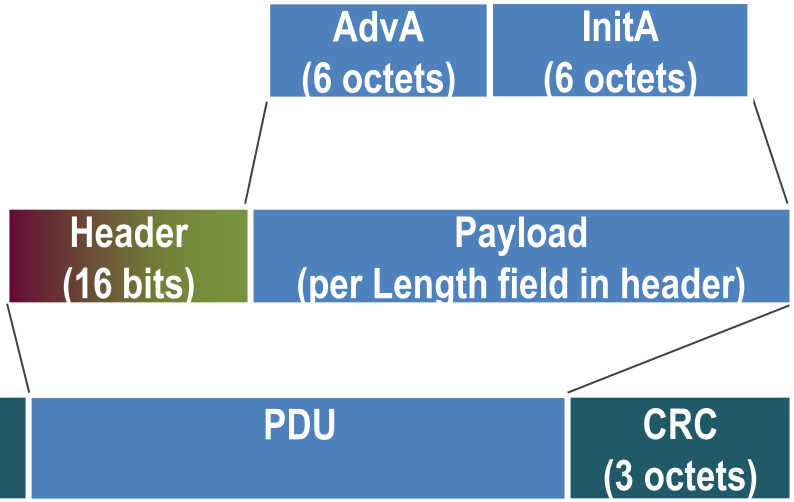
- **AdvA** – contains advertiser’s public address if TxAdd=0, or a random address if TxAdd=1
- **AdvData** – contains advertising data `from the host`



Air Interface Packets – Advertising PDUs (Directed)

Type	Packet	Usage
0001	ADV_DIRECT_IND	Connectable directed advertising event

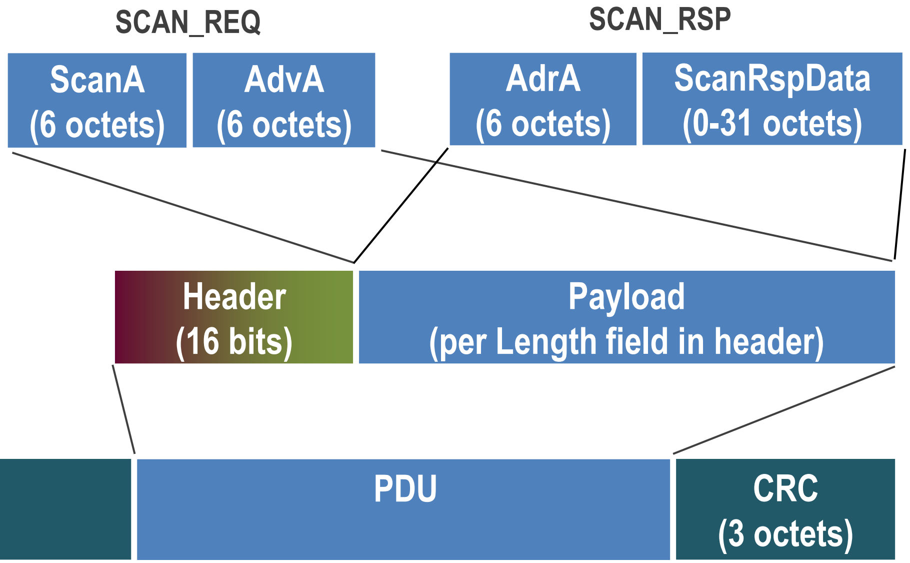
- **AdvA** – contains advertiser's public address if TxAdd=0, or a random address if TxAdd=1
- **InitA** – contains initiator's public address if RxAdd=0, or random address if RxAdd=1



Air Interface Packets – Scanning PDUs

Type	Packet	Usage
0011	SCAN_REQ	Scan request for further information from advertiser
0100	SCAN_RSP	Response to scan request from scanner

- **ScanA** – contains Scanner’s public or random device identified by TxAdd
- **AdvA** – contains Advertiser’s public or random device address identified by TxAdd/RxAdd
- **ScanRspData** – data from advertiser’s host

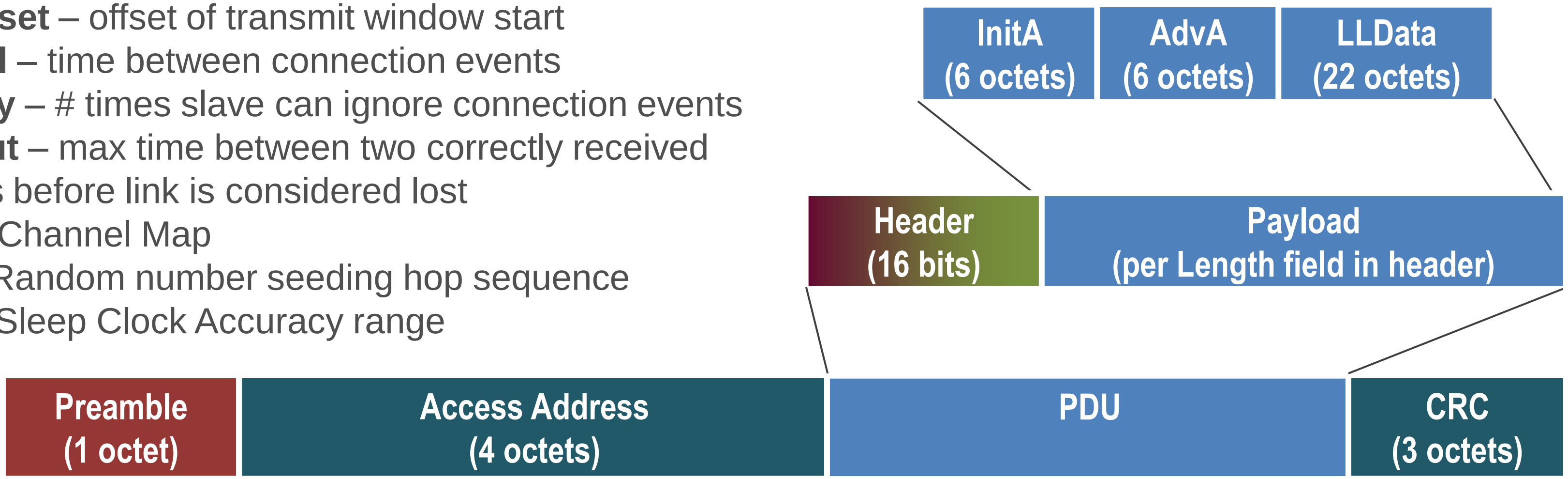


Air Interface Packets – Initiating PDUs

Type	Packet	Usage
0101	CONNECT_REQ	Connect request by Initiator

AA	CRCInit	WinSize	WinOffset	Interval	Latency	Timeout	ChM	Hop	SCA
(4 octets)	(3 octets)	(1 octets)	(2 octets)	(2 octets)	(2 octets)	(2 octets)	(5 octets)	(5 bits)	(3 bits)

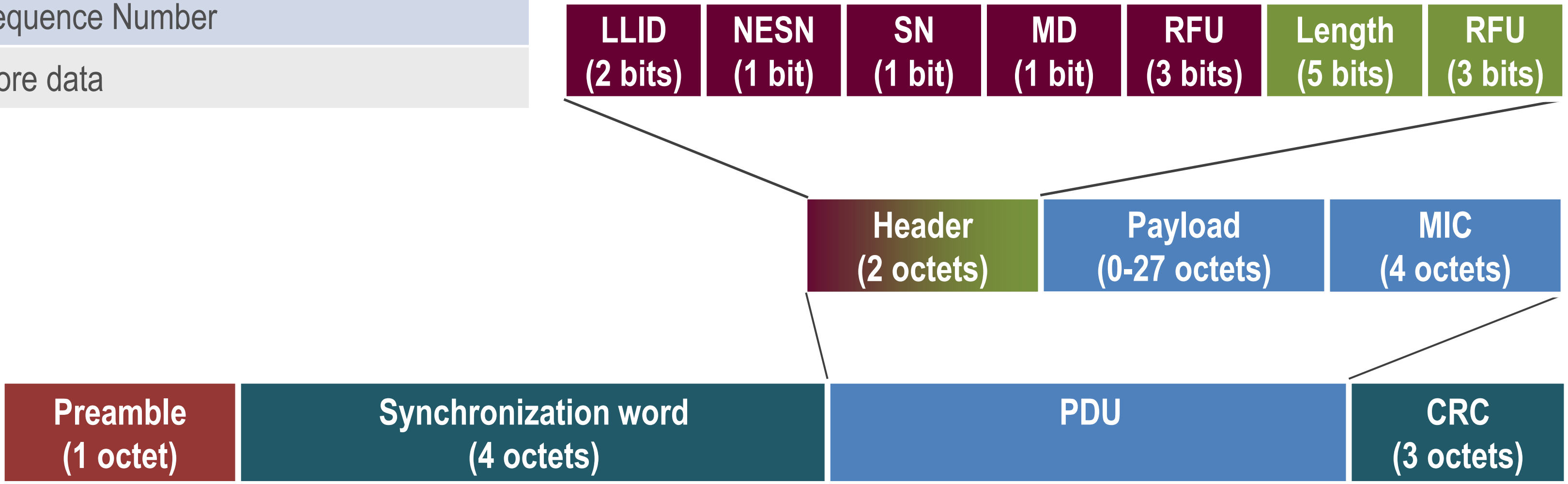
- **InitA** –initiator’s public/random address based on TxAdd
- **AdvA** –advertiser’s public/random address based on RxAdd
- **AA** – contains Link Layer’s connection address
- **CRCInit** –initialization value for CRC calculation
- **WinSize** – defines timing window for first data packet
- **WinOffset** – offset of transmit window start
- **Interval** – time between connection events
- **Latency** – # times slave can ignore connection events
- **Timeout** – max time between two correctly received packets before link is considered lost
- **ChM** – Channel Map
- **Hop** – Random number seeding hop sequence
- **SCA** – Sleep Clock Accuracy range



Air Interface Packets – LL Data Channel

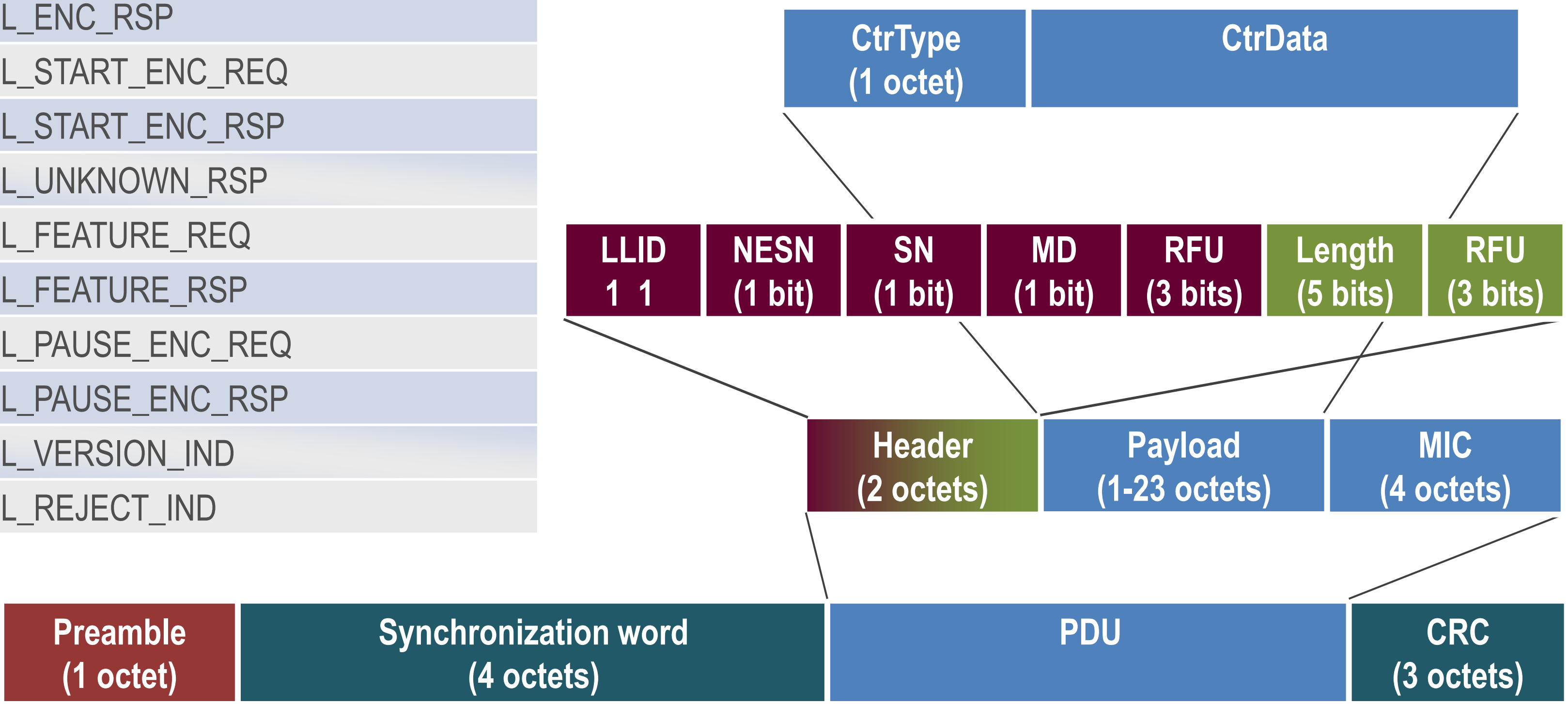
Field	Purpose and Encoding
LLID	**0x01** = Continuation/empty L2CAP packet **0x02** = Start of an L2CAP packet **0x03** = LL Control packet
NESN	Next Expected Sequence Number
SN	Sequence Number
MD	More data

- **Preamble** – frequency synchronization and AGC training (`01010101`) or (`10101010`)
- **Synchronization word** – 32 bit link layer connection access address
- **CRC** – computed over PDU
- **MIC** – Message Integrity Code, for use with encrypted links



Air Interface Packets – LL Control Packets

Opcode	Control packet name
0x00	LL_CONNECTION_UPDATE_REQ
0x01	LL_CHANNEL_MAP_REQ
0x02	LL_TERMINATE_IND
0x03	LL_ENC_REQ
0x04	LL_ENC_RSP
0x05	LL_START_ENC_REQ
0x06	LL_START_ENC_RSP
0x07	LL_UNKNOWN_RSP
0x08	LL_FEATURE_REQ
0x09	LL_FEATURE_RSP
0x0a	LL_PAUSE_ENC_REQ
0x0b	LL_PAUSE_ENC_RSP
0x0c	LL_VERSION_IND
0x0d	LL_REJECT_IND



Packet Timings

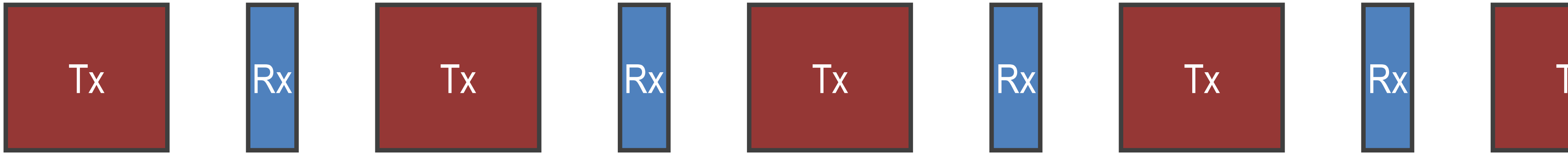
- Peer device transmits 150 µs after last packet

- Minimum size packet = 80 µs

(Preamble + Access Address + Header + CRC)

- Maximum size packet = 328 µs

(Preamble + Access Address + Header + Payload + MIC + CRC)



Tx Rx Tx Rx Tx Rx Tx Rx T

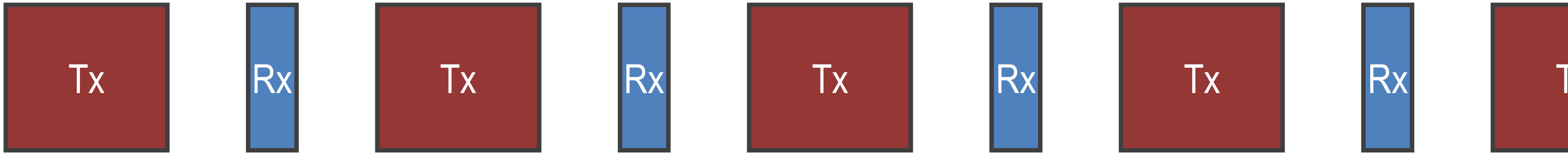
Maximum Data Rate

- Asymmetric Tx/Rx Packet Sequence

  328 + 150 + 80 + 150 = 708 µs

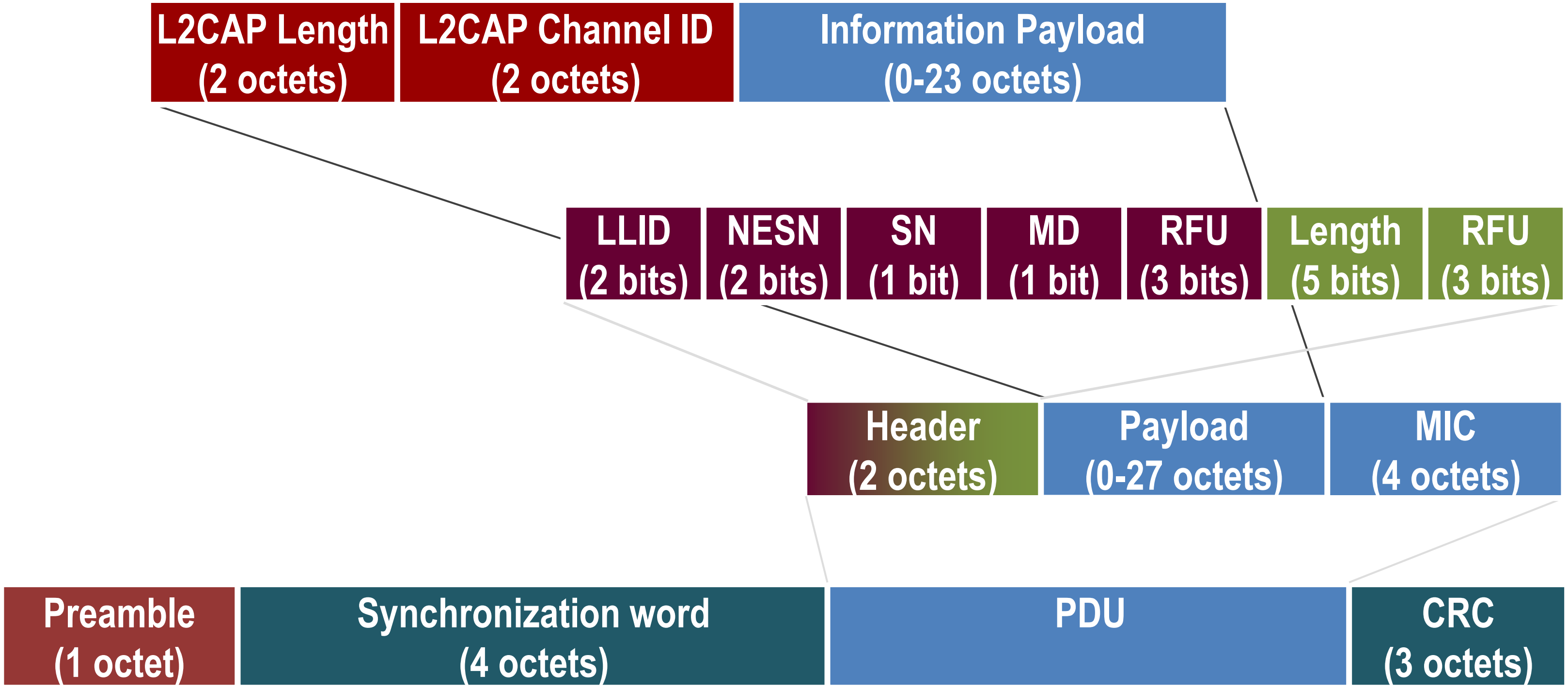
Transmitting 27 octets of application data

~305 kbps



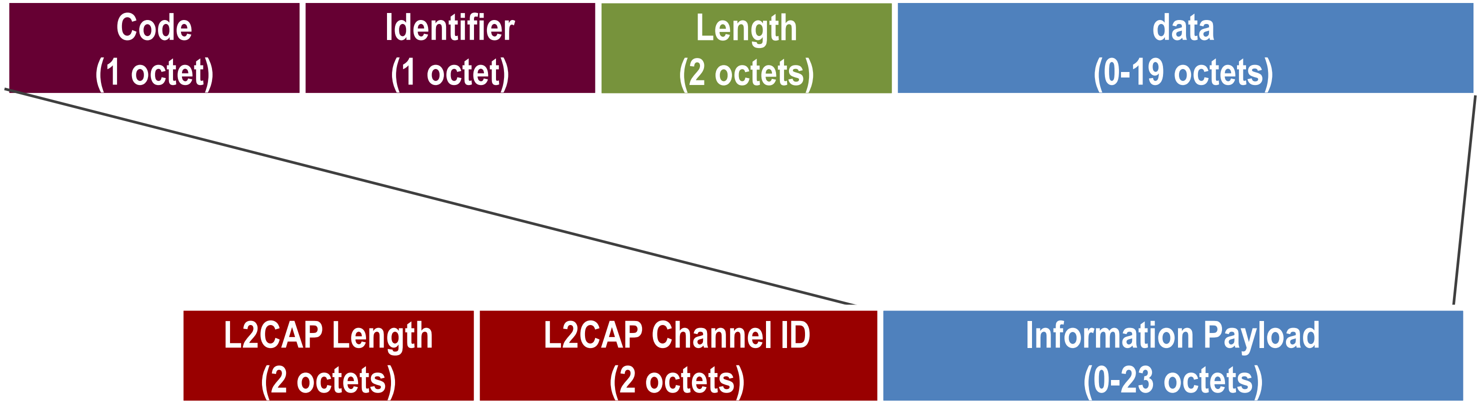
Logical Link Control and Adaptation Protocol (L2CAP)

- Provides fixed channel data services to upper layer protocols
- Provides protocol multiplexing capability through concept of channels
  - Channel Identifier is local name representing a logical channel
  - Channels are bi-directional

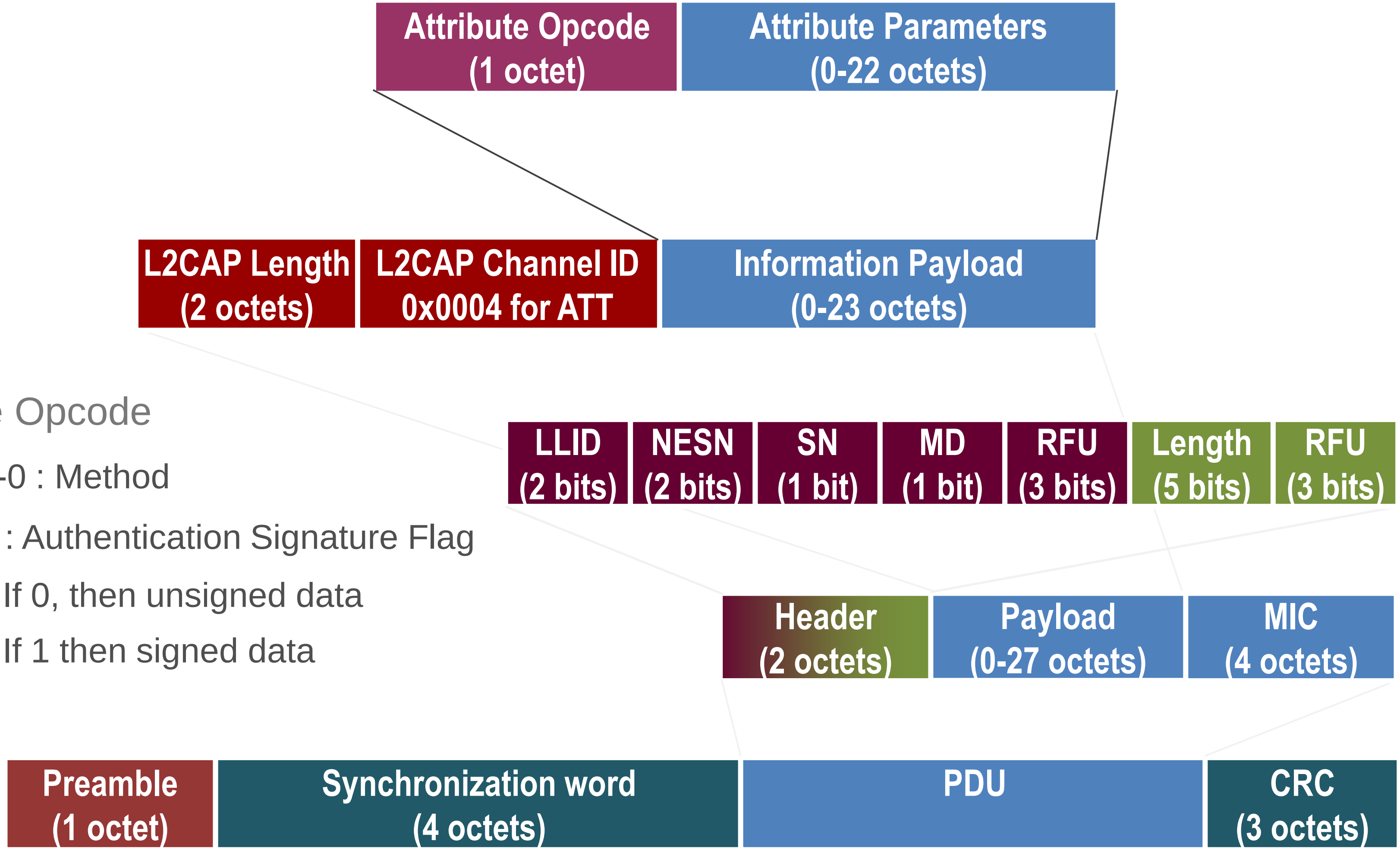


L2CAP Signaling for Bluetooth LE

Code	Description	Usage
0x00	Reserved	Reserved
0x01	Command Reject	Sent in response when unknown command code or inappropriate response
0x12	Connection Parameter Update Request	Allows slave device to request new connection parameter targets
0x13	Connection Parameter Update Response	Master response to slave Connection Parameter Update Request



Attribute PDU format



- Attribute Opcode
  - bit 6-0 : Method
  - bit 7 : Authentication Signature Flag
    - If 0, then unsigned data
    - If 1 then signed data

Questions

Developer Initiative at SIG

- Developer.bluetooth.org
  - Monthly Webinars
  - Quick Start Kit
  - Developer Forums
  - GATT Adopted specifications
  - Training Videos
- Your participation is a key
- Follow us on
  - Twitter – @BluetoothSIGDev
  - LinkedIn – BluetoothSIGDeveloper

Additional Information and Training

- Bluetooth low energy specification can be found on the Bluetooth website,

  www.bluetooth.org/Technical/Specifications/adopted.htm

  http://developer.bluetooth.org/gatt/

  http://developer.bluetooth.org

- Online training is also available on the Bluetooth website,

  http://developer.bluetooth.org/KnowledgeCenter/Pages/Training-Videos.aspx

Bluetooth®

SIMPLE. SECURE. EVERYWHERE.

Creating Android Applications for Today’s Bluetooth Devices

Second Half

Last Session Recap

- Bluetooth Technology Evolution
- Architectural Overview
- Stack Architecture
  - Physical Layer
  - Link Layer
  - HCI Layer
  - L2CAP Layer
  - Security Manager Protocol
  - Attribute Protocol
  - Generic Attribute Profile
  - Generic Access Profile
  - Applications
- Air Interface Packet Structure

Agenda

- Development Tools
- BlueZ stack
- Android bluetooth package
- Android Third part packges provided by Motorola and Broadcom
- Source walk through
- Capture Traces using Ellisys Sniffer

Development Tools

BlueZ stack

Android bluetooth package

Android Third Party Bluetooth Low Energy packge – Motorola Mobility

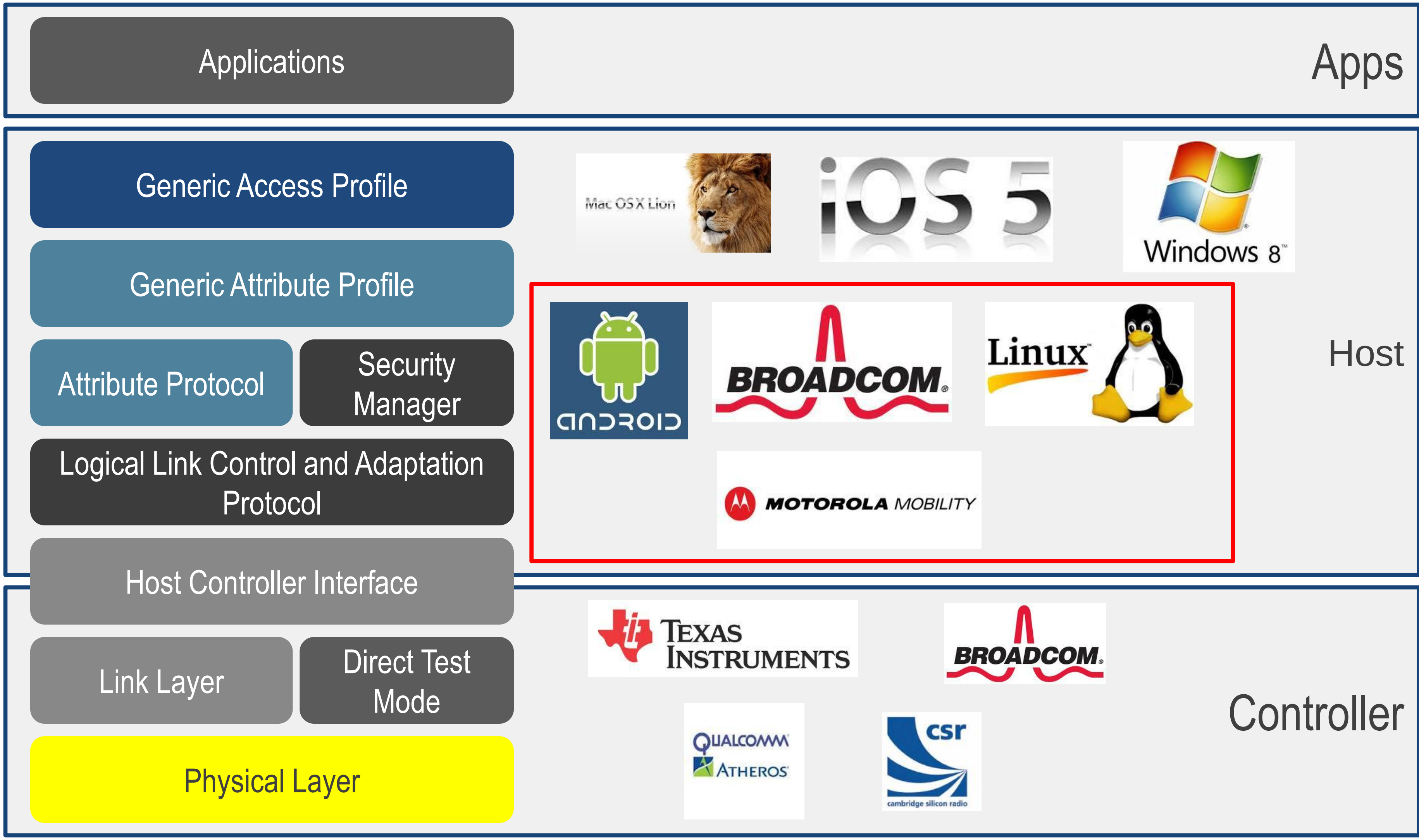
Android Third Party Bluetooth Low Energy packge - Broadcom

Source walk through

Capture Traces using Ellisys Sniffer

Development Tools

SMART READY Platforms



Applications

Apps

Generic Access Profile

Generic Attribute Profile

Attribute Protocol

Security Manager

Logical Link Control and Adaptation Protocol

Host

Host Controller Interface

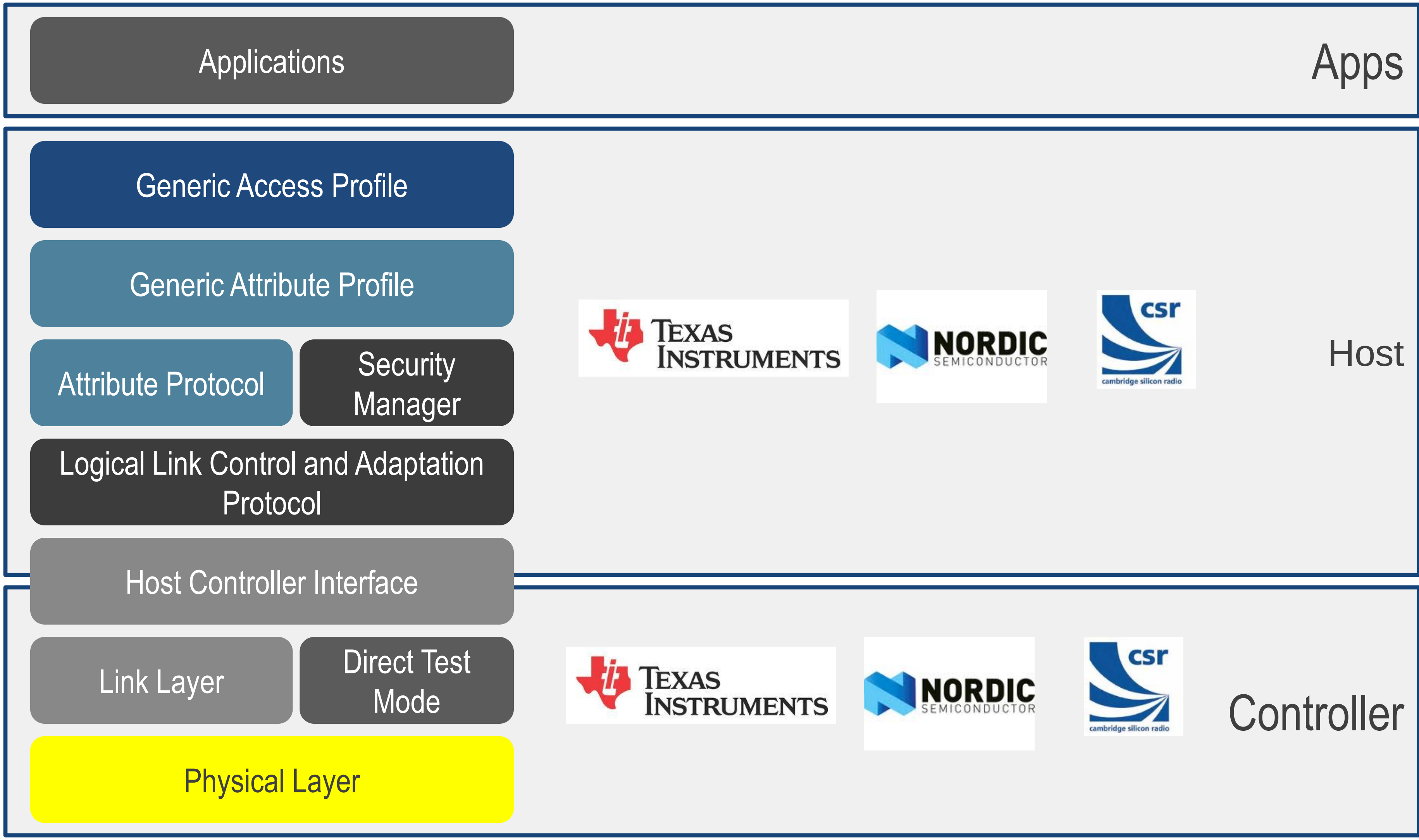
Link Layer

Direct Test Mode

Physical Layer

Controller

SMART Platforms



Applications

Apps

Generic Access Profile

Generic Attribute Profile

Attribute Protocol

Security Manager

Logical Link Control and Adaptation Protocol

Host Controller Interface

Host

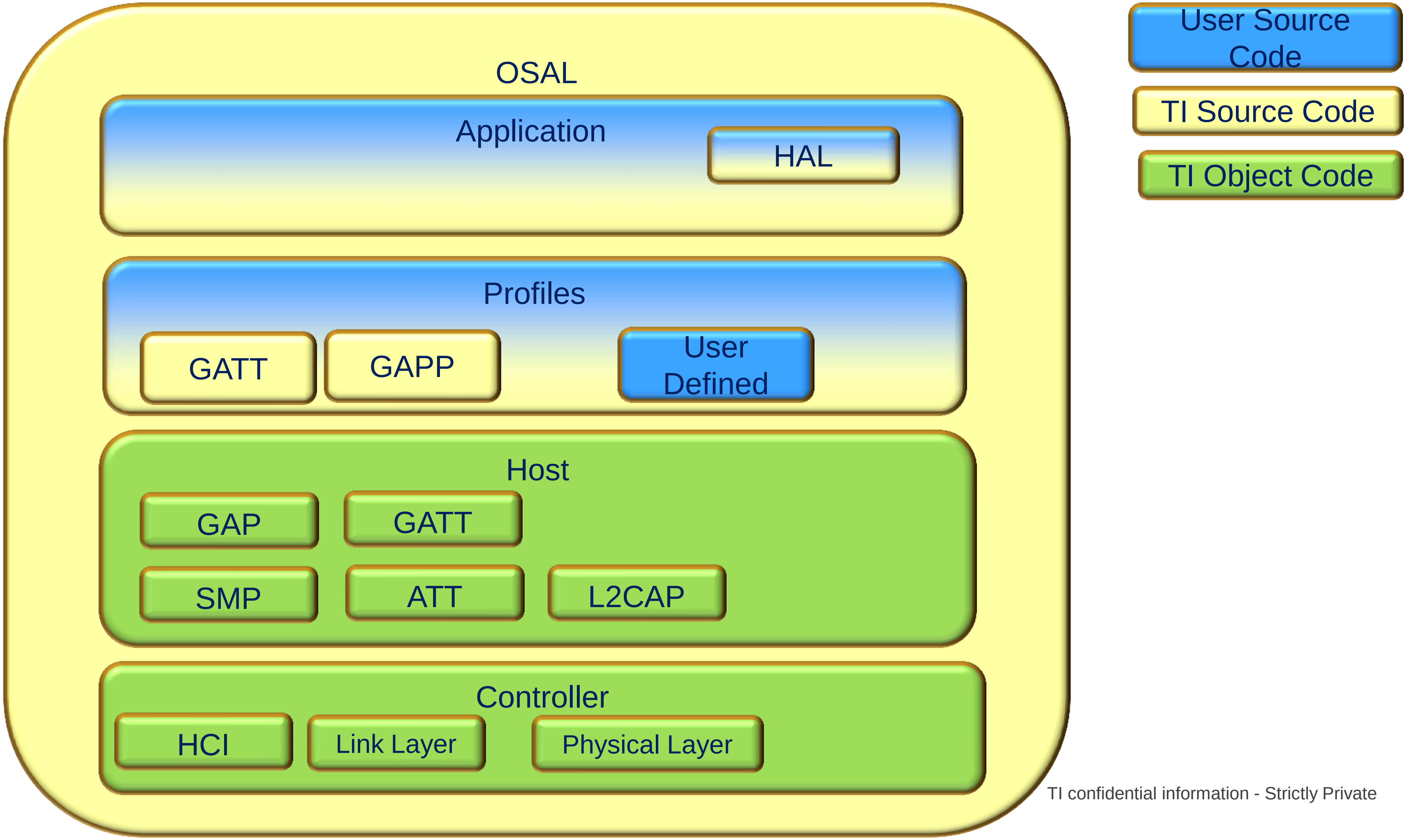
Link Layer

Direct Test Mode

Physical Layer

Controller

SMART PLATFORM - TI CC2540 SDK



TI confidential information - Strictly Private

TI CC2540DK-MINI Hardware

Debugger

- Works with keyfob and USB dongle
- Supports IAR and TI flash programmer

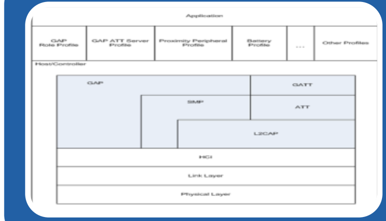
CC2540 Keyfob

- Powered by CR2032 coin cell battery
- LED, buttons, buzzer, accelerometer
- Usually acts as peripheral, application is on chip.

USB Dongle

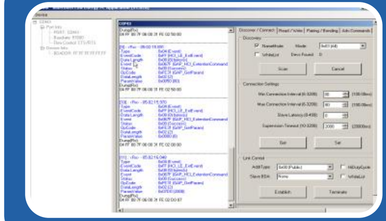
- Use Btool.exe to or custom app to send HCI commands.
- Usually acts as master (cell phone)

TI CC2540DK-MINI Software



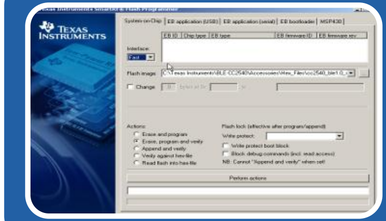
Stack Libraries

- Royalty free
- Full qualification
- Example Projects



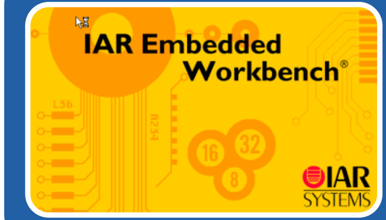
Btool Application

- Drives USB dongle with HCI commands
- Scan for devices, connect, authentication
- Log messages



SmartRF Flash Programmer

- Can flash CC2540
- Change address on device



IAR Compiler and IDE

- Robust 8051 compiler with CC2540 support.
- 30 day free evaluation

BlueZ – Setup & Configuration

- BlueZ – Linux official Bluetooth Stack
  - PTS Dongle
  - BlueZ source code

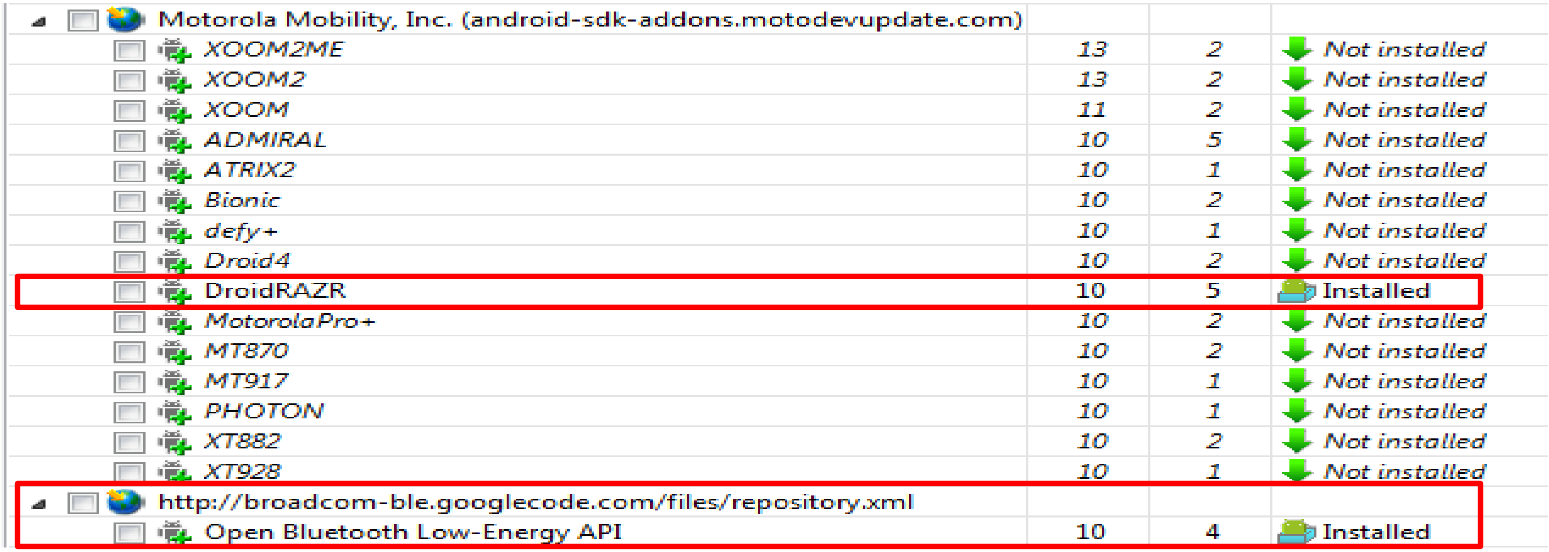
- Download Soure Code
  - git clone http://git.kernel.org/pub/scm/bluetooth/bluez.git
  - Resolving Dependencies: sudo apt-get build-dep bluez
  - Update the tree - Git pull
- To turn on LE, go to src/main.conf and modify the two parameters
  - EnableLE = true
  - AtrributeServer = true

BlueZ – Setup & Configuration (Contd)

- Configuring the PTS module
  - Bccmd – bccmd issues Blue Core commands to CSR devices.
  - Download the latest PSR file and install it on the device using bccmd
- Example
  - Sudo bccmd –d hci1 psload LE-Dongle…psr
- Building Source Code
  - ./bootstrap-configure && make
- Starting the latest built bluetooth daemon
  - sudo /etc/init.d/bluetooth stop
  - sudo src/bluetoothd -n –d

Android Development Environment

- Eclipse
- Android Packages
  - Android SDK API Level 10 - android.bluetooth.
  - Motorola Add-On for Droid Razr – API Level 10 revision 5 - package com.motorola.bluetooth.ble
  - Broadcom Add-On for Open Bluetooth – API Level 10 revision 4 - com.broadcom.bt.ble.
- Droid Razr phone



◢ □ Motorola Mobility, Inc. (android-sdk-addons.motodevupdate.com)			
□ XOOM2ME	13	2	Not installed
□ XOOM2	13	2	Not installed
□ XOOM	11	2	Not installed
□ ADMIRAL	10	5	Not installed
□ ATRIX2	10	1	Not installed
□ Bionic	10	2	Not installed
□ defy+	10	1	Not installed
□ Droid4	10	2	Not installed
□ DroidRAZR	10	5	Installed
□ MotorolaPro+	10	2	Not installed
□ MT870	10	2	Not installed
□ MT917	10	1	Not installed
□ PHOTON	10	1	Not installed
□ XT882	10	2	Not installed
□ XT928	10	1	Not installed
◢ □ http://broadcom-ble.googlecode.com/files/repository.xml			
□ Open Bluetooth Low-Energy API	10	4	Installed

Air Interface Sniffers

- Ellisys Bluetooth Explorer 400
- Frontline

Development Tools

BlueZ stack - www.bluez.org

Android bluetooth package - http://developer.android.com

Android Third Party Bluetooth Low Energy packge – Motorola Mobility

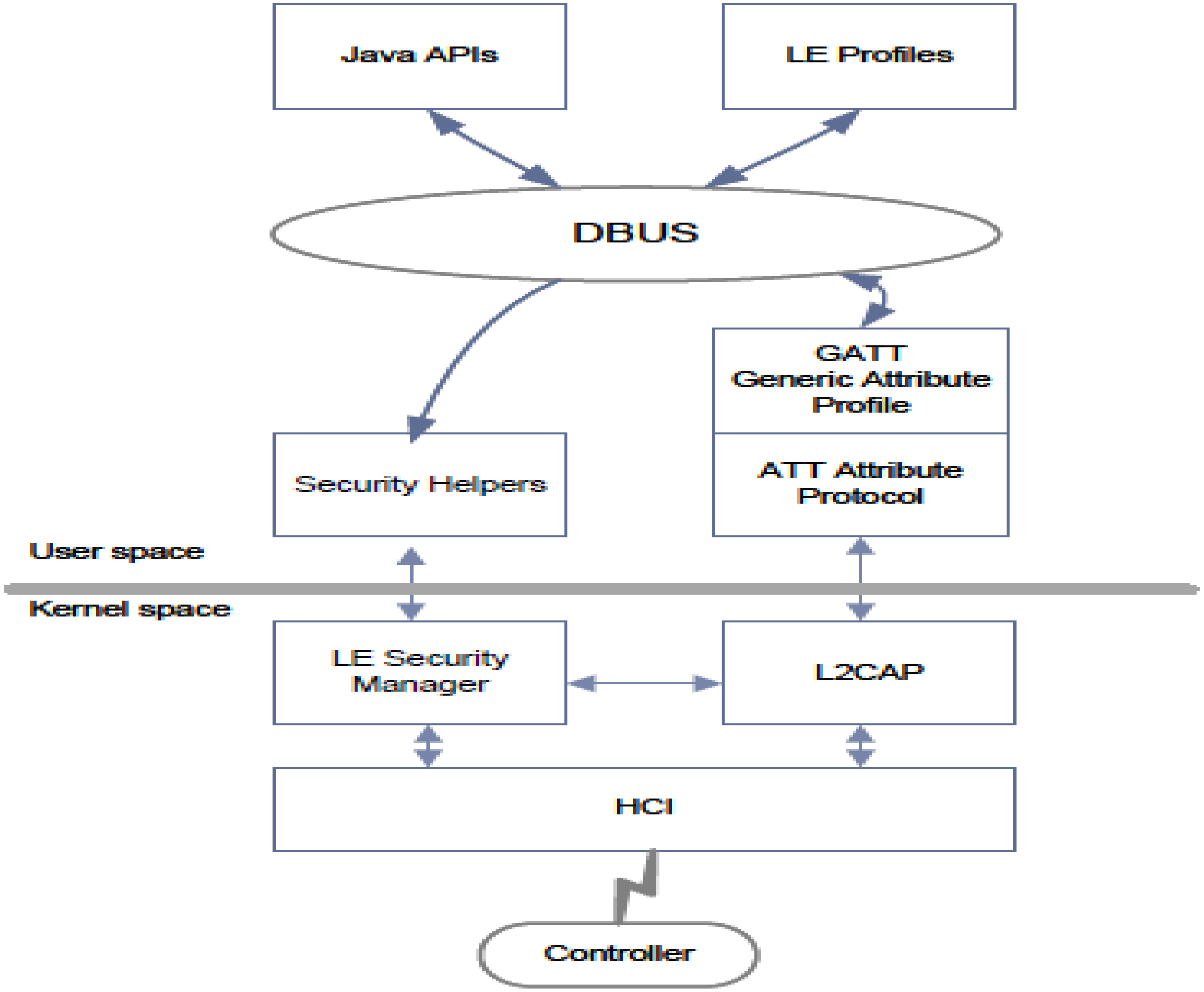
Android Third Party Bluetooth Low Energy packge - Broadcom

Source walk through

Capture Traces using Ellisys Sniffer

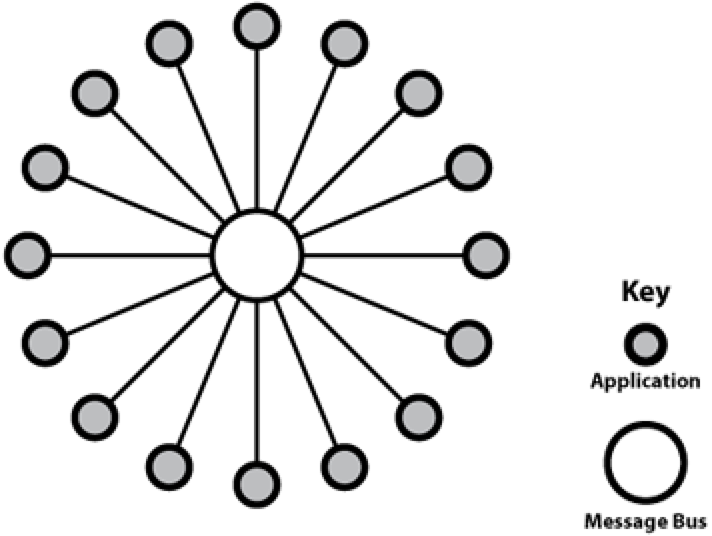
BlueZ Stack – Architecture Overview

BlueZ BLE Architecture



- BlueZ official bluetooth stack for linux and comes in all linux distributions
- Supports BR/EDR and Bluetooth Low Energy
- Multiple Processes linked together by DBUS
- Bluetooth Daemon runs in the background and handles DBUS core functionality
- Low Energy profiles run as pluggable linux modules.
- Java APIs hook to Android bluetooth packages via D-BUS

D-BUS Basics



- BlueZ uses System Bus
- 3 important parameters
  - Bus Address – bluez.org
  - Object Paths
  - Interfaces

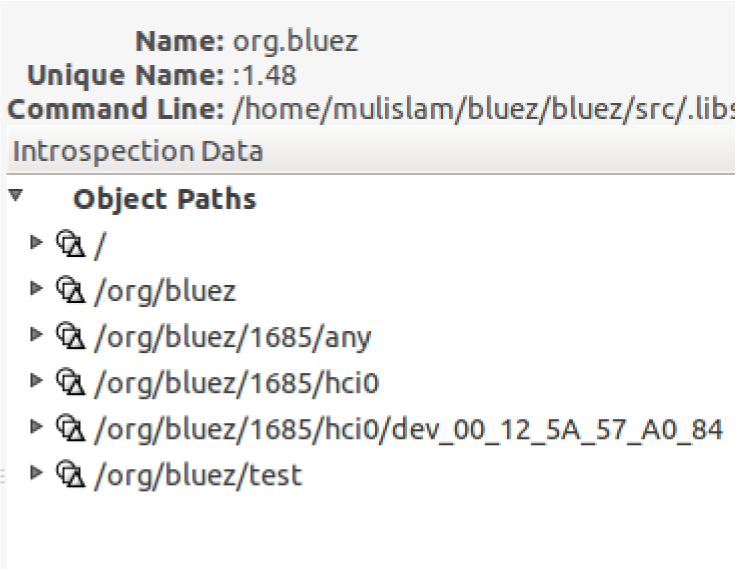
- D-BUS is an inter-process communication mechanism.
- Routes message from one end to the other end.
- Two types of buses
  - Session Bus – machine global bus. Single instance of the daemon with security restrictions
  - System Bus – Create for each user session

Interfaces - Methods & Signals

- Each interface consist of
- **Methods** - Exposes the functions that the object exposes to other objects
  - A method call in DBus consists of two messages; a method call message sent from process A to process B, and a matching method reply message sent from process B to process A. Both the call and the reply messages are routed through the bus daemon.
- **Signals –** Other objects can register to get notified by this object if they register for the signals. (Basically equivalent to callbacks)
  - Methods: Signals: A signal in DBus consists of a single message, sent by one process to any number of other processes. That is, a signal is a unidirectional broadcast.

Interfaces

- Interfaces and Methods can be added or removed at run time.
- BlueZ will use D-Bus signals to report found devices and services
- Example: After a connection is established, a new Device is added (captured using D-Feet tool)

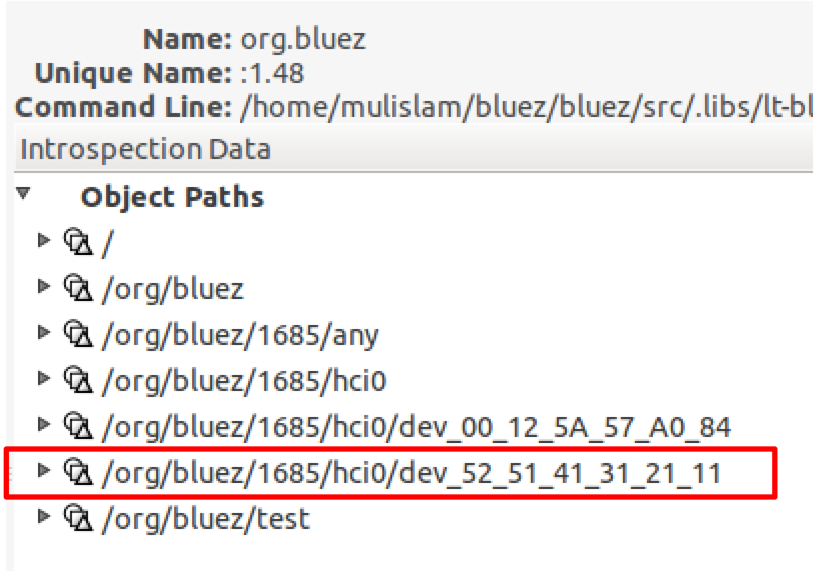


Name: org.bluez
Unique Name: :1.48
Command Line: /home/mulislam/bluez/bluez/src/.libs

Introspection Data

Object Paths

- /
- /org/bluez
- /org/bluez/1685/any
- /org/bluez/1685/hci0
- /org/bluez/1685/hci0/dev_00_12_5A_57_A0_84
- /org/bluez/test



Name: org.bluez
Unique Name: :1.48
Command Line: /home/mulislam/bluez/bluez/src/.libs/lt-bl

Introspection Data

Object Paths

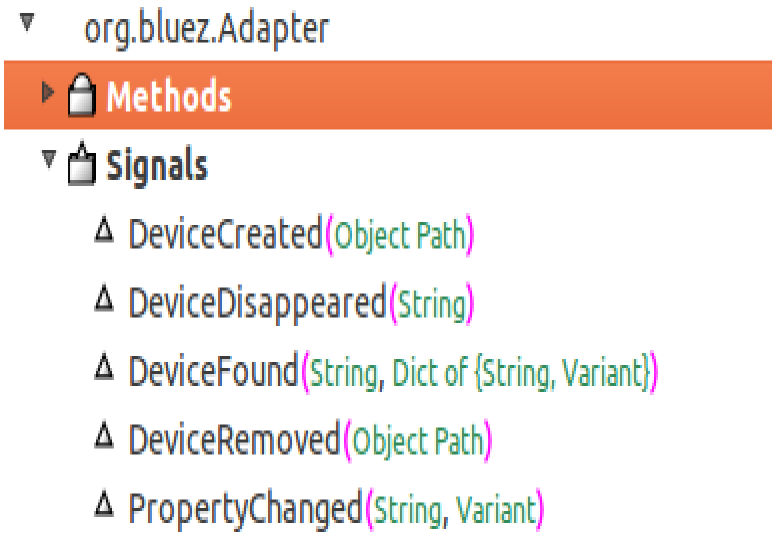
- /
- /org/bluez
- /org/bluez/1685/any
- /org/bluez/1685/hci0
- /org/bluez/1685/hci0/dev_00_12_5A_57_A0_84
- /org/bluez/1685/hci0/dev_52_51_41_31_21_11
- /org/bluez/test

BlueZ Interfaces

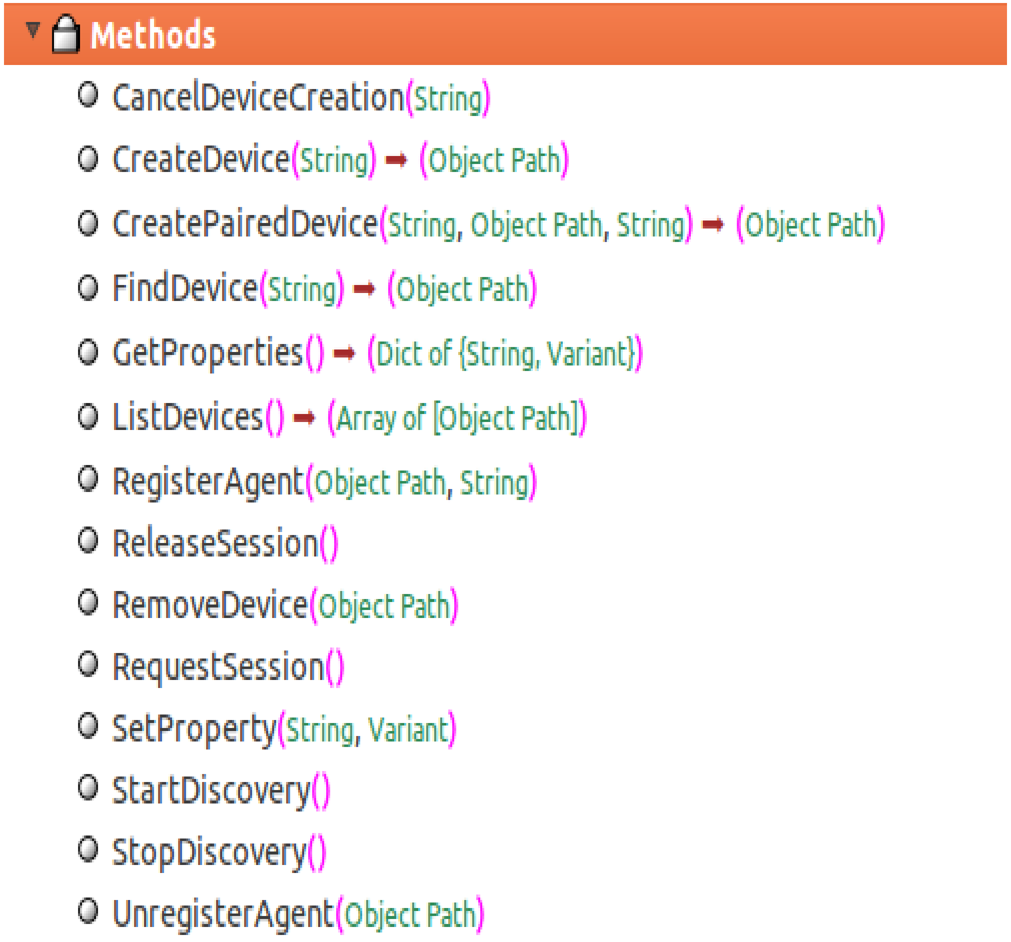
- Manager
  - **Methods**: DefaultAdapter(), FindAdapter(), ListAdapters()
  - **Signals**: AdapterAdded(object adapter), AdapterRemoved(object adapter), DefaultAdapterChanged(object adapter)
- Adapter
  - **Methods**: StartDiscovery(), StopDiscovery(), FindDevice(string address), CreateDevice(string address), RemoveDevice(object device)
  - **Signals**: DeviceFound(string address, dict values), DeviceDisappeared(string address), DeviceCreated(object device), DeviceRemoved(object device)
- Device
  - Methods: dict DiscoverServices(string pattern), void CancelDiscovery(), void Disconnect()
  - Signals: DisconnectRequested()
- Each profile is a pluggable module has its own interface

Interfaces – D-Feet tool

- **D-Feet:** a graphical D-Bus debugging tool.
- Here's an example of it monitoring org.bluez
- **Org.bluez.Adapter interface show below**



```
▼ org.bluez.Adapter
  ▶ Methods
  ▼ Signals
      DeviceCreated(Object Path)
      DeviceDisappeared(String)
      DeviceFound(String, Dict of {String, Variant})
      DeviceRemoved(Object Path)
      PropertyChanged(String, Variant)
```



```
▼ Methods
    CancelDeviceCreation(String)
    CreateDevice(String) ➡ (Object Path)
    CreatePairedDevice(String, Object Path, String) ➡ (Object Path)
    FindDevice(String) ➡ (Object Path)
    GetProperties() ➡ (Dict of {String, Variant})
    ListDevices() ➡ (Array of [Object Path])
    RegisterAgent(Object Path, String)
    ReleaseSession()
    RemoveDevice(Object Path)
    RequestSession()
    SetProperty(String, Variant)
    StartDiscovery()
    StopDiscovery()
    UnregisterAgent(Object Path)
```

Bluetooth Daemon

- Resides at /etc/init.d/bluetoothd
- Runs in the background and handles the addition/removal of interfaces
- Adds a new adapter when a Bluetooth controller is detected on your machine
- When your adapter is in a connected state, a new interface called “Device Interface” is created by the adapter

TOOLS

- BCCMD – Used to configure PTS(CSR) dongle
  - Example: sudo –d bccmd –hci0 *.psr
- Hciconfig - Used to configure Bluetooth devices.
  - Example: hciconfig –a hci0
- Hcitool - This tool is used to configure bluetooth connections
  - Example: sudo hcitool –i hci0 lescan
  - Example: sudo hcitool -i hci0 lecc <<BD_ADDR>
- Sdp - Provides the interface for doing queries on supported services on SDP server
  - Example: sdp add ServiceName– Adds new service
  - Example: sdp browse local – List all the services supported on your device
- Gatttool - Used to read/write characteristics
  - Example: ./gatttool -i hci1 -b C0:FF:EE:C0:FF:F1 -p 31 –characteristics
- Hcidump – Captures all packets going back and forth between host and controller

Code Walkthrough

Development Tools

BlueZ stack - www.bluez.org

Android bluetooth package - http://developer.android.com

Android Third Party Bluetooth Low Energy packge – Motorola Mobility

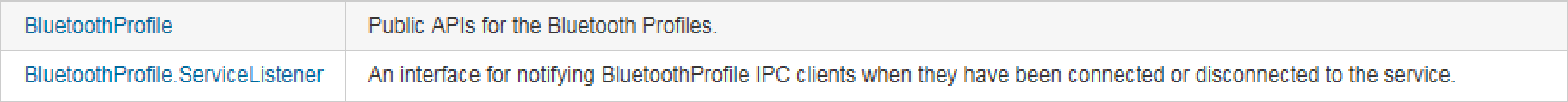
Android Third Party Bluetooth Low Energy packge - Broadcom

Source walk through

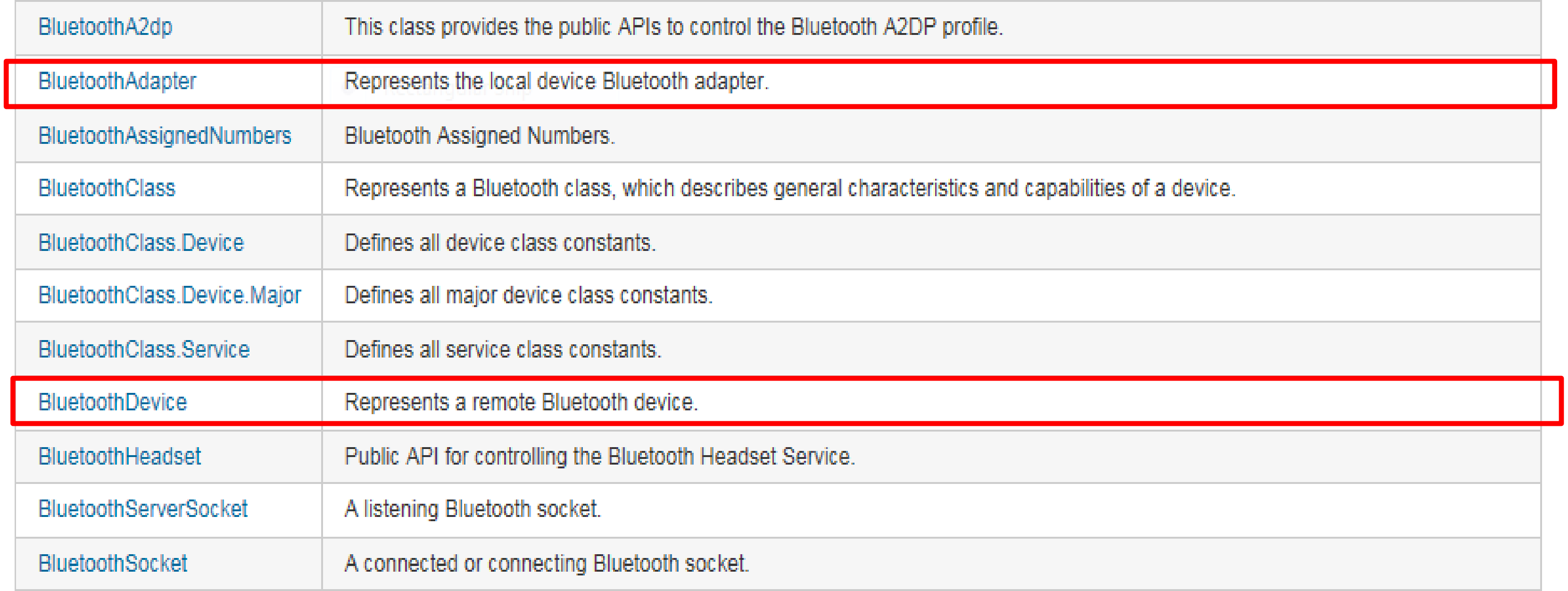
Capture Traces using Ellisys Sniffer

Android Bluetooth Package

Android. Bluetooth Interfaces & Classes



BluetoothProfile	Public APIs for the Bluetooth Profiles.
BluetoothProfile.ServiceListener	An interface for notifying BluetoothProfile IPC clients when they have been connected or disconnected to the service.



BluetoothA2dp	This class provides the public APIs to control the Bluetooth A2DP profile.
BluetoothAdapter	Represents the local device Bluetooth adapter.
BluetoothAssignedNumbers	Bluetooth Assigned Numbers.
BluetoothClass	Represents a Bluetooth class, which describes general characteristics and capabilities of a device.
BluetoothClass.Device	Defines all device class constants.
BluetoothClass.Device.Major	Defines all major device class constants.
BluetoothClass.Service	Defines all service class constants.
BluetoothDevice	Represents a remote Bluetooth device.
BluetoothHeadset	Public API for controlling the Bluetooth Headset Service.
BluetoothServerSocket	A listening Bluetooth socket.
BluetoothSocket	A connected or connecting Bluetooth socket.

Android Mainfest

- To access the APIs, the following items have to be added to the AndroidManifest.xml of your application:

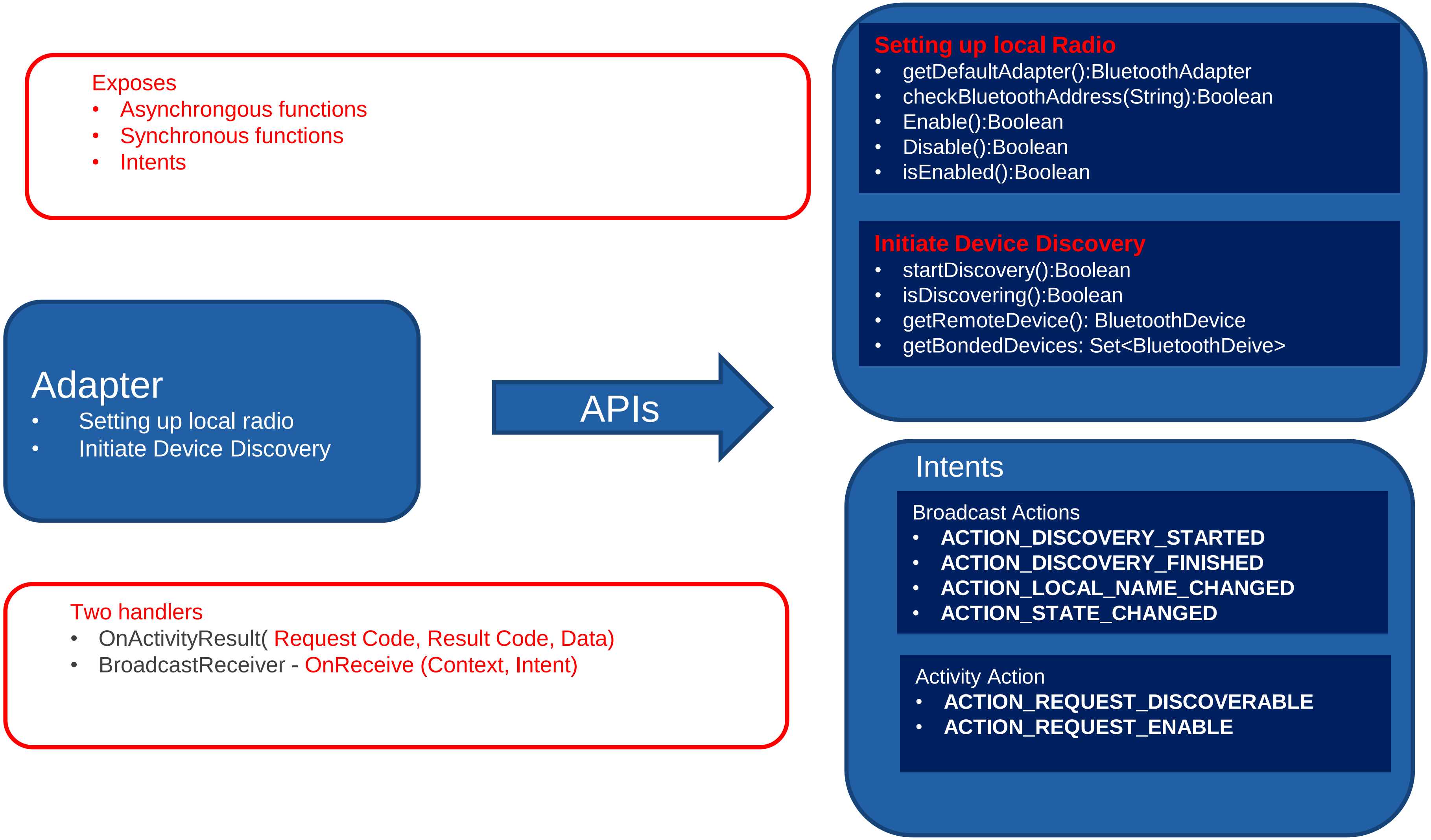
```
<!– Android Manifest.xml -->


<!-- Permission to use Bluetooth -->
<uses-permission android:name = "android.permission.BLUETOOTH" />

<!-- Add BLUETOOTH_ADMIN permissions -->
<uses-permission android:name = "android.permission.BLUETOOTH_ADMIN" />

<!-- Uses Library -->
<uses-library android:name = "com.broadcom.bt.le" android:required = "true" />
```

API Overview: android.bluetooth

- Android Bluetooth Package
  - Used to access BR/EDR and BLE stacks
  - We will be covering classes specific to accessing Bluetooth Low Energy Stack

- From a Low Energy perspective, we will look at the following classes
  - BluetoothAdapter: Represents the local Bluetooth adapter (Bluetooth radio).
  - BluetoothDevice: Represents a remote Bluetooth device.

Bluetooth Adapter



Exposes
- Asynchrongous functions
- Synchronous functions
- Intents

Adapter
- Setting up local radio
- Initiate Device Discovery

APIs

Setting up local Radio
- getDefaultAdapter():BluetoothAdapter
- checkBluetoothAddress(String):Boolean
- Enable():Boolean
- Disable():Boolean
- isEnabled():Boolean

Initiate Device Discovery
- startDiscovery():Boolean
- isDiscovering():Boolean
- getRemoteDevice(): BluetoothDevice
- getBondedDevices: Set<BluetoothDeive>

Intents

Broadcast Actions
- **ACTION_DISCOVERY_STARTED**
- **ACTION_DISCOVERY_FINISHED**
- **ACTION_LOCAL_NAME_CHANGED**
- **ACTION_STATE_CHANGED**

Activity Action
- **ACTION_REQUEST_DISCOVERABLE**
- **ACTION_REQUEST_ENABLE**

Two handlers
- OnActivityResult(Request Code, Result Code, Data)
- BroadcastReceiver - OnReceive (Context, Intent)

Development Tools

BlueZ stack

Android bluetooth package

Android Third Party Bluetooth Low Energy packge – Motorola Mobility

Android Third Party Bluetooth Low Energy packge - Broadcom

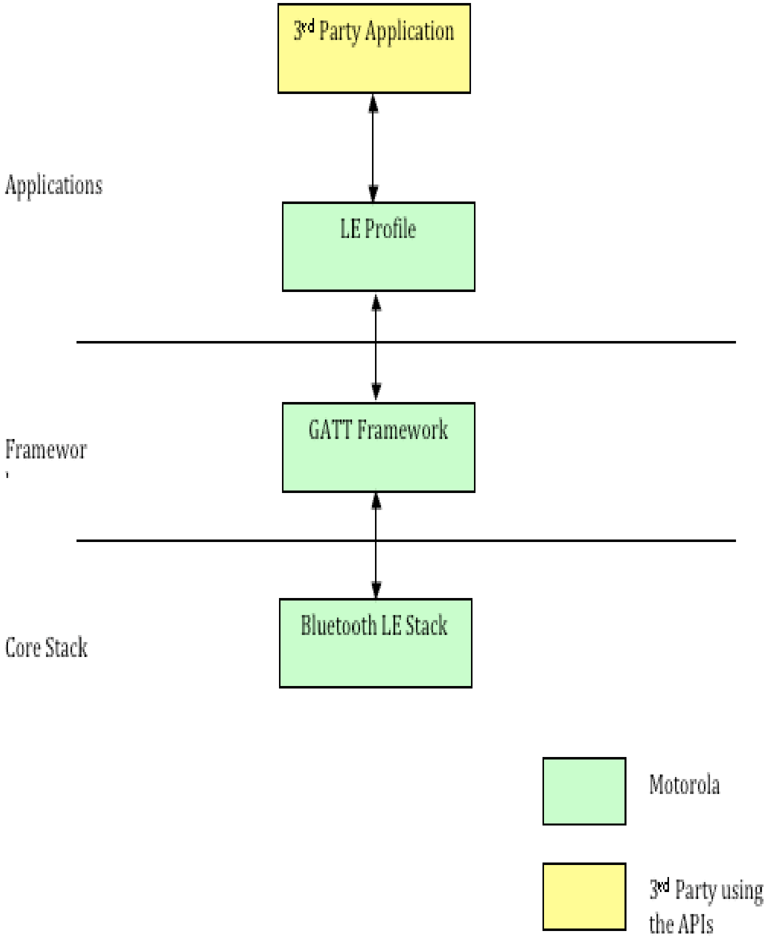
Source walk through

Capture Traces using Ellisys Sniffer

Motorola Bluetooth Low Energy GATT Framework API

Motorola Stack

Motorola's HRM API



3rd Party Application

Applications

LE Profile

Framewor
'

GATT Framework

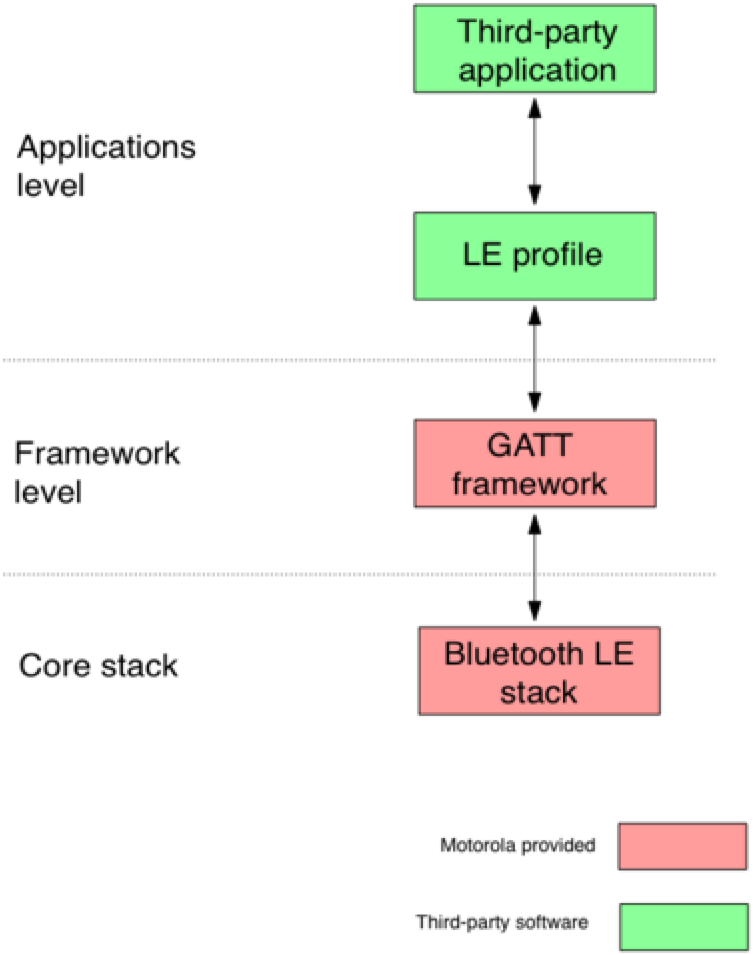
Core Stack

Bluetooth LE Stack

🟩 Motorola

🟨 3rd Party using the APIs

Motorola GATT API



Third-party application

Applications level

LE profile

Framework level

GATT framework

Core stack

Bluetooth LE stack

Motorola provided 🟥

Third-party software 🟩

API Overview: com.motorola.bluetooth.ble

- Motorola Mobility – com.motorola.bluetoothle
  - BluetoothGatt
    - BluetoothGatt(Context)
    - connectGatt(BluetoothDevice, String, IBluetoothGattCallback)
    - disconnectGatt(BluetoothDevice, String)
    - getGattCharacteristics(BluetoothDevice, String)
    - getGattCharacteristicValue(BluetoothDevice, String, String)
    - getGattPrimaryServices(BluetoothDevice)
    - readGattCharacteristics(BluetoothDevice, String)
    - readGattCharacteristicValue(BluetoothDevice, String, String)
    - writeGattCharacteristicValue(BluetoothDevice, String, String, byte[], int)
    - writeGattConfigurationDesc(BluetoothDevice, String, String, byte[], int)
  - IBluetoothGattCallback
    - indicationGattCb(BluetoothDevice, String, String, String[])
    - notificationGattCb(BluetoothDevice, String, String, byte[])

Development Tools

BlueZ stack

Android bluetooth package

Android Third Party Bluetooth Low Energy packge – Motorola Mobility

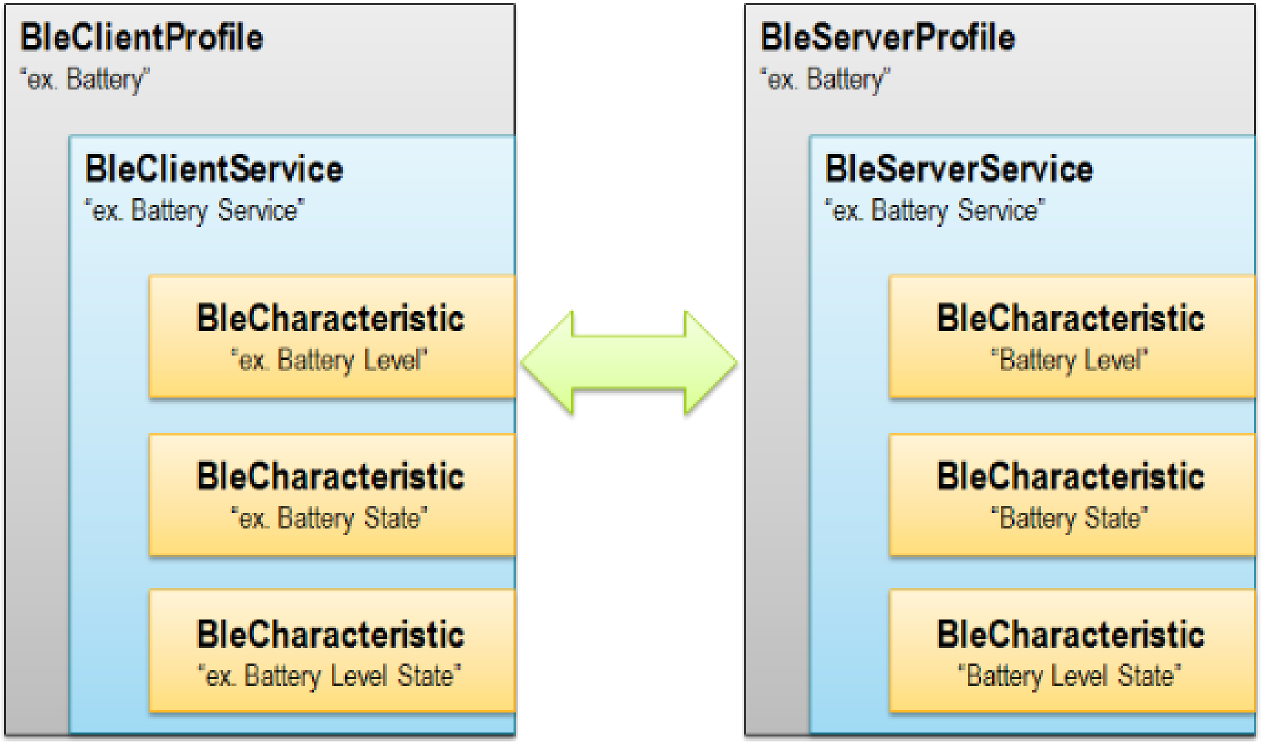
Android Third Party Bluetooth Low Energy packge - Broadcom

Source walk through

Capture Traces using Ellisys Sniffer

Broadcom Open *Bluetooth* Low Energy API

Overview


BleClientProfile
"ex. Battery"
BleClientService
"ex. Battery Service"
BleCharacteristic
"ex. Battery Level"
BleCharacteristic
"ex. Battery State"
BleCharacteristic
"ex. Battery Level State"
BleServerProfile
"ex. Battery"
BleServerService
"ex. Battery Service"
BleCharacteristic
"Battery Level"
BleCharacteristic
"Battery State"
BleCharacteristic
"Battery Level State"

API Overview: com.broadcom.bt.ble

- Client Side APIs
  - BleAdapter.class
  - BleClientProfile.class
  - BleClientService.class
  - BleClientConfig.class
- Utility APIs
  - BleApiHelper.class
  - BleConstants.class
  - BleGattID.class

- Server Side
  - BleServerProfile.class
  - BleServerService.class
  - BleServerConfig.class
  - BleAttribute.class
    - BleCharacteristic.class
    - BleDescriptor.class
    - BleUserDescriptor.class
      - BleExtProperty.class
      - BlePresentationFormat.class
    - BleUserDescription.class

Development Tools

BlueZ stack

Android bluetooth package

Android Third Party Bluetooth Low Energy packge – Motorola Mobility

Android Third Party Bluetooth Low Energy packge - Broadcom

Source walk through

Capture Traces using Ellisys Sniffer

Source Code Walkthrough

Android Bluetooth Package

- Tasks we want to accomplish
  - Scan for devices
  - Connecting with devices
  - Find Services & characteristics
  - Read and write data
  - Enable Notifications
  - Terminate connection

1 - Setting Up Bluetooth

```
// STEP 1: Get access to local Bluetooth radio
BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

if (mBluetoothAdapter == null) {
   // Device does not support Bluetooth
}

// STEP 2: Enable Bluetooth
if (!mBluetoothAdapter.isEnabled()) {
   // Asynchronous
   Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
   startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
}

// STEP 3: OnActivityResult
protected void onActivityResult(int requestCode, int resultCode, Intent data) {

   switch (requestCode) {
      case REQUEST_ENABLE_BT :
         if (resultCode == Activity.RESULT_OK) {
         // BT Enabled
         .....
         }
      break;
   }
}
```

2- Finding Devices

```
// STEP 1: Register for ACTION_FOUND intent
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
registerReceiver(mReceiver, filter); // Don't forget to unregister during onDestroy

// STEP 2: Start Discovery – Asynchronous function
 mAdapter.startDiscovery();

// STEP 3: Create a BroadcastReceiver for ACTION_FOUND
private final BroadcastReceiver mReceiver = new BroadcastReceiver() {

   public void onReceive(Context context, Intent intent) {
      String action = intent.getAction();

      // When discovery finds a device
      if (BluetoothDevice.ACTION_FOUND.equals(action)) {

         BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);

         if (BleAdapter.getDeviceType(device) == BleAdapter.DEVICE_TYPE_BLE)   {
            // Add the name and address to an array adapter to show in a ListView
            mArrayAdapter.add(device.getName() + "\n" + device.getAddress());
         }
      }
   }
};
```

3- Motorola: Discovering Services & Connect

```
// STEP 1: Create GATTService
private BluetoothGatt mGattService;
mGattService = new BluetoothGatt(this.getApplicationContext());

// STEP 2: Register Intents
filter_scan = new IntentFilter(btGatt.ACTION_GATT_CONNECTED);
filter_scan.addAction(btGatt.ACTION_GATT_DISCONNECTED);
filter_scan.addAction(btGatt.ACTION_GATT_CHARACTERISTICS_GET);
filter_scan.addAction(btGatt.ACTION_GATT_CHARACTERISTICS_READ );
filter_scan.addAction(btGatt.ACTION_GATT_CHARACTERISTICS_WRITE );
registerReceiver(mConn_Receiver, filter_scan);

// STEP 3: connect To GATT service
String[] primaryServices = mGattService.getGattPrimaryServices(device);
status = mGattService.connectGatt( BTDevice, serviceUUID, callbackNotifications_Indications);


//STEP 4:
BroadcastReceiver mConn_Receiver
    onReceive()- HANDLE the events below
     - ACTION_GATT_CHARACTERISTICS_READ
     - ACTION_GATT_CHARACTERISTICS_GET
     - ACTION_GATT_CONNECTED
     - ACTION_GATT_DISCONNECTED
*/
```

4- Motorola: Reading/Writing

```
// STEP 1: Enable Characteristic discovery
mGattService.readGattCharacteristics( BTDevice, serviceUUID);

// STEP 2: Save the handle returned on ACTION_GATT_CHARACTERISTICS_GET
BroadcastReceiver mReceiver:OnReceive() - ACTION_GATT_CHARACTERISTICS_GET


// STEP 3: Read value of the characteristics by passing the handle retrieved in STEP 2
mGattService.readGattCharacteristicValue( BTDevice, serviceUUID, char_handle_read);

// STEP 4: Returns the value of the characteristics
BroadcastReceiver mReceiver:OnReceive() - ACTION_GATT_CHARACTERISTICS_READ


// STEP 5: Write value to the characteristics
status = mGattService.writeGattCharacteristicValue( BTDevice, serviceUUID, char_handle_write,
data, 1);
status = mGattService.disconnectGatt( BTDevice, service);


// STEP 6: ACK for the Write operation
BroadcastReceiver mReceiver:OnReceive() - ACTION_GATT_CHARACTERISTICS_WRITE
```

5- Motorola: Notifications

```
// STEP 1: Override Notification and Indication Callbacks
private class BtGattCallback extends IBluetoothGattCallback.Stub {
     public void indicationGattCb(BluetoothDevice device, String service,
                                 String characterstic_handle, String[] data)
     {
        // handle the returned data here
     }

     public void notificationGattCb(BluetoothDevice device, String service,
                                    String characterstic_handle, byte[] data) {

        // handle the returned data here
     }
}

// NOTE: When connecting to GATT, we passed a callback. That was the notification callback
private BtGattCallback myCallback;
myCallback = new BtGattCallback();
status = mGattService.connectGatt(device, hrmUUID, myCallback);
```

3- Broadcom: Discovering Services

```
// ACTION_UUID

// STEP 1: Start service discovery for a remote device (example Bluetooth address used)
BleAdatper.getRemoteServices("00:11:22:33:44:55");

// STEP 2: Create a broadcast receiver for ACTION_UUID
private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
   public void onReceive(Context context, Intent intent) {
      String action = intent.getAction();

      // Evaluate service discovery result
      if (BleAdapter.ACTION_UUID.equals(action)) {
        Bundle bundle = intent.getExtras();
        Parcelable[] uuids = bundle.getParcelableArray(BleAdapter.EXTRA_UUID);

         for( int i = 0; i != uuids.length; ++i ) {
            ParcelUuid uuid = (ParcelUuid) uuids[i];
            // Access desired services ...
         }
      }
   }
};
```

3- Broadcom: Connect to a profile/service

```
// NOTE: In this example we are showing how to access Immediate Alert Service
// NOTE: Classes used are BleClientProfile & BleClientService

// - STEP 1: Create Client Side Service and overwrite required callbacks
public class ImmediateAlertService extends BleClientService {
   // UUID from the Bluetooth Assigned Numbers
   static public BleGattID myUuid = new BleGattID("00001802-0000-1000-8000-00805f9b34fb");  public

   ImmediateAlertService() {
      super(myUuid);
   }
   ...
}

//- STEP 2: Create FindMe profile and include the Immediate Alert service created above and implement the required
functions

public class FindMeProfile extends BleClientProfile {

   // Unique UUID used to register the profile with the Bluetooth stack
   static BleGattID myUuid = new BleGattID("015f613f-fe1d-475b-b0da-dd947ead9c2d");

   // Service(s) used by this profile ImmediateAlertService
   mImmediateAlertService = new ImmediateAlertService();
   public ArrayList mServices = new ArrayList();

   // Constructor
   public FindMeProfile(Context ctxt) {
      super(ctxt, myUuid);
      mServices.add(mImmediateAlertService);
      init(mServices, null);
   }

   public void onDeviceConnected(BluetoothDevice device) { .. .. }
   public void onRefreshed(BluetoothDevice device) { ... .}
   ...
}
```

3b - Connect to a profile/service (Contd)

```
// STEP 1: Get a Bluetooth device (samble Bluetooth Address used)
BluetoothDevice btDev =
   BluetoothAdapter.getDefaultAdapter().getRemoteDevice("00:11:22:33:44:55");

// STEP 2: Connect to the remote device
FindMeProfile mFindMeProfile = new FindMeProfile(this);
mFindMeProfile.connect(btDev);

// STEP 3: Override the OnDeviceConnected function of BleClientProfile
// Refresh method will read all the values on the server and update its local copy
public void onDeviceConnected(BluetoothDevice device) {
   // Refresh services and characteristics
   mFindMeProfile.refresh(device);
}
```

4- Read/Write

```
// NOTE: Refresh function of BleClientProfile is used to update the local copy

// STEP 1: Override OnRefreshed function.
// STEP 2: In order to write, use the writeCharacteristic function of BleClientProfile
public void onRefreshed(BluetoothDevice device) {
   // Get the AlertLevel characteristic object from the service
   BleCharacteristic alertLevelCharacteristic = mImmediateAlertService.getCharacteristic(
                                 device, new BleGattID(ALERT_LEVEL_CHARACTERISTIC_UUID));
   // Assign a new value
   byte[] value = { FindMeProfileClient.ALERT_LEVEL_HIGH };
   alertLevelCharacteristic.setValue(value);

   // Write the characteristic
   mImmediateAlertService.writeCharacteristic(device, 0, alertLevelCharacteristic);
}
```

5- Client Notifications

//STEP 1: Turn on Notifications on the server
BleClientService has a method registerForNotification (BluetoothDevice remoteDevice, int instanceID, BleGattID characteristicID)

```
// STEP 2:  Example: Register a broadcast receiver that receives internal battery alerts
registerReceiver( mAlertLevelReceiver )

...

// Notify all clients by calling the BleServerService.updateCharacteristic() method

private final BroadcastReceiver mAlertLevelReceiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
      String alertLevel = intent.getParcelableExtra(ALERT_LEVEL);

      alertChar.setValue(alertLevel.getBytes());

      mLinkLossService.updateCharacteristic(alertChar);
   }
}

public void onDeviceConnected(BluetoothDevice device) {
   // Refresh services and characteristics
   mFindMeProfile.refresh(device);
}
```

Source Walkthrough

Demo using TI CC2540 & Android App on MotorRazr

Development Tools

BlueZ stack

Android bluetooth package

Android Third Party Bluetooth Low Energy packge – Motorola Mobility

Android Third Party Bluetooth Low Energy packge - Broadcom

Source walk through

Capture Traces using Ellisys Sniffer

Captured Air Interface Trace Analysis

Questions

Developer Initiative at Bluetooth SIG

Developer Initiative at SIG

- Developer.bluetooth.org
  - Monthly Webinars
  - Quick Start Kit
  - Forums
  - GATT Adopted specifications
  - Training Videos
- Your participation is a key
- Follow us on
  - Twitter – @BluetoothSIGDev
  - LinkedIn – BluetoothSIGDeveloper

Additional Information and Training

- Bluetooth low energy specification can be found on the Bluetooth website,

  www.bluetooth.org/Technical/Specifications/adopted.htm

  http://developer.bluetooth.org/gatt/

  http://developer.bluetooth.org

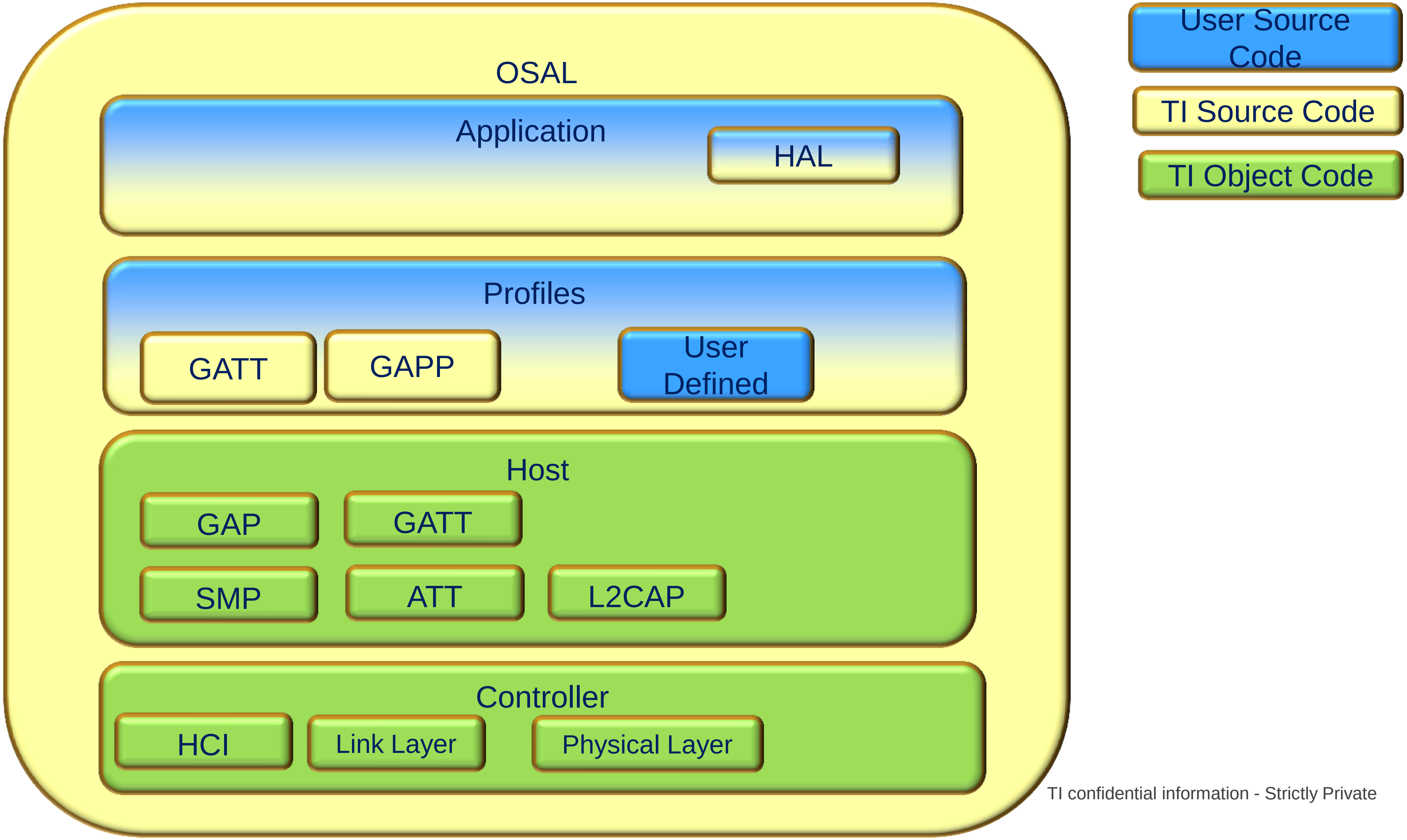
- Online training is also available on the Bluetooth website,

  http://developer.bluetooth.org/KnowledgeCenter/Pages/Training-Videos.aspx

Bluetooth®

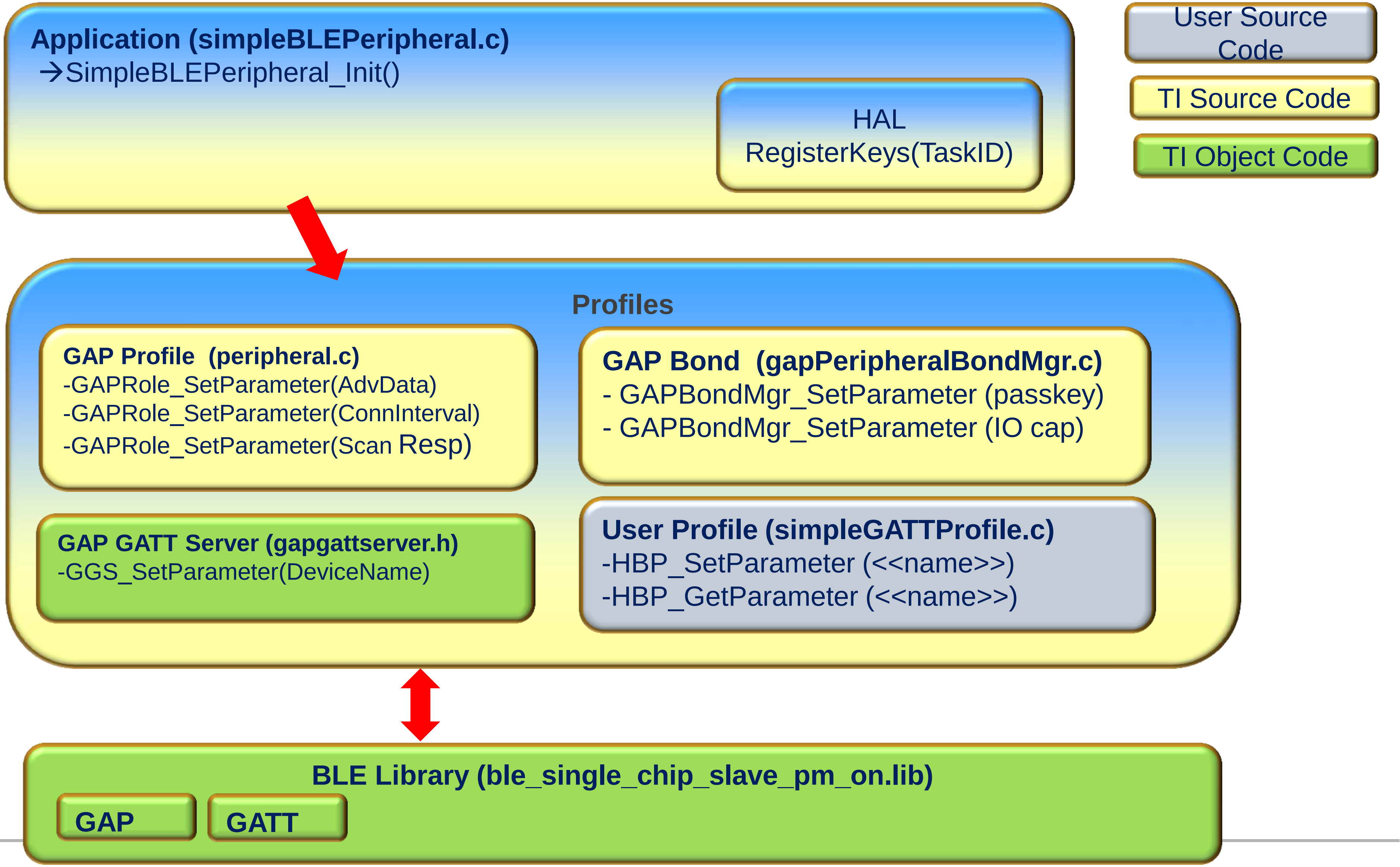
SIMPLE. SECURE. EVERYWHERE.

TI CC2540 SDK



TI confidential information - Strictly Private

Application Startup (set values)



Application (simpleBLEPeripheral.c)
→SimpleBLEPeripheral_Init()

HAL
RegisterKeys(TaskID)

User Source Code

TI Source Code

TI Object Code

Profiles

GAP Profile (peripheral.c)
-GAPRole_SetParameter(AdvData)
-GAPRole_SetParameter(ConnInterval)
-GAPRole_SetParameter(Scan Resp)

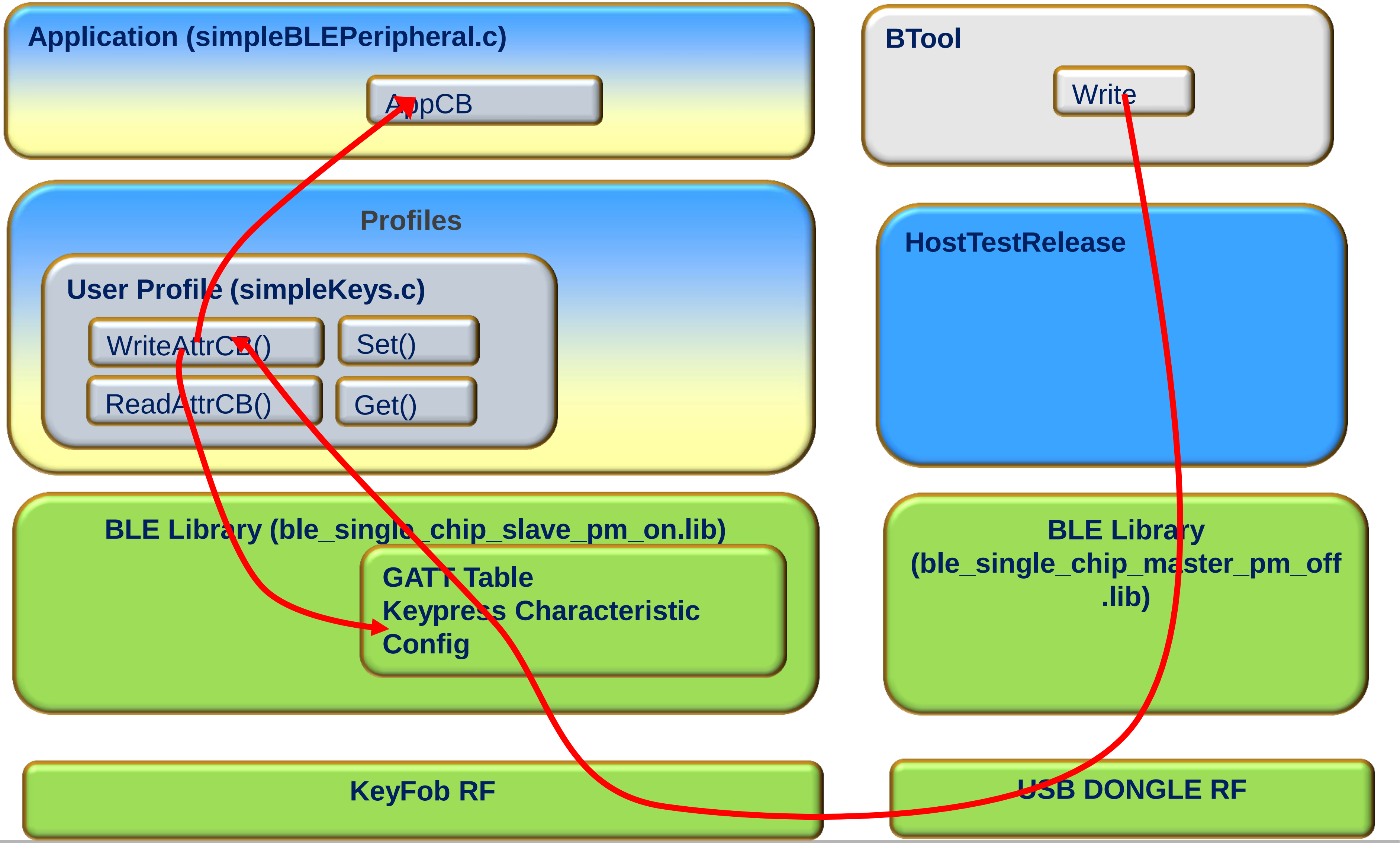
GAP Bond (gapPeripheralBondMgr.c)
- GAPBondMgr_SetParameter (passkey)
- GAPBondMgr_SetParameter (IO cap)

GAP GATT Server (gapgattserver.h)
-GGS_SetParameter(DeviceName)

User Profile (simpleGATTProfile.c)
-HBP_SetParameter (<<name>>)
-HBP_GetParameter (<<name>>)

BLE Library (ble_single_chip_slave_pm_on.lib)

Application – Turn on Notifications



Application (simpleBLEPeripheral.c)

AppCB

BTool

Write

Profiles

User Profile (simpleKeys.c)

WriteAttrCB()

Set()

ReadAttrCB()

Get()

HostTestRelease

BLE Library (ble_single_chip_slave_pm_on.lib)

GATT Table
Keypress Characteristic
Config

BLE Library
(ble_single_chip_master_pm_off
.lib)

KeyFob RF

USB DONGLE RF