Creating a robot app

1. Creating a project

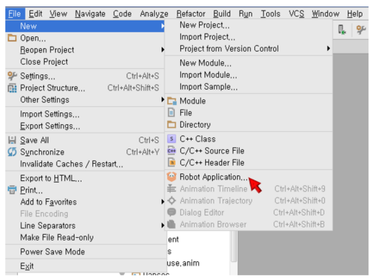
To create a robot application, we first create a standard Android project, which we then convert into a robot application.

Step	Action
	Select in **Android Studio**: **File** > **New** > **New Project** Name your application

.

Now that you have created your Android project, let's convert it into a robot application:

Creating Robot AppsNow that you have created your Android project, let's convert it into a robot app. Creating Robot AppsNow that you have created your Android project, let's convert it to a robot app.
After creating the project, select **File** > **New** > **Robot Application**.



Steps	Action
	Select the lowest Robot SDK version and module for robotify. Select **OK**.
	Synchronising projects with Gradle files

Results

Your project is now a robotics application!

So you should see the following changes:

The

- *robotsdk.xml* file has been added to the *assets/robot* directory
  The toolbar now has some new available tools **QiSDK** dependency has been added to the *build.gradle* file under your module. file under your module
- A *uses-feature tag has been added to your AndroidManifest.xml*

robotsdk.xml file contains the minimum robots SDK version for the project (previously selected one)

The new available tools allow you to start a new robots virtual machine

and connecting to real robots

.

QiSDK dependencies give you all the functionality you need to interact with Pepper

.

The *uses-feature* tag indicates that your app uses the control Pepper and will only be available on robot tablets

.

3. Implementing QiSDK and the robot life cycle

We've configured a robot app, now it's time for our code to control Pepper's body

.

Step	Action

	be in `onCreate` method to register the `MainActivity` up to **QiSDK** :
	
	`@Override`
	`protected void onCreate(Bundle savedInstanceState) {`
	`super.onCreate(savedInstanceState);`
	`// Register the RobotLifecycleCallbacks to this Activity.`
	`QiSDK.register(this, this);`
	`}`

	in `onDestroy` Unregister in method.
	
	`@Override`
	`protected void onDestroy() {`
	`// Unregister the RobotLifecycleCallbacks for this Activity.`
	`QiSDK.unregister(this, this);`
	`super.onDestroy();`
	`}`
	
	class='last'>**Hint:** You can also cancel all RobotLifecycleCallbacks for this campaign using `QiSDK.unregister(this`
	`unregister all RobotLifecycleCallbacks for this activity, being careful not to inadvertently d`
	`callbacks that should remain registered`

	envoy `MainActivity` class reversionary `RobotActivity` : `public class MainActivity extends RobotActivity` **Why?** This activity brings visual feedback when language interactions run, such as SpeechBar.

	class="first">envoy `MainActivity` class realise `RobotLifecycleCallbacks` connector:
	
	`public class MainActivity extends RobotActivity implements RobotLifecycleCallb`
	
	Why? This interface allows `RobotLifecycleCallbacks` to know `Activity` when it registers, gains, or loses RobotFocus.

	rewrite `onRobotFocusGained` , `onRobotFocusLost` and `onRobotFocusRefused` method:
	
	`@Override`
	`public void onRobotFocusGained(QiContext qiContext) {`
	`// The robot focus is gained.`
	`}`
	
	`@Override`
	`public void onRobotFocusLost() {`
	`// The robot focus is lost.`
	`}`
	
	`@Override`
	`public void onRobotFocusRefused(String reason) {`
	`// The robot focus is refused.`
	`}`
	
	Why? The onRobotFocusGained method is called when the associated Activity gets the robot Focus. After getting the F we can perform various actions on the robot. When onRobotFocusLost method is called, the related Activity loses the Fo and the operations will not be able to run on Pepper.
	
	Important
	
	`onRobotFocusGained` and `onRobotFocusLost` methods run in the background, and the UI thread is not blocked w QiSDK is used synchronously
	
	.