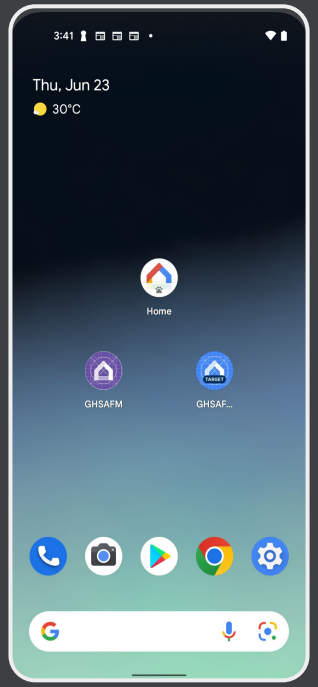
Nest

BUILDING SMART HOME APPS WITH GOOGLE HOME MOBILE SDK

Google Home Sample App for Matter (**GHSAFM**)

Demo Components

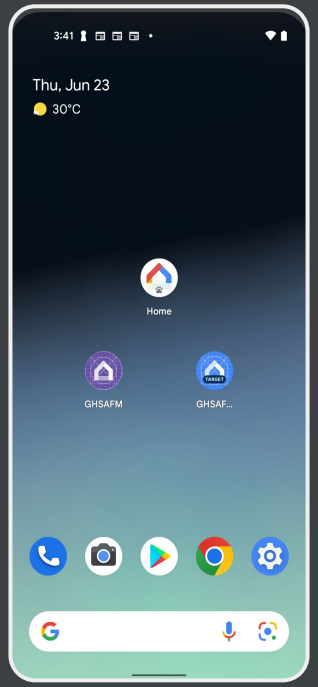
Demo Components



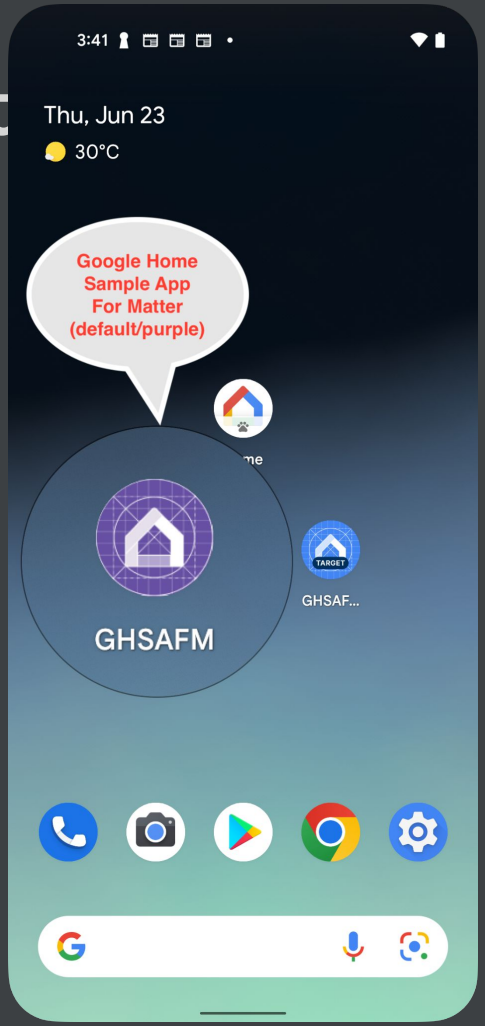
Pixel Phone

Nest

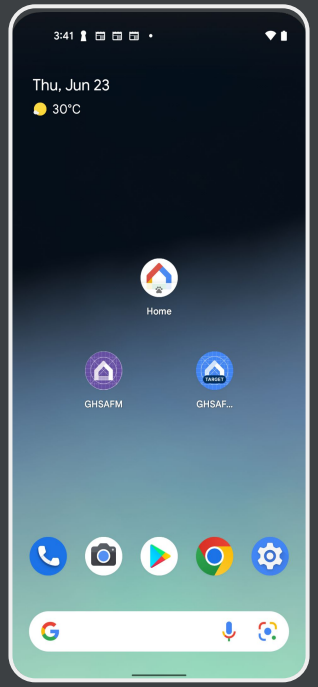
Demo Component



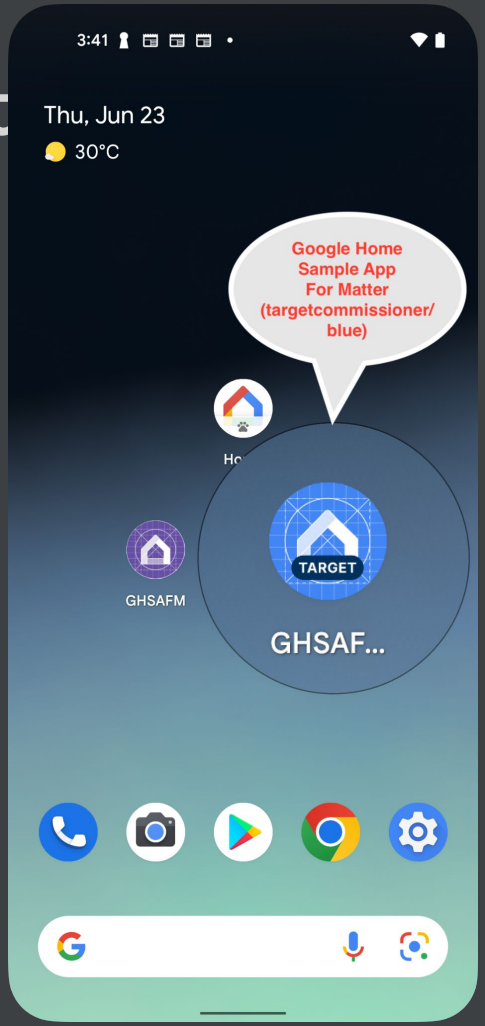
Pixel Phone



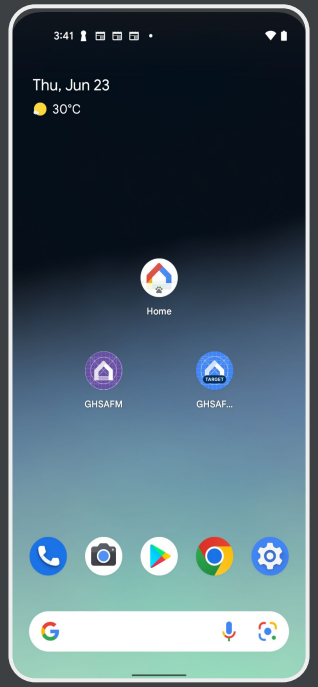
Demo Component



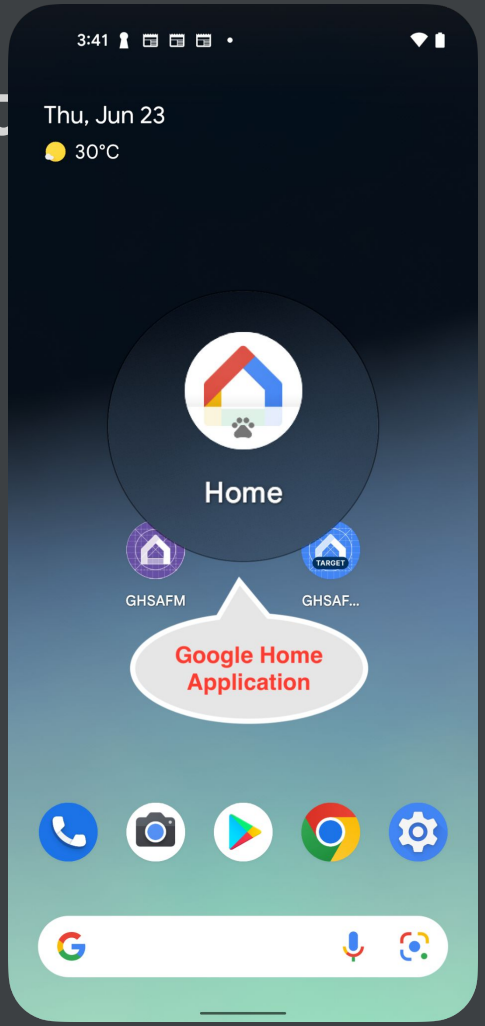
Pixel Phone



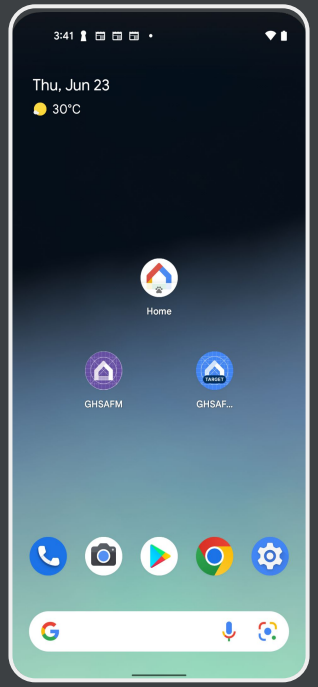
Demo Component



Pixel Phone



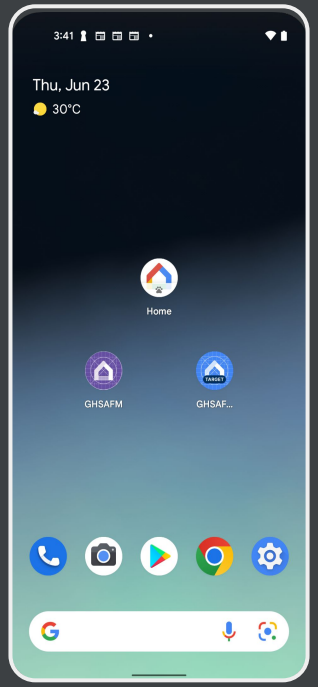
Demo Components



Pixel Phone

Nest

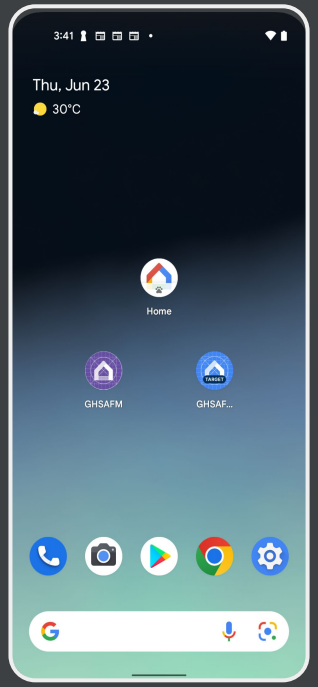
Demo Components



Pixel Phone

M5Stack
ESP-32 based IoT device

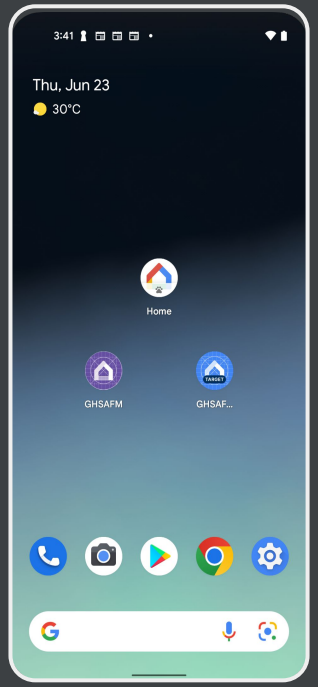
Demo Components



Pixel Phone

M5Stack
ESP-32 based IoT dev device

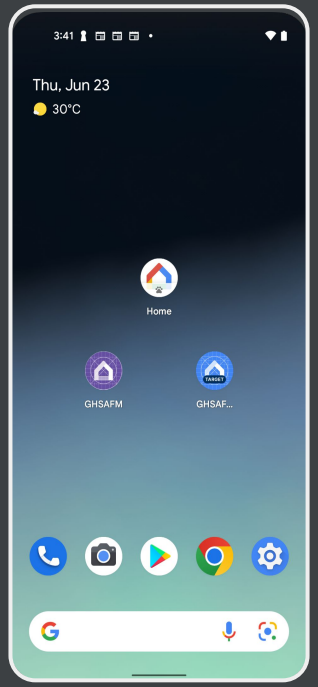
Demo Components



Pixel Phone

M5Stack
ESP-32 based IoT dev device

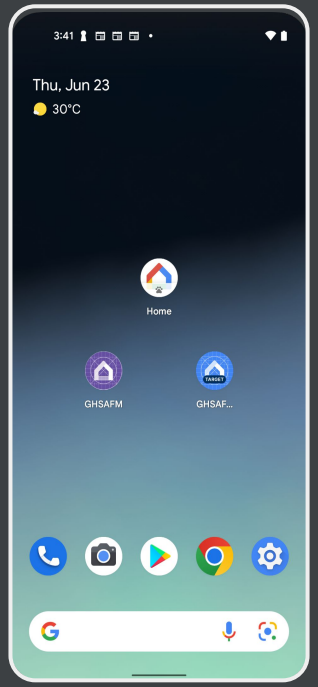
Demo Components



Pixel Phone

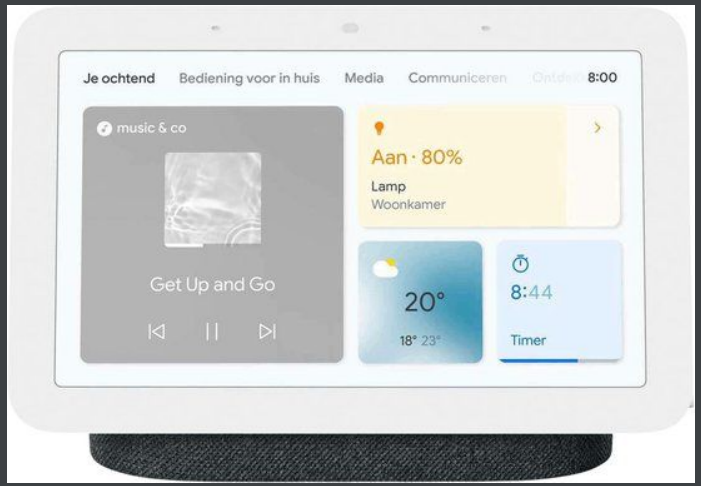
M5Stack
ESP-32 based IoT dev device

Demo Components



Pixel Phone

M5Stack
ESP-32 based IoT dev device



Google Nest Hub

Nest

Demo

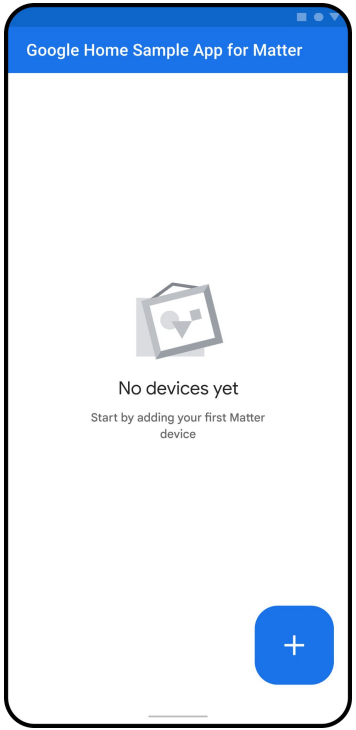
Google Play Services **Matter APIs**

Matter

```
public final class Matter {
   ...
}
```

Adding a device

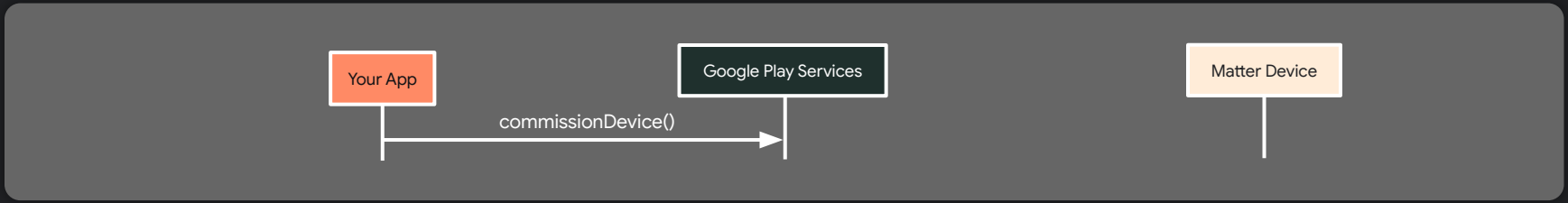
Commissioning



Matter

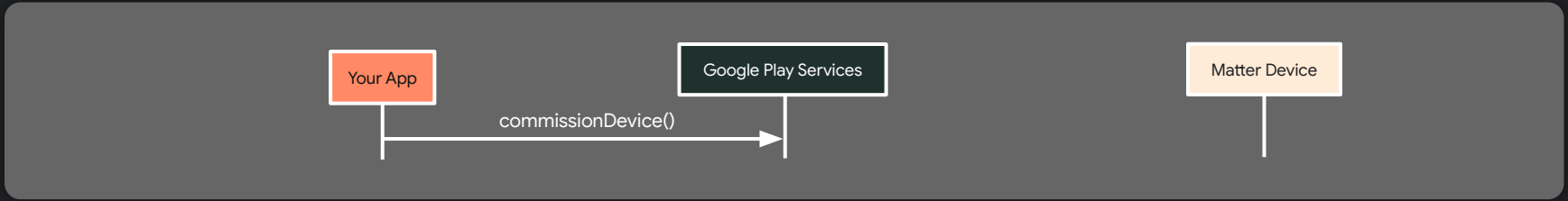
```
public final class Matter {
  public static CommissioningClient getCommissioningClient(Context
context);
  public static CommissioningClient getCommissioningClient(Activity
activity);
}
```

CommissioningClient.commissionDevice()




```
interface CommissioningClient {
  /**
   * Commission a new Matter device.
   */
  Task<IntentSender> commissionDevice(CommissioningRequest request);
}
```

CommissioningClient.commissionDevice()




```
interface CommissioningClient {
  /**
   * Commission a new Matter device.
   */
  Task<IntentSender> commissionDevice(CommissioningRequest request);
}
```

Task

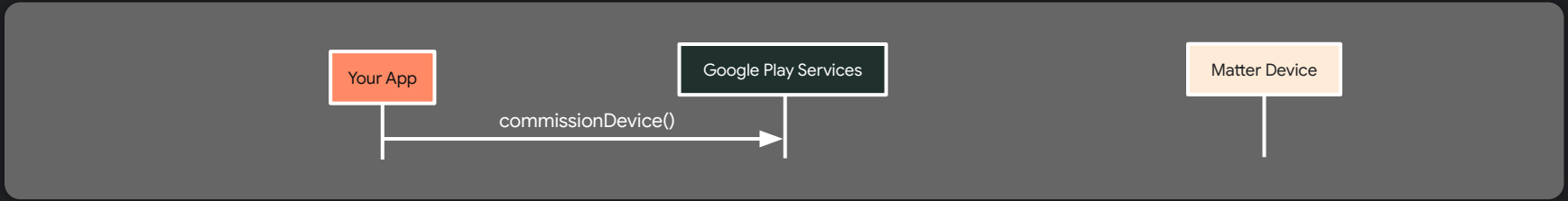
public abstract class Task
extends Object

java.lang.Object
↳ com.google.android.play.core.tasks.Task<ResultT>

Represents an asynchronous operation.

```
Matter.getCommissioningClient(activity)
    .commissionDevice(commissionDeviceRequest)
    .addOnSuccessListener { result ->
      // Communication with fragment is via livedata
      _commissionDeviceStatus.postValue(TaskStatus.Completed("Received IntentSender."))
      _commissionDeviceIntentSender.postValue(result)
    }
    .addOnFailureListener { error ->
      _commissionDeviceStatus.postValue(TaskStatus.Failed(error))
      Timber.e(error)
    }
```

CommissioningClient.commissionDevice()




```
interface CommissioningClient {
  /**
   * Commission a new Matter device.
   */
  Task<IntentSender> commissionDevice(CommissioningRequest request);
}
```

Would be great if I could simply do this...

```
val request = CommissioningRequest.builder().set(...).build()
val result = Matter.commissionDevice(request)
```

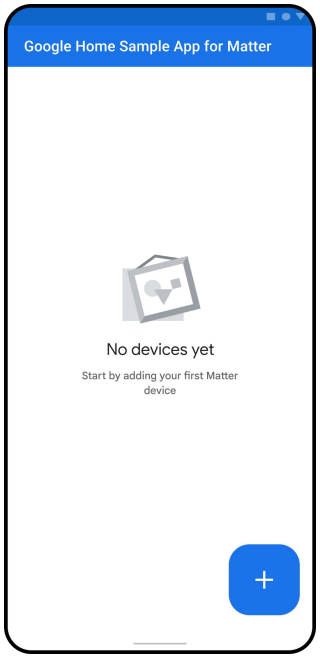
Would be great if I could simply do this...

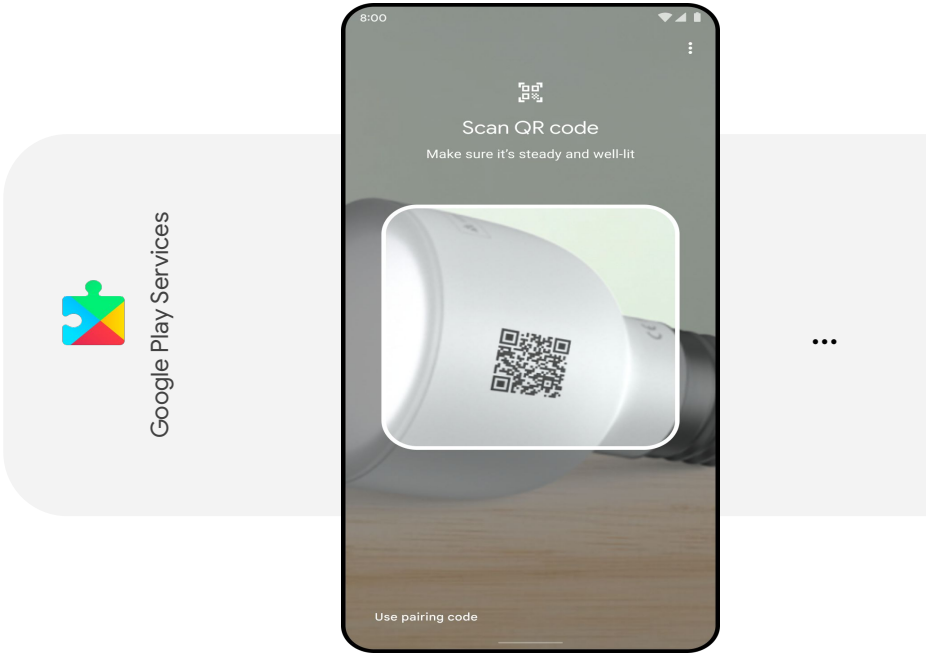
```
val request = CommissioningRequest.builder().set(...).build()
val result = Matter.commissionDevice(request)
```

Unfortunately, this won't cut it
because we have to deal with an IntentSender

Why an IntentSender?

Need to go from my app's UI → to the Google Play Services UI for Commissioning

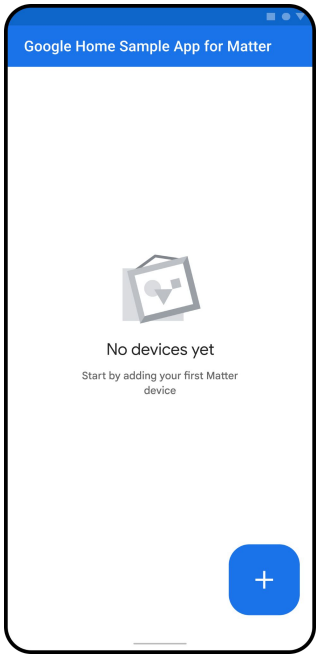




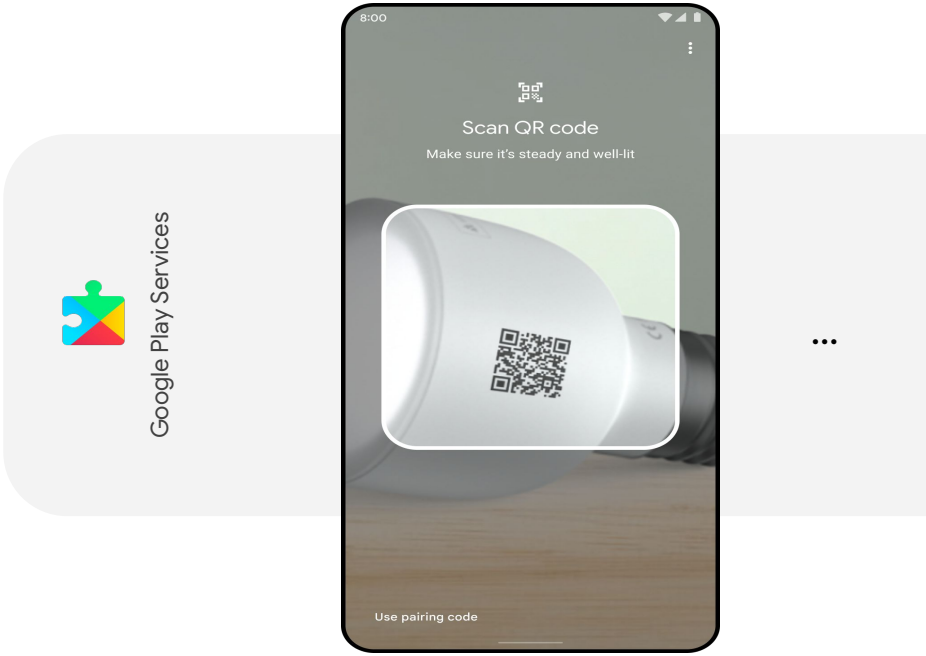
Result ← and return back a result

Why an IntentSender?

Need to go from my app's UI → to the Google Play Services UI for Commissioning



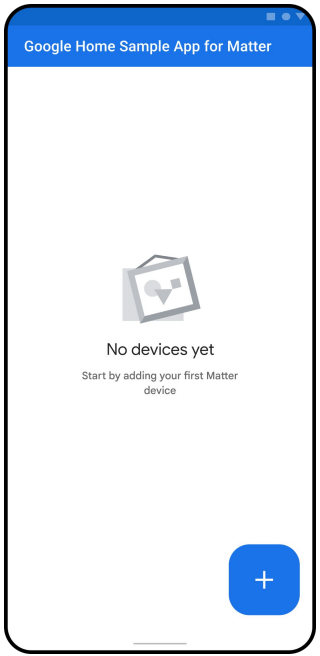
When starting an activity for a result, it is possible that the process and activity will be destroyed due to low memory.



Result ← and return back a result

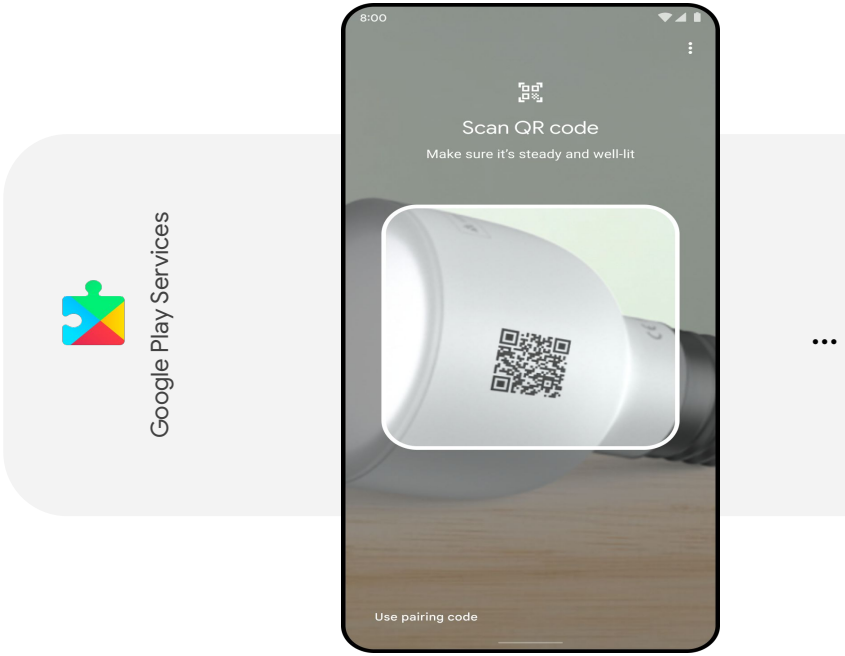
Android Activity Result API

Need to go from my app's UI → to the Google Play Services UI for Commissioning



Decouples the result callback from the place where the external activity is launched.

As the result callback needs to be available when the activity is recreated, the callback must be unconditionally registered every time the activity is created



Result ← and return back a result

Five steps to mastering ActivityResult API

...

Five steps to mastering ActivityResult API

1 Register for ActivityResult

HomeFragment.kt

```
private lateinit var commissionDeviceLauncher: ActivityResultLauncher<IntentSenderRequest>
...
override fun onCreate(savedInstanceState: Bundle?) {
  ...
  commissionDeviceLauncher =
      registerForActivityResult(ActivityResultContracts.StartIntentSenderForResult()) { result ->
        // The Commission Device activity in GPS has completed.
        val resultCode = result.resultCode
        if (resultCode == Activity.RESULT_OK) {
          viewModel.commissionDeviceSucceeded(
              result, getString(R.string.commission_device_status_success))
        } else {
          viewModel.commissionDeviceFailed(getString(R.string.status_failed_with, resultCode))
        }
      }
```

Five steps to mastering ActivityResult API

2 Trigger the “commission device” flow — part 1

HomeFragment.kt

```
private fun setupAddDeviceButton() {
  // Add device button click listener. This triggers the commissioning of a Matter device.
  binding.addDeviceButton.setOnClickListener {
      viewModel.commissionDevice(requireActivity())
  }
}
```

Five steps to mastering ActivityResult API

2 Trigger the “commission device” flow — part 2

HomeViewModel.kt

```
fun commissionDevice(activity: FragmentActivity) {
    val commissionDeviceRequest =
        CommissioningRequest.builder()
            .setCommissioningService(ComponentName(activity,AppCommissioningService::class.java))
            .build()

    Matter.getCommissioningClient(activity)
        .commissionDevice(commissionDeviceRequest)
        .addOnSuccessListener { result ->
          _commissionDeviceIntentSender.postValue(result)
        }
        .addOnFailureListener { error ->
          ...
        }
  }
```

Five steps to mastering ActivityResult API

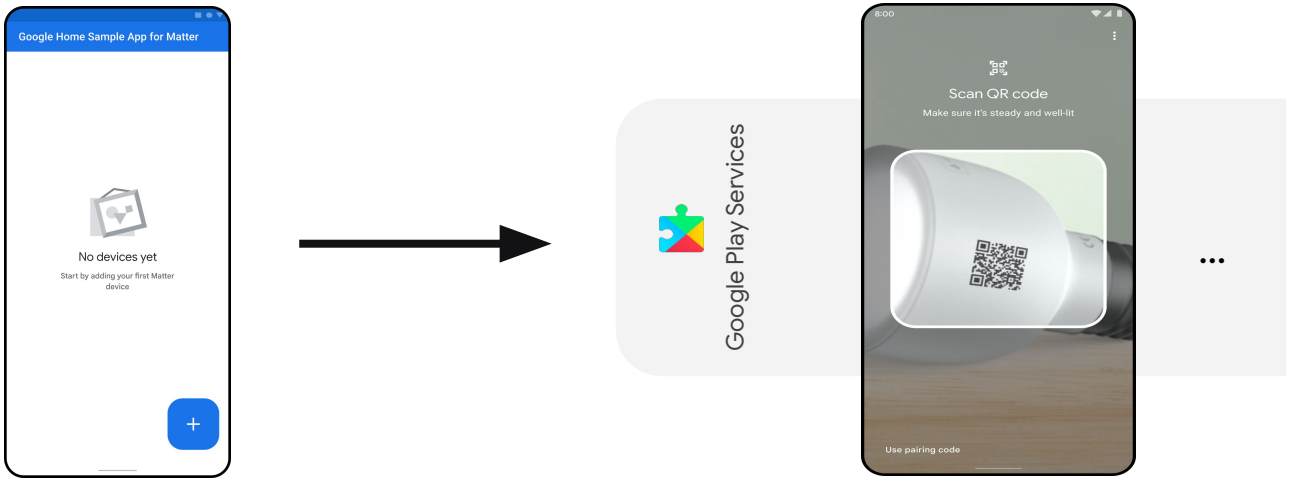
3 Launch the GPS Activity for device commissioning

HomeFragment.kt

```
private fun setupObservers() {
   ...
   viewModel.commissionDeviceIntentSender.observe(viewLifecycleOwner) { sender ->
       commissionDeviceLauncher.launch(IntentSenderRequest.Builder(sender).build())
   }
   ...
}
```

Five steps to mastering ActivityResult API

4 GPS “CommissionDevice” Activity is launched



Five steps to mastering ActivityResult API

5 Control returns to app with result

HomeFragment.kt

```
private lateinit var commissionDeviceLauncher: ActivityResultLauncher<IntentSenderRequest>
...
override fun onCreate(savedInstanceState: Bundle?) {
  ...
  commissionDeviceLauncher =
      registerForActivityResult(ActivityResultContracts.StartIntentSenderForResult()) { result ->
        // The Commission Device activity in GPS has completed.
        val resultCode = result.resultCode
        if (resultCode == Activity.RESULT_OK) {
          viewModel.commissionDeviceSucceeded(
              result, getString(R.string.commission_device_status_success))
        } else {
          viewModel.commissionDeviceFailed(getString(R.string.status_failed_with, resultCode))
        }
      }
```

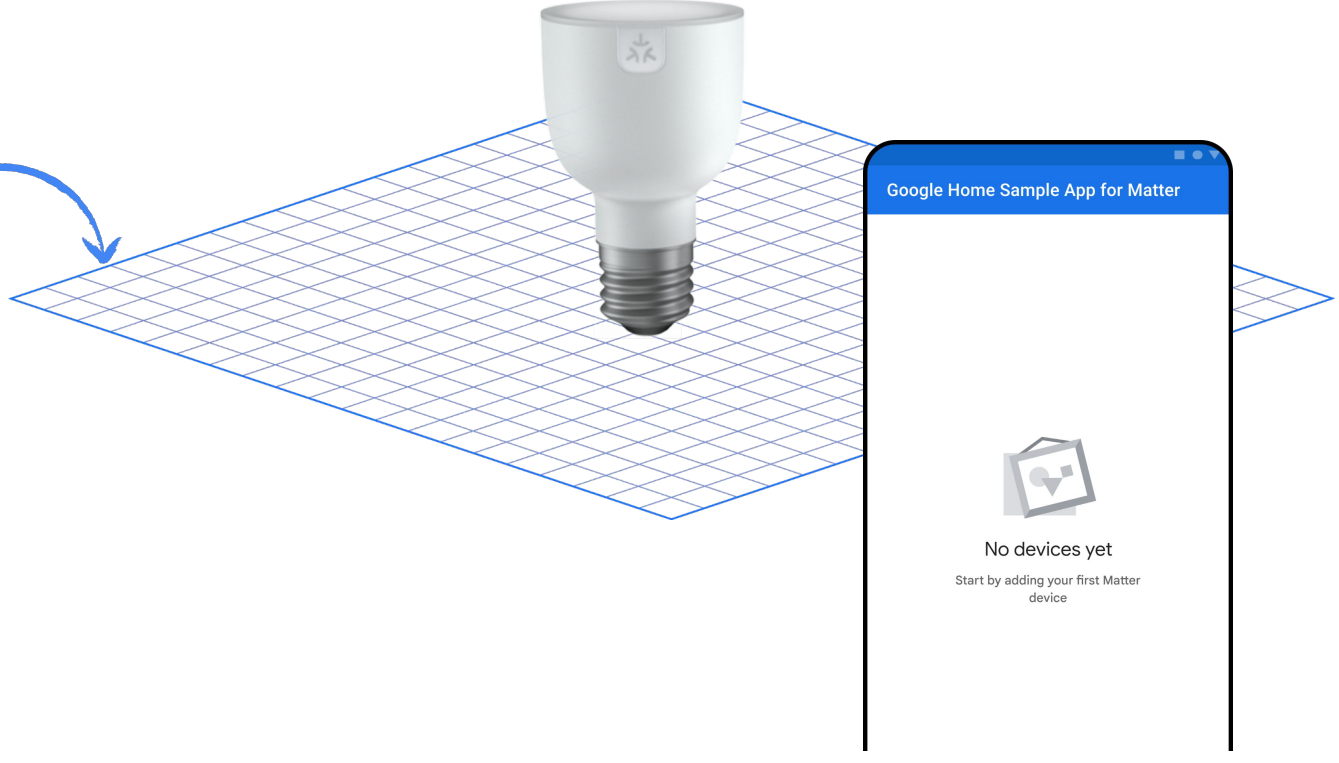
Fabric

Shared domain of trust among devices in a home network that enables them to communicate with each other.

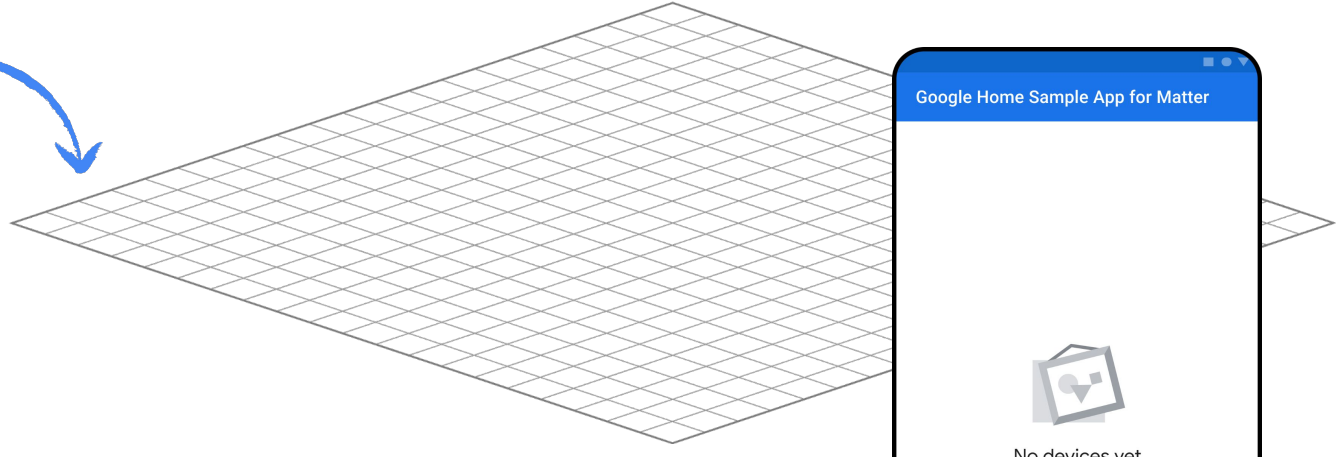
This shared domain of trust is established via a common set of cryptographic keys.

Google Fabric

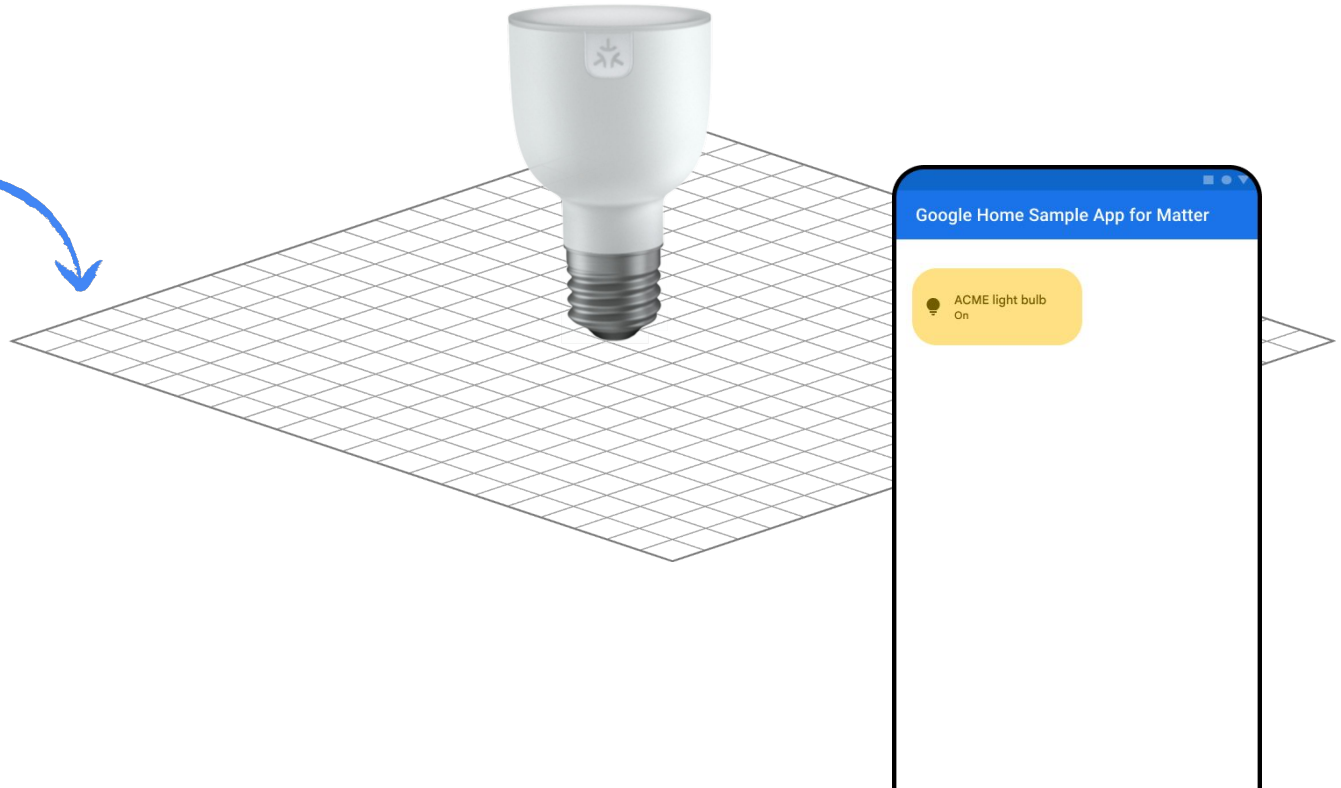
Google Fabric



Your fabric


Google Home Sample App for Matter
No devices yet
Start by adding your first Matter device

Your fabric



setCommissioningService

```
val request = CommissioningRequest.builder()
    .setCommissioningService(ComponentName(requireContext(),
                                AppCommissioningService::class.java))
    .build()
```

AppCommissioningService

```
class AppCommissioningService : Service(), CommissioningService.Callback {
  ...
  override fun onCommissioningRequested(metadata:
CommissioningRequestMetadata) {
    // Perform commissioning on the app's custom fabric.
    ...
  }
  ...
}
```

CommissioningRequestMetadata

public final class **CommissioningRequestMetadata** extends Object
implements Parcelable

Metadata providing information necessary to complete commissioning of a Matter device.

Public Method Summary

static CommissioningRequestMetadata.Builder	builder() Returns an empty **`CommissioningRequestMetadata.Builder`** instance.
boolean	equals(Object other)
DeviceDescriptor	getDeviceDescriptor() Returns information about the device to be commissioned.
NetworkLocation	getNetworkLocation() Returns the location on the current network at which the device can be reached.
long	getPasscode() Returns the code needed to establish a connection to the device to be commissioned.
int	hashCode()
void	writeToParcel(Parcel dest, int flags)

Commission to 3P fabric using Matter SDK

```
// Perform commissioning on custom fabric for the sample app.
serviceScope.launch {
  val deviceId = devicesRepository.incrementAndReturnLastDeviceId()
  chipClient.awaitEstablishPaseConnection(
      deviceId,
      metadata.networkLocation.ipAddress.hostAddress!!,
      metadata.networkLocation.port,
      metadata.passcode)
  chipClient.awaitCommissionDevice(deviceId, null)
  devicesRepository.addDevice(
      Device.newBuilder()
          .setName("REAL-$deviceId") // default name that can be overridden by user in next step
          .setDeviceId(deviceId)
          .setDateCommissioned(getTimestampForNow())
          .setVendorId(metadata.deviceDescriptor.vendorId.toString())
          .setProductId(metadata.deviceDescriptor.productId.toString())
          .setDeviceType(convertToAppDeviceType(metadata.deviceDescriptor.deviceType))
          .build())
```

Commission to 3P fabric using Matter SDK

```
// Perform commissioning on custom fabric for the sample app.
serviceScope.launch {
  val deviceId = devicesRepository.incrementAndReturnLastDeviceId()
  chipClient.awaitEstablishPaseConnection(
      deviceId,
      metadata.networkLocation.ipAddress.hostAddress!!,
      metadata.networkLocation.port,
      metadata.passcode)
  chipClient.awaitCommissionDevice(deviceId, null)
  devicesRepository.addDevice(
      Device.newBuilder()
          .setName("REAL-$deviceId") // default name that can be overridden by user in next step
          .setDeviceId(deviceId)
          .setDateCommissioned(getTimestampForNow())
          .setVendorId(metadata.deviceDescriptor.vendorId.toString())
          .setProductId(metadata.deviceDescriptor.productId.toString())
          .setDeviceType(convertToAppDeviceType(metadata.deviceDescriptor.deviceType))
          .build())
```

Commission to 3P fabric using Matter SDK

```
// Perform commissioning on custom fabric for the sample app.
serviceScope.launch {
  val deviceId = devicesRepository.incrementAndReturnLastDeviceId()
  chipClient.awaitEstablishPaseConnection(
      deviceId,
      metadata.networkLocation.ipAddress.hostAddress!!,
      metadata.networkLocation.port,
      metadata.passcode)
  chipClient.awaitCommissionDevice(deviceId, null)
  devicesRepository.addDevice(
      Device.newBuilder()
          .setName("REAL-$deviceId") // default name that can be overridden by user in next step
          .setDeviceId(deviceId)
          .setDateCommissioned(getTimestampForNow())
          .setVendorId(metadata.deviceDescriptor.vendorId.toString())
          .setProductId(metadata.deviceDescriptor.productId.toString())
          .setDeviceType(convertToAppDeviceType(metadata.deviceDescriptor.deviceType))
          .build())
```

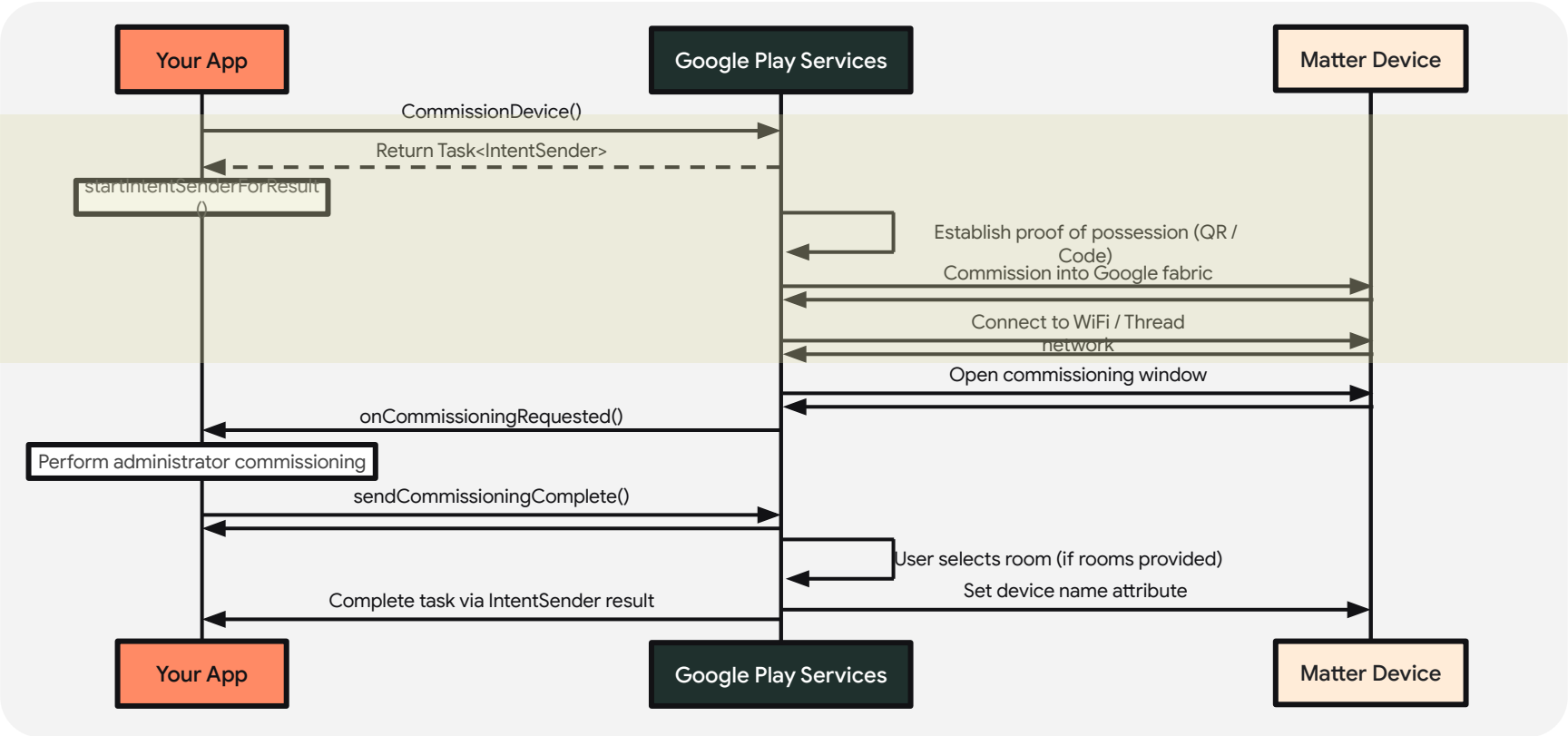
Commission to 3P fabric using Matter SDK

```
// Perform commissioning on custom fabric for the sample app.
serviceScope.launch {
  val deviceId = devicesRepository.incrementAndReturnLastDeviceId()
  chipClient.awaitEstablishPaseConnection(
      deviceId,
      metadata.networkLocation.ipAddress.hostAddress!!,
      metadata.networkLocation.port,
      metadata.passcode)
  chipClient.awaitCommissionDevice(deviceId, null)
  devicesRepository.addDevice(
      Device.newBuilder()
          .setName("REAL-$deviceId") // default name that can be overridden by user in next step
          .setDeviceId(deviceId)
          .setDateCommissioned(getTimestampForNow())
          .setVendorId(metadata.deviceDescriptor.vendorId.toString())
          .setProductId(metadata.deviceDescriptor.productId.toString())
          .setDeviceType(convertToAppDeviceType(metadata.deviceDescriptor.deviceType))
          .build())
```

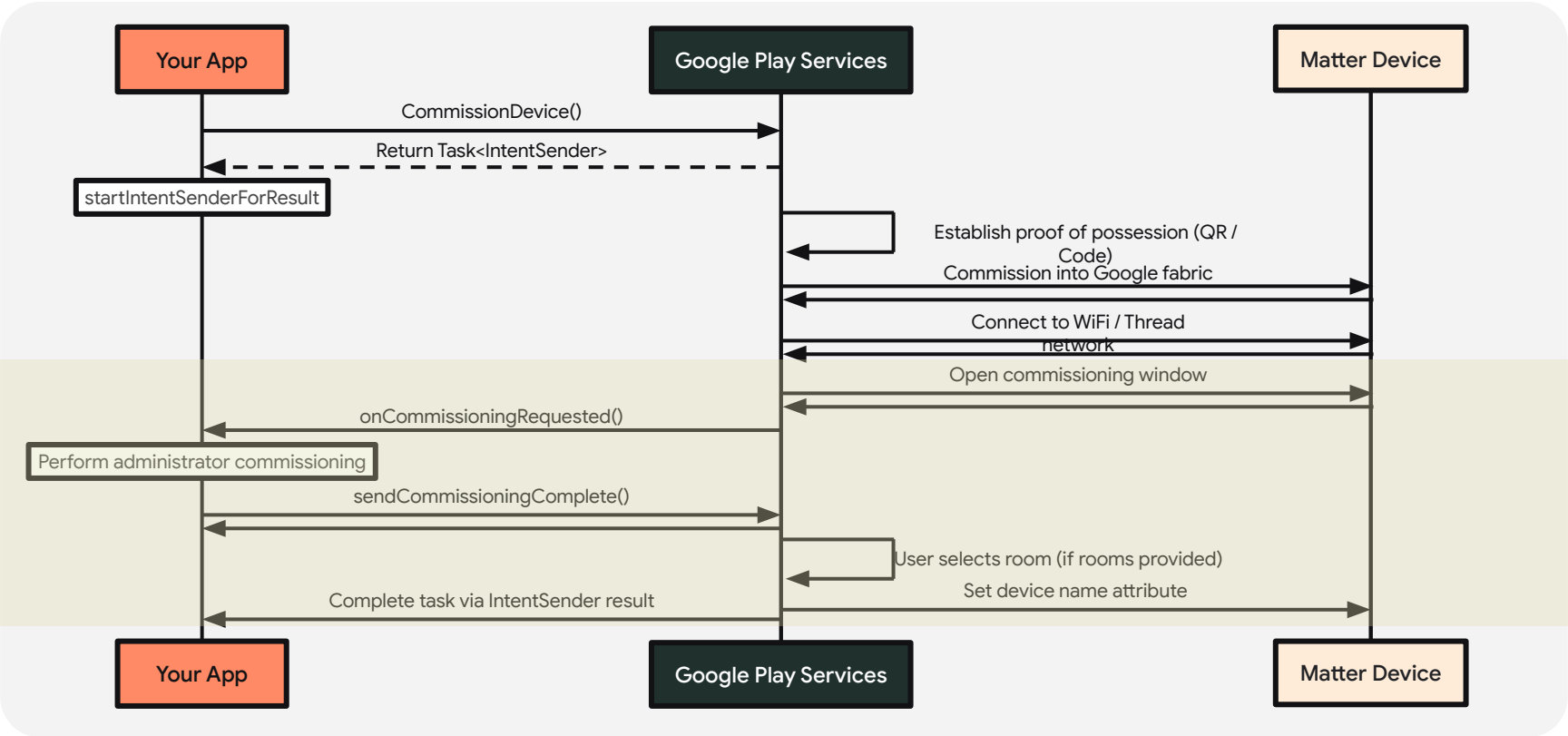
Commission to 3P fabric using Matter SDK

```
// Perform commissioning on custom fabric for the sample app.
serviceScope.launch {
  val deviceId = devicesRepository.incrementAndReturnLastDeviceId()
  chipClient.awaitEstablishPaseConnection(
      deviceId,
      metadata.networkLocation.ipAddress.hostAddress!!,
      metadata.networkLocation.port,
      metadata.passcode)
  chipClient.awaitCommissionDevice(deviceId, null)
  devicesRepository.addDevice(
      Device.newBuilder()
          .setName("REAL-$deviceId") // default name that can be overridden by user in next step
          .setDeviceId(deviceId)
          .setDateCommissioned(getTimestampForNow())
          .setVendorId(metadata.deviceDescriptor.vendorId.toString())
          .setProductId(metadata.deviceDescriptor.productId.toString())
          .setDeviceType(convertToAppDeviceType(metadata.deviceDescriptor.deviceType))
          .build())
```

Commission Flow

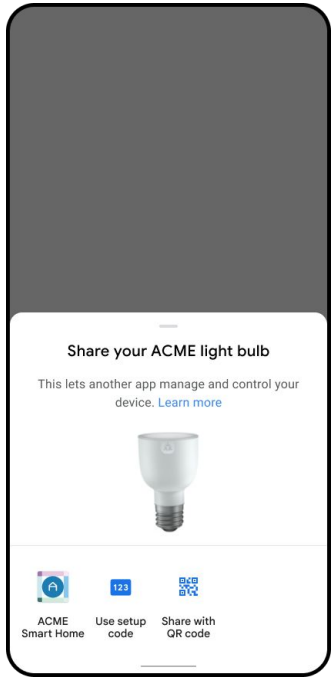


Commission Flow



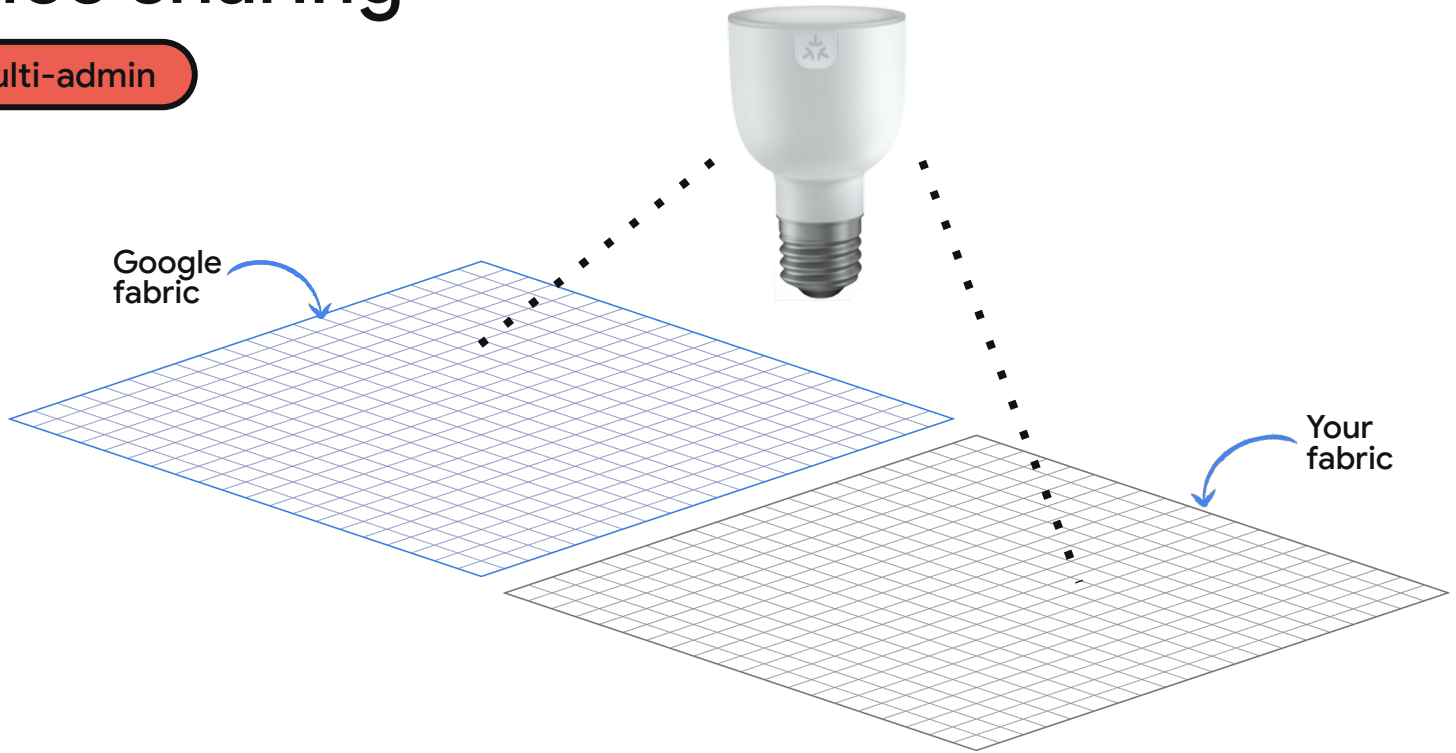
Device Sharing

Multi-Admin

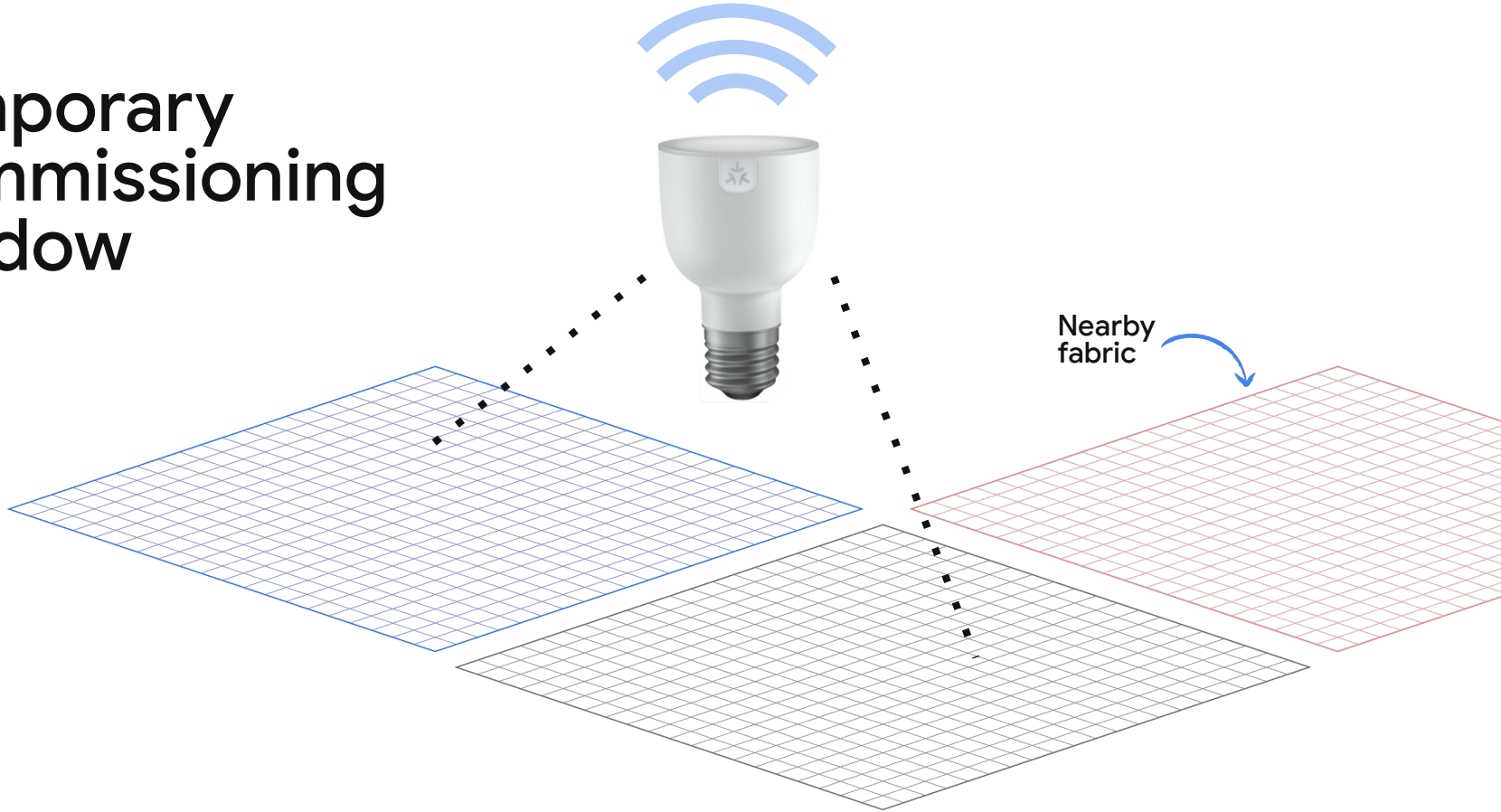


Device sharing

aka: Multi-admin



Temporary Commissioning window



Temporary Commissioning Window

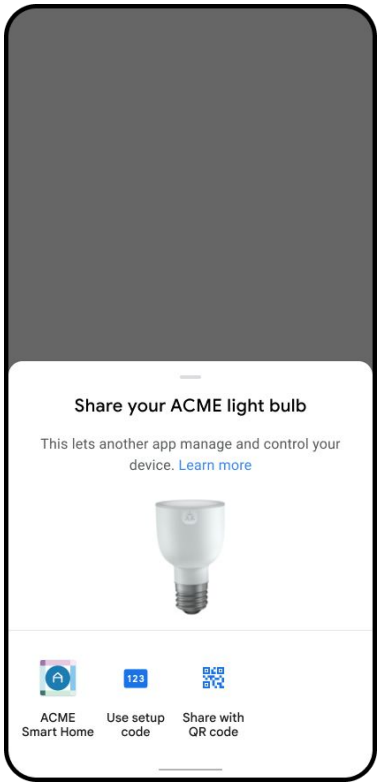
```
public boolean openPairingWindowWithPIN(
    long devicePtr,
    int duration,
    long iteration,
    int discriminator,
    long setupPinCode)
```

```
shareDeviceRequest = ShareDeviceRequest.builder()
        .setDeviceDescriptor(...)
        .setDeviceName(...)
        .setCommissioningWindow(
          CommissioningWindow.builder()
            .setDiscriminator(...)
            .setPasscode(...)
            .setWindowOpenMillis(SystemClock.elapsedRealtime())
            .setDurationSeconds(180)
            .build()
        )
        .build()
```

```
commissioningClient
  .shareDevice(shareDeviceRequest)
  .addOnSuccessListener { result ->
      shareDeviceLauncher.launch(IntentSenderRequest.Builder(result).build()
  }
  .addOnFailureListener { error ->
      …
  }
```

```
shareDeviceRequest = ShareDeviceRequest.builder()
        .setDeviceDescriptor(...)
        .setDeviceName(...)
        .setCommissioningWindow(
          CommissioningWindow.builder()
            .setDiscriminator(...)
            .setPasscode(...)
            .setWindowOpenMillis(SystemClock.elapsedRealtime())
            .setDurationSeconds(180)
            .build()
        )
        .build()
```

```
commissioningClient
  .shareDevice(shareDeviceRequest)
  .addOnSuccessListener { result ->
      shareDeviceLauncher.launch(IntentSenderRequest.Builder(result).build()
  }
  .addOnFailureListener { error ->
      …
  }
```



123

Use setup code

Share with QR code

Share with any app via code

Once the code is shared, the user launches commissioner app and proceeds with commissioning the device using the code received.

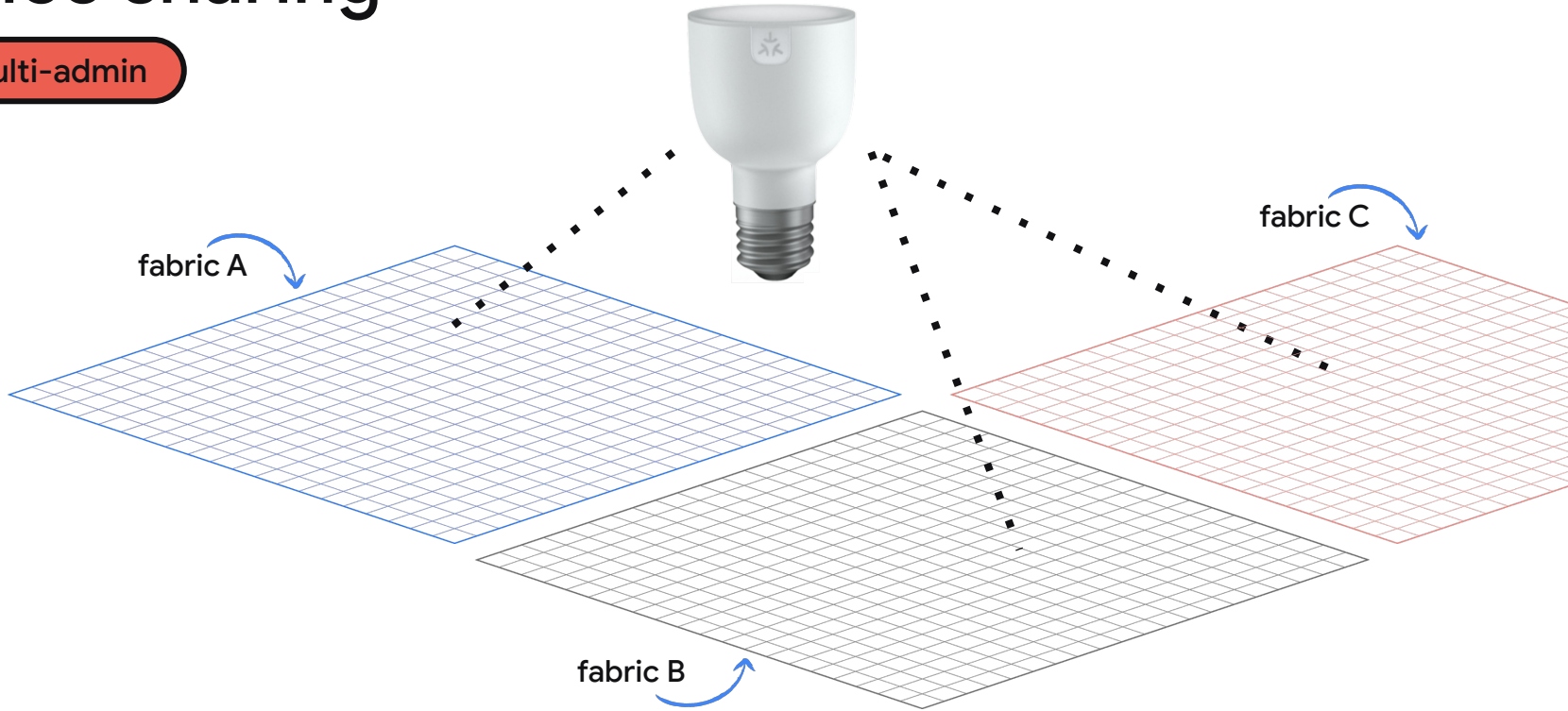
ACME Smart Home

Share with another app on Android

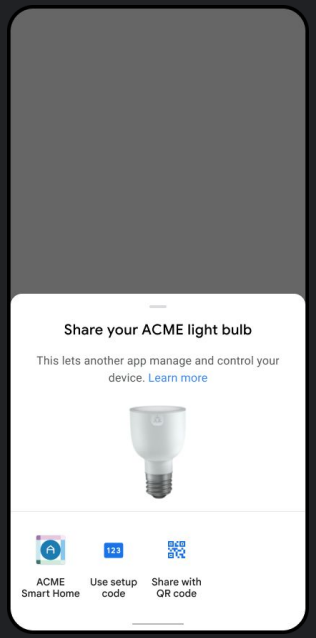
Target commissioner app receives an Intent and proceeds with commissioning the device using the code received.

Device sharing

aka: Multi-admin



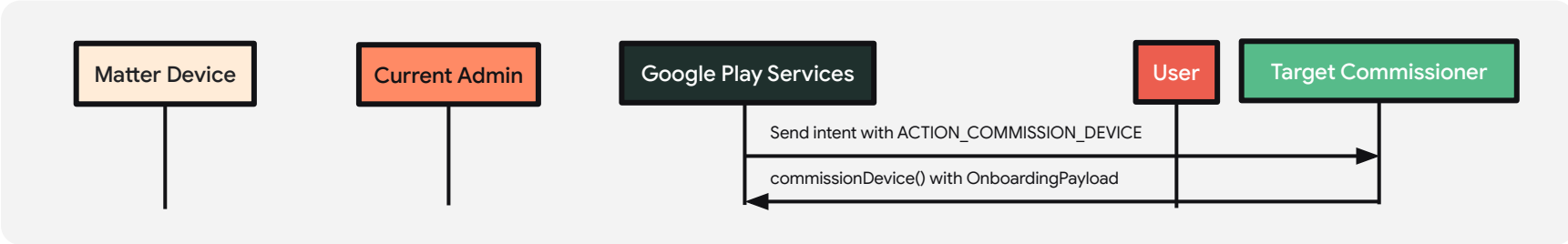
Register as a Commissioner



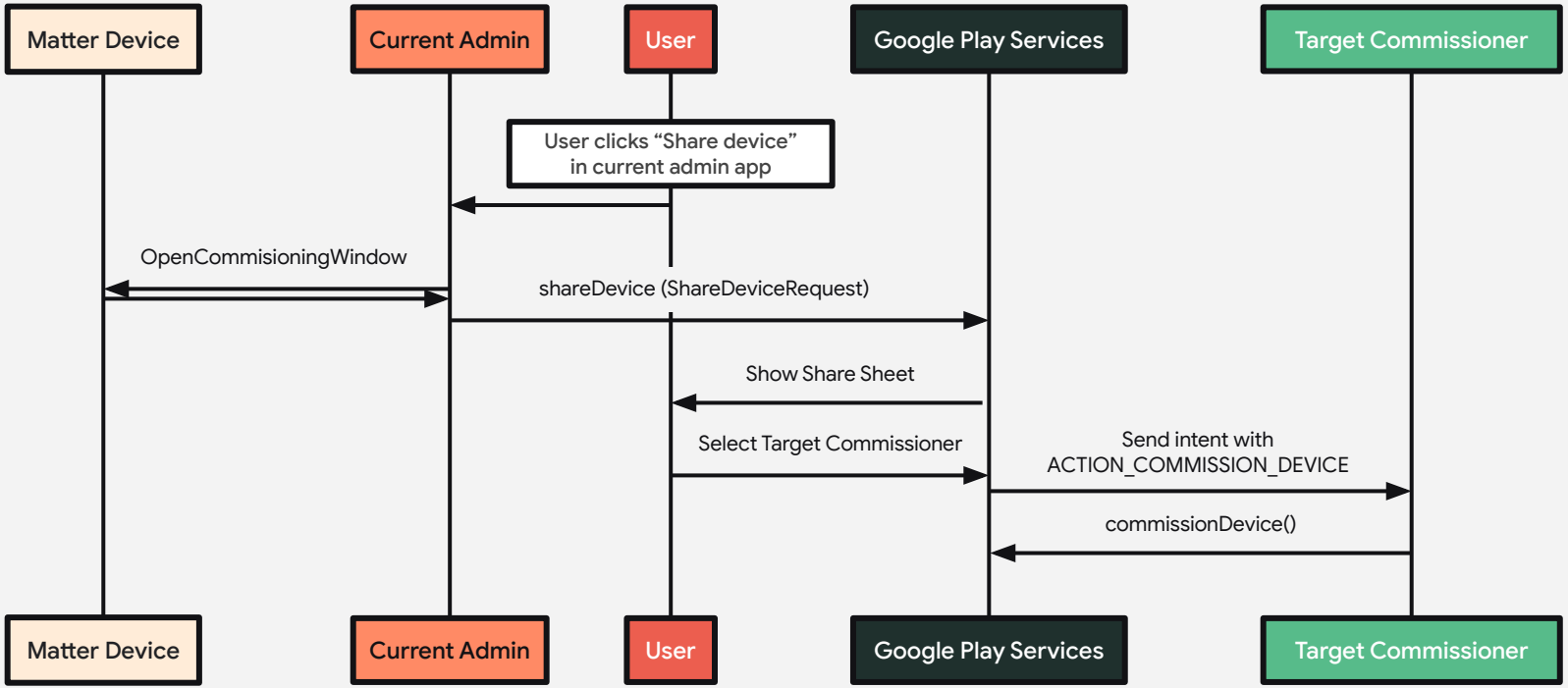

```
<intent-filter>
  <action
android:name="com.google.android.gms.connectedhome.matter.ACTION_COMMISSION_DEVICE"/>
  <category android:name="android.intent.category.DEFAULT"/>
</intent-filter>
```

List any apps on the phone which have registered to act as a commissioner excluding the application which initiated the sharing request.

Intent ACTION_COMMISSION_DEVICE

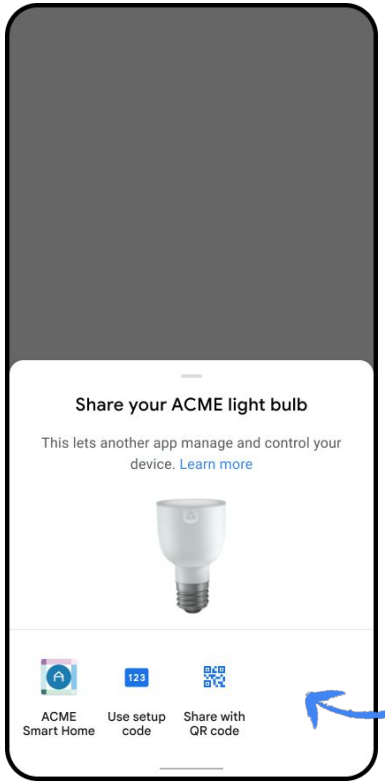


Key	Type
EXTRA_MANUAL_PAIRING_CODE	String
EXTRA_COMMISSIONING_WINDOW_EXPIRATION	long
EXTRA_DEVICE_NAME	String
EXTRA_VENDOR_ID	int
EXTRA_PRODUCT_ID	int
EXTRA_DEVICE_TYPE	int


Matter Device
Current Admin
User
Google Play Services
Target Commissioner
User clicks “Share device” in current admin app
OpenCommisioningWindow
shareDevice (ShareDeviceRequest)
Show Share Sheet
Select Target Commissioner
Send intent with ACTION_COMMISSION_DEVICE
commissionDevice()
Matter Device
Current Admin
User
Google Play Services
Target Commissioner

How to test device sharing?

How to test device sharing?

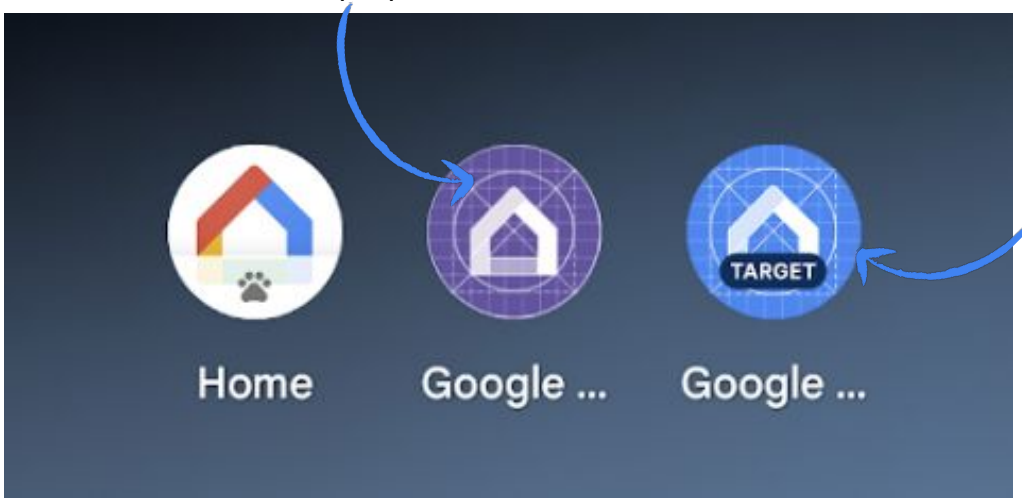


Since Sample App initiates “Device Sharing”, it does not show up in the Device Sharing Half Sheet.

How to test device sharing?

Two flavors of the Sample App: “default” and “targetcommissioner”

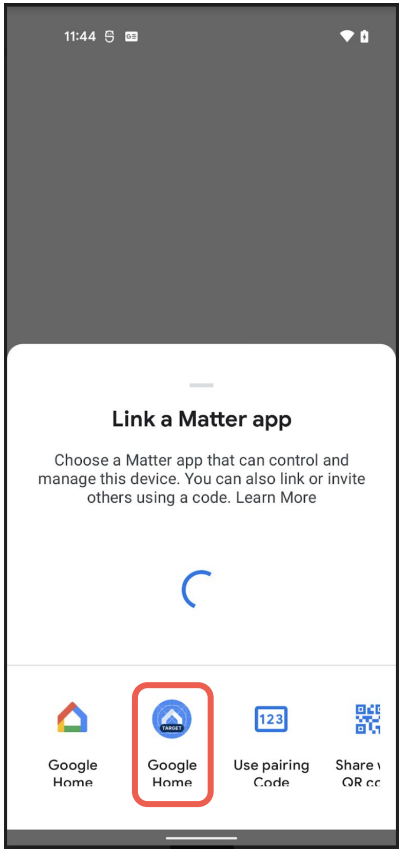
“default”
(purple icon and theme)



“targetcommissioner”
(blue icon and theme)

How to test device sharing?

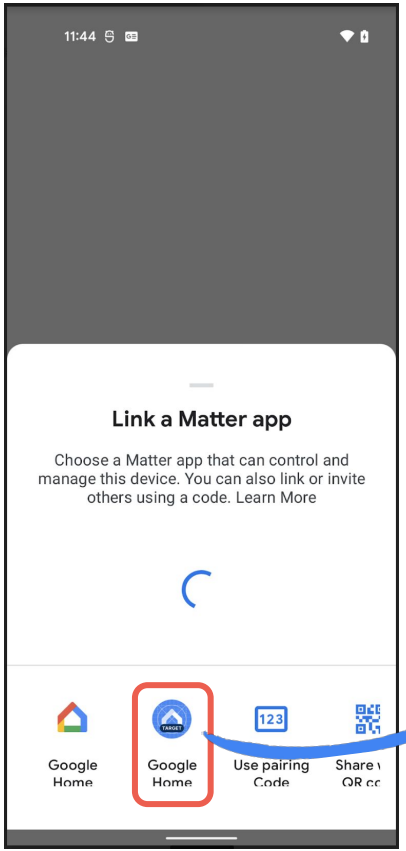
“default” flavor of Sample App
(purple icon and theme)
initiates device sharing

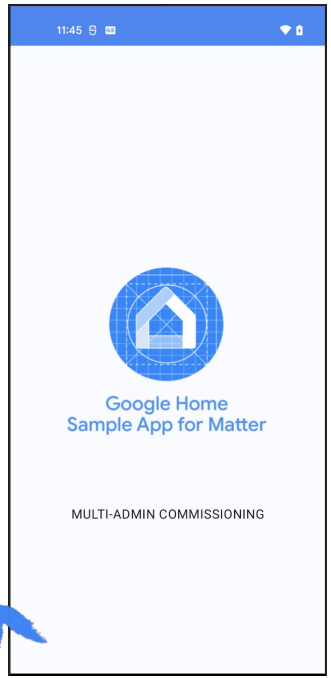


“targetcommissioner”
flavor of Sample App
(blue icon and theme)
is used as the target commissioner

How to test device sharing?

“default” flavor of Sample App
(purple icon and theme)
initiates device sharing





“targetcommissioner”
flavor of Sample App
(blue icon and theme)
is used as the target commissioner