CommLib Migration Guide

This guide is intended for app developers that need to update their app from CommLib versions up to 7.x.x to 8.0.0 and up.

Introduction

CommLib 8.0.0 marks a big API change for LAN, where the `DiscoveryManager` is deprecated in favour of the new CommLib API for discovery, appliance management and connection handling that was already available for BLE.

API changes

One API for all transports

CommLib 7.x.x provides an API to communicate with BLE devices. Since the release of CommLib 8.0.0, this API is also available for communicating with LAN and cloud devices, meaning all communication is done using a single API. This API effectively hides whatever communication technology is used to talk to the physical peripheral, be it BLE, Wifi, cellular and whatever transport is added in the future (Ethernet, NFC, Zigbee, etc.) but allows the app to 'talk' to the `Appliance` and making sure that it 'just works'.

Module name changes

The Gradle module names within the CommLib project have been updated to better reflect their responsibilities. Some of the modules have been extracted from the original CommLib module. The name changes are performed as follows:

Old name	Old purpose	New name	New purpose
`dicommClientLib`	Complete CommLib library	`commlib`	Umbrella module, providing BLE, LAN and Cloud implementations of CommLib API
`<API packages in dicommClientLib>`	Public API classes	`commlib-api`	Public API classes
`commlib-all`	Glue module between CommLib API and BlueLib	`commlib-ble`	BLE implementation of CommLib API
`<cloud packages in dicommClientLib>`	Cloud classes	`commlib-cloud`	Cloud implementation of CommLib API
`<LAN packages in dicommClientLib>`	LAN classes	`commlib-lan`	LAN implementation of CommLib API
`commlib-bdd`	submodule in commlib-all	`commlib-integration-tests`	Module providing on-device test app and BDD tests

API classes

DICommClientWrapper (deprecated in 8.0.0)

This type was used to properly initialise a singleton DiscoveryManager with its dependencies as provided by the app, such as an `ApplianceFactory` or `ApplianceDatabase` instance. This is currently done during initialisation of the `CommCentral` and the `*TransportContext` classes.

DiscoveryManager (deprecated in 8.0.0)

In CommLib versions up to 7.x.x most LAN functionality is contained in the `DiscoveryManager` class, ie. discovery, `Appliance` creation, persistent storage handling and connection handling. These tasks have now been split up to `CommCentral` (wiring of `DiscoveryStrategies`, `ApplianceFactory` and `TransportContexts`), the different `DiscoveryStrategy` implementations (discovery of `NetworkNodes`, connection handling) and `ApplianceManager` (persistent storage, appliance availability handling).

CommCentral

This is the main entry point of the CommLib library. The app uses this class to perform discovery for appliances in a transport-agnostic fashion. During construction, the app provides an `ApplianceFactory` implementation and one or more `TransportContext` implementations. The `CommCentral` instance is then used to start and stop discovery, and to obtain a reference to the `ApplianceManager`.

Only one `CommCentral` instance may be created during an app execution lifecycle. This is not enforced on a code level however, but since it is possible that the provided `TransportContexts` may use a protocol or code that doesn't allow multiple concurrent clients (which is the case of BlueLib, which is used for BLE connectivity), correct functioning of CommLib cannot be guaranteed when multiple instances of `CommCentral` are created.

This class now effectively deprecates `DICommClientWrapper`.

ApplianceManager (new in 8.0.0)

The `ApplianceManager` is used by the app to query for available appliances, store and read them from persistent storage and subscribe for changes on appliances, which can be lost, found and updated.

LanTransportContext

This type already existed in CommLib versions before 8.0.0; it provides ways to handle SSL certificate errors that can occur by accepting/rejecting certificate pins for an `Appliance`. Also, this is the location where an app can query the `Appliances` that have a pin mismatch.

LanCommunicationStrategy

This type already existed in CommLib versions before 8.0.0 and was used internally by `DiscoveryManager`, currently it's provided via the `LanTransportContext`. This is the type that implements the DiComm specific methods for PUT/GET/etc. using HTTP(S) requests; subscriptions are done using UDP broadcasts.

LanDiscoveryStrategy (new in 8.0.0)

This class uses SSDP to perform LAN discovery and is provided via the `LanTransportContext`.

Step-by-step guide

CommLib initialization

1. Setup all `TransportContext` instances
2. Setup a custom `ApplianceFactory` that can produce Appliance instances that are supported by the app and the hardware
3. Create an instance of `CommCentral` that accepts the `ApplianceFactory` and all `*TransportContext` instances the app can work with

Persistent storage

1. To store an `Appliance`, call `ApplianceManager#storeAppliance(Appliance)` with the `Appliance` instance that should be stored. Note that only the underlying `NetworkNode` is stored; to store an `Appliance` as-is, the app needs to provide an `ApplianceDatabase` to `CommCentral` during construction.