What’s New in Core Bluetooth

Session 712

Craig Dooley, Bluetooth Engineer
Duy Phan, Bluetooth Engineer
Introduction
Enhanced reliability
Platform support
L2CAP channels
Best practices
Getting the most out of Core Bluetooth
Introduction
Thank you!
Built-in Profiles

Apple Notification Center Service
Apple Media Service
MIDI over Bluetooth Low Energy
iBeacon
Current Time Service
HID Over GATT
Centrals and Peripherals
Centrals and Peripherals
Centrals and Peripherals
Centrals and Peripherals
GATT Database

- CBServices
- CBCharacteristics

- CBServices
- CBCharacteristics
GATT Database
GATT Database

CBService
CBCharacteristic
CBCharacteristic
CBCharacteristic

CBService
CBCharacteristic
CBCharacteristic
CBCharacteristic
Reading Characteristics as a Central

Services can be read from a connected Central

• Retrieve by identifier
• Retrieve connected devices

```swift
open class CBCentralManager : CBManager {
    open func retrievePeripherals(withIdentifiers identifiers: [UUID]) -> [CBPeripheral]
    open func retrieveConnectedPeripherals(withServices serviceUUIDs: [CBUUID]) -> [CBPeripheral]
}
```
Enhanced Reliability
Summary
Backgrounded Apps

iOS Apps can continue using Core Bluetooth in the background

- Audio, AirPlay, and Picture in Picture
- Location updates
- Newsstand downloads
- External accessory communication
- Uses Bluetooth LE accessories
- Acts as a Bluetooth LE accessory
- Background fetch
- Remote notifications

Steps: Add the Required Background Modes key to your info.plist file
Backgrounded Apps

iOS Apps can continue using Core Bluetooth in the background
CBCentralManager restoration

Central operations can continue when your app is not running

• Scan for new devices with services
• Connect to an already known device
public let CBCentralManagerOptionRestoreIdentifierKey: String

/**< * @seealso CBCentralManagerRestoredStatePeripheralsKey; * @seealso CBCentralManagerRestoredStateScanServicesKey; * @seealso CBCentralManagerRestoredStateScanOptionsKey; */

optional public func centralManager(_ central: CBCentralManager, willRestoreState dict: [String: Any])
Peripheral operations can continue when your app is not running

- Publish local services
- Advertise service UUID
public let CBPeripheralManagerOptionRestoreIdentifierKey: String

/*
 * @seealso            CBPeripheralManagerRestoredStateServicesKey;
 * @seealso            CBPeripheralManagerRestoredStateAdvertisementDataKey;
 */

optional public func peripheralManager(_ peripheral: CBPeripheralManager,
                                       willRestoreState dict: [String: Any])
State Preservation and Restoration
State Preservation and Restoration

Works across device reboot or Bluetooth system events
State Preservation and Restoration

Works across device reboot or Bluetooth system events

• Try to ask for as few system resources as possible
State Preservation and Restoration

Works across device reboot or Bluetooth system events

• Try to ask for as few system resources as possible
• Background activities will be stopped if
State Preservation and Restoration

Works across device reboot or Bluetooth system events

• Try to ask for as few system resources as possible
• Background activities will be stopped if
  - User force quits the app
State Preservation and Restoration

Works across device reboot or Bluetooth system events

• Try to ask for as few system resources as possible
• Background activities will be stopped if
  - User force quits the app
  - User disables Bluetooth
Write Without Response would be dropped due to memory pressure

New property will tell your app if more data can be sent

```swift
open class CBPeripheral: CBPeer {
    open var canSendWriteWithoutResponse: Bool { get }
}

public protocol CBPeripheralDelegate: NSObjectProtocol {
    optional public func peripheralIsReady(toSendWriteWithoutResponse peripheral: CBPeripheral)
}
```
Platform Support
macOS 10.7

iOS 5

tvOS 9
iOS + macOS

Foreground and background apps
iOS + macOS

Foreground and background apps
Central and Peripheral
15 ms minimum connection interval
iOS + macOS

Foreground and background apps
Central and Peripheral
15 ms minimum connection interval
State Preservation and Restoration on iOS
Foreground app only
tvOS

Foreground app only

Central role only
tvOS

Foreground app only

Central role only

Limited to 2 simultaneous connections
tvOS

Foreground app only
Central role only
Limited to 2 simultaneous connections
30 ms minimum connection interval
tvOS

Foreground app only

Central role only

Limited to 2 simultaneous connections

30 ms minimum connection interval

Peripherals disconnected when app is moved to the background
watchOS
watchOS

Access dictated by system runtime policies
watchOS

Access dictated by system runtime policies
Central role only
watchOS

Access dictated by system runtime policies

Central role only

Limited to 2 simultaneous connections
watchOS

Access dictated by system runtime policies

Central role only

Limited to 2 simultaneous connections

30 ms minimum connection interval
Access dictated by system runtime policies

Central role only

Limited to 2 simultaneous connections

30 ms minimum connection interval

Peripherals disconnected when app is suspended
watchOS

Access dictated by system runtime policies

Central role only

Limited to 2 simultaneous connections

30 ms minimum connection interval

Peripherals disconnected when app is suspended

Supported on Apple Watch Series 2
L2CAP Channels
L2CAP Connection Oriented Channels

Bluetooth SIG Protocol underlying all communication

Logical Link Control and Adaptation Protocol

Stream between two devices

Introduced for LE in Bluetooth Core Spec 4.1
L2CAP Channels

CBCentralManager

CBService
CBCharacteristic
CBCharacteristic
CBCharacteristic
L2CAP Channels

[Diagram showing two mobile devices with 'CBL2CAPChannel', 'CBCentralManager', 'CBService', 'CBCharacteristic' elements]

The diagram illustrates the interaction between L2CAP Channels, CBCentralManager, CBService, and CBCharacteristic.
Open an L2CAP Channel on an existing CBPeripheral connection

```swift
open class CBPeripheral: CBPeer {
    open func openL2CAPChannel(_ PSM: CBL2CAPPSM)
}

public protocol CBPeripheralDelegate: NSObjectProtocol {
    optional public func peripheral(_ peripheral: CBPeripheral,
           didOpen channel: CBL2CAPChannel?, error: Error?)
}
```
SIG Specified PSM for standardized profiles

Locally assigned PSM for dynamic services

```swift
/*!  @const CBUUIDL2CAppSMCharacteristicString
 *  @discussion The PSM (a little endian uint16_t) of an L2CAP Channel associated with the
 *  GATT service containing this characteristic. Servers can publish this characteristic with
 *  the UUID ABDD3056-28FA-441D-A470-55A75A52553A
 */

public let CBUUIDL2CAppSMCharacteristicString: String
```
Peripheral Side L2CAP

Listen for incoming L2CAP Channels

```swift
open class CBPeripheralManager : CBManager {
    open func publishL2CAPChannel(withEncryption encryptionRequired: Bool)
    open func unpublishL2CAPChannel(_ PSM: CBL2CAPPSM)
}

public protocol CBPeripheralManagerDelegate : NSObjectProtocol {
    optional public func peripheralManager(_ peripheral: CBPeripheralManager,
                                            didPublishL2CAPChannel PSM: CBL2CAPPSM,
                                            error: Error?)
}
```
Opening an L2CAP Channel

 CBCentralManager

 MyService
 MyData
Opening an L2CAP Channel

peripheral.publishL2CAPChannel(withEncryption: true)
Opening an L2CAP Channel

```swift
optional public func peripheralManager(_ peripheral: CBPeripheralManager,
didPublishL2CAPChannel PSM: CBL2CAPPSM, error: Error?)
```
Opening an L2CAP Channel

```swift
optional public func peripheralManager(_ peripheral: CBPeripheralManager,
    didPublishL2CAPChannel PSM: CBL2CAPPSM, error: Error?)
```
Opening an L2CAP Channel

CBCentralManager

MyService

MyData

L2CAP PSM
Opening an L2CAP Channel

CBCentralManager

MyService

MyData

L2CAP PSM
Opening an L2CAP Channel

peripheral.openL2CAPChannel(PSM)
Opening an L2CAP Channel

```swift
optional public func peripheralManager(_ peripheral: CBPeripheralManager, didOpen channel: CBL2CAPChannel?, error: Error?)
```
@available(macOS 10.13, iOS 11.0, *)
open class CBL2CAPChannel: NSObject {

    open var peer: CBPeer! { get }

    open var inputStream: InputStream! { get }

    open var outputStream: OutputStream! { get }

    open var psm: CBL2CAPPSM { get }

}
Stream Events

Stream events are delivered through NSSStream

```swift
public protocol StreamDelegate: NSObjectProtocol {
    optional public func stream(_ aStream: Stream, handle eventCode: Stream.Event)
}

public struct Stream.Event: OptionSet {
    public static var openCompleted: Stream.Event { get }
    public static var hasBytesAvailable: Stream.Event { get }
    public static var hasSpaceAvailable: Stream.Event { get }
    public static var errorOccurred: Stream.Event { get }
    public static var endEncountered: Stream.Event { get }
}
```
Closing Channels

Channels may be closed due to

• Link loss
• Central close
• Peripheral unpublished
• Peripheral object is released
When Should L2CAP Be Used?

Use GATT where it makes sense

Lowest overhead

Best performance

Best for large data transfers

Great for stream protocols
Best Practices
Follow the Bluetooth Accessory Design Guidelines for Apple Products
Use Existing Profiles and Services
Why does it take so long to connect?
Time to Discover

Peripheral

Advertisement

Central

Advertising

Scanning
Time to Discover

Peripheral

Advertisement

Central

Scan

Advertising

Scanning
Connection Speed

Use the shortest advertising interval possible

Optimize for when users are trying to use your accessory

See the Bluetooth Accessory Design Guidelines for power-efficient advertising intervals
No need to scan for a peripheral for reconnect

Retrieve the peripheral and directly connect

```swift
let identifier = UUID()

let peripherals = central.retrievePeripherals(withIdentifiers: [ identifier ])

central.connect(peripherals[0])
```
## Service Discovery Speed

<table>
<thead>
<tr>
<th>Battery (16 bit)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Battery Level</td>
<td></td>
</tr>
<tr>
<td>MyService (128 bit)</td>
<td></td>
</tr>
<tr>
<td>MyData</td>
<td></td>
</tr>
<tr>
<td>OtherData</td>
<td></td>
</tr>
<tr>
<td>MoreData</td>
<td></td>
</tr>
<tr>
<td>Device Information (16 bit)</td>
<td></td>
</tr>
<tr>
<td>Serial Number</td>
<td></td>
</tr>
<tr>
<td>Software Version</td>
<td></td>
</tr>
<tr>
<td>PnP ID</td>
<td></td>
</tr>
<tr>
<td>CBService</td>
<td></td>
</tr>
<tr>
<td>CBCharacteristic</td>
<td></td>
</tr>
<tr>
<td>CBCharacteristic</td>
<td></td>
</tr>
<tr>
<td>CBCharacteristic</td>
<td></td>
</tr>
</tbody>
</table>
Service Discovery Speed

Use as few services/characteristics as possible

<table>
<thead>
<tr>
<th>Battery (16 bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Battery Level</td>
</tr>
<tr>
<td>MyService (128 bit)</td>
</tr>
<tr>
<td>MyData</td>
</tr>
<tr>
<td>OtherData</td>
</tr>
<tr>
<td>MoreData</td>
</tr>
<tr>
<td>Device Information (16 bit)</td>
</tr>
<tr>
<td>Serial Number</td>
</tr>
<tr>
<td>Software Version</td>
</tr>
<tr>
<td>PnP ID</td>
</tr>
<tr>
<td>CBService</td>
</tr>
<tr>
<td>CBCharacteristic</td>
</tr>
<tr>
<td>CBCharacteristic</td>
</tr>
<tr>
<td>CBCharacteristic</td>
</tr>
</tbody>
</table>
Service Discovery Speed

Use as few services/characteristics as possible

- Battery (16 bit)
- Battery Level
- MyService (128 bit)
- MyData
- OtherData
- MoreData
- Device Information (16 bit)
- Serial Number
- Software Version
- PnP ID
Service Discovery Speed

Use as few services/characteristics as possible

Group services by UUID size
Service Discovery Speed

Use as few services/characteristics as possible

Group services by UUID size

Support GATT Caching
Service Discovery Speed

Use as few services/characteristics as possible

Group services by UUID size

Support GATT Caching

Use “Service Changed”
New Accessory Recommendations

Use the newest chipset / Bluetooth standard available

4.2 and 5.0 are backward compatible

Follow these best practices
Getting the Most out of Core Bluetooth

Duy Phan, Bluetooth Engineer
1MB = 3,240 seconds
2.5 kbps
Protocol Overhead

Application

GATT

ATT

L2CAP

Controller

Application

GATT

ATT

L2CAP

Controller

+-------------------+-------------------+
| L2CAP             | Attribute Data    |
| 4 Bytes           | 20 Bytes          |
+-------------------+-------------------+

Packet
Write With Response

Central

Write

Peripheral

Response

Interval
Write With Response

Central

Peripheral

Interval

Interval

Interval
Write Without Response

Reliable with Core Bluetooth flow control

Use all available connection events to transmit

Takes advantage of larger Connection Event Length
Write Without Response
Write Without Response

Default MTU

37 kbps
Write Without Response

Default MTU

37 kbps
Write Without Response

Default MTU

Interval

- 37 kbps

Larger MTU

Interval

- 48 kbps
Fitting your data

Apple devices determine the optimal MTU

Accessories should support a large MTU

Use large attributes aligned to MTU

```swift
open class CBPeripheral: CBPeer {
    open func maximumWriteValueLength(for type: CBCharacteristicWriteType) -> Int
}

open class CBCentral: CBPeer {
    open var maximumUpdateValueLength: Int { get }
}
```
Write Without Response

- Default MTU: 37 kbps
- Larger MTU: 48 kbps

Interval
Write Without Response

Default MTU

Larger MTU

EDL

Interval

Interval

Interval

37 kbps

48 kbps
Extended Data Length

New Feature in Bluetooth 4.2

Much larger packets (251 vs 27 bytes)

Transparent to the application

4x throughput with the same radio time

Available on iPhone 7 and Apple Watch Series 2
L2CAP Connection Oriented Channels

Application
  GATT
  ATT
  L2CAP
  Controller

Application
  GATT
  ATT
  L2CAP
  Controller

L2CAP 4 Bytes
Attribute Data 247 Bytes
Packet
L2CAP Connection Oriented Channels

197 kbps

Packet

L2CAP
4 Bytes

Attribute Data
247 Bytes
Faster Connection Interval

L2CAP + EDL
Faster Connection Interval

L2CAP + EDL

Interval

Interval

Interval

Interval

Interval

394 kbps
Throughput (kbps)

- Write With Response: 2.5
- Write Without Response: 5.2
- Packed CE Length: 37
- Larger MTU: 48
- EDL: 135
- L2CAP + EDL: 197
- L2CAP + EDL + 15ms Int: 394
Summary

Request a shorter Connection Interval

Take advantage of GATT optimizations

Use L2CAP Channel for large transfers and stream protocols

Update your hardware (4.2 EDL, 5.0) for best performance and battery life
Wrap Up
Key Takeaways

Check out State Restoration

Expand your app to tvOS and watchOS

Use L2CAP for stream protocols or large data transfers

Use the newest Bluetooth chipset available

Follow the Bluetooth Accessory Design Guidelines
More Information

## Related Sessions

<table>
<thead>
<tr>
<th>Session</th>
<th>Conference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core Bluetooth 101</td>
<td>WWDC 2012</td>
</tr>
<tr>
<td>Core Bluetooth</td>
<td>WWDC 2013</td>
</tr>
<tr>
<td>Labs</td>
<td>Location</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Bluetooth Lab</td>
<td>Technology Lab J</td>
</tr>
<tr>
<td>Bluetooth Lab</td>
<td>Technology Lab J</td>
</tr>
</tbody>
</table>