Mastering Focus and the Robot Lifecycle

Understanding Focus focus

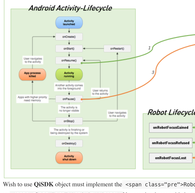
Robot Focus is `Activity` States necessary to run actions

It guarantees `Activity` Only one at a time： `Activity` Known as the owner of Robot Focus. This ensures that the robot focus owner has sole control of the robot

Robot focus is managed by a service that is used to give focus to activities. T his mechanism implies that `Activity` Gain or lose robot focus at any time.

Robot life cycle

QiSDK Provides a robot lifecycle that allows any `<span class="pre">Activity</span>` Have a robot focus.



Wish to use **QiSDK** object must implement the `<span class="pre">RobotLifecycleCallbacks</span>` interfaces. such as `<span class="pre">Activity</span>` You can implement this interface:

```
public class MyActivity extends RobotActivity implements RobotLifecycleCallbacks
```

```
<p>In addition, this object must be registered to the `<span class="pre">Activity</span>` in order to rec
<p>It has to be in `<span class="pre">onCreate</span>` method to register the</p>
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Register the RobotLifecycleCallbacks to this Activity.
    QiSDK.register(this, this);
}
```

```
<p>As well as, in `<span class="pre">onDestroy</span>` Unregister in method.</p>
```

```
@Override
protected void onDestroy() {
    // Unregister the RobotLifecycleCallbacks for this Activity.
    QiSDK.unregister(this, this);
    super.onDestroy();
}
```

Getting the Robot Spotlight

when `Activity` Getting the focus of Android (move to the foreground), it will request the robot's focus.

when `Activity` Getting the robot spotlight. `onRobotFocusGained` The callback function will be registered to the `Activity` Every single one of these `RobotLifecycleCallbacks` Called by.

```
void onRobotFocusGained(QiContext qiContext);
```

Warning

This callback runs in a **background thread**, so you can't manipulate the UI components directly in it. See also back to UI thread to run the code on the UI thread.

It provides a `QiContext` that allows you to.

- create actions,
- Creating resources for actions,
- Obtaining the services of a robot（services）.

create action:

```
// Create a Say action with a QiContext.
Say say = SayBuilder.with(qiContext)
                    .withText("Hello")
                    .build();
```

Creating Resources resource:

```
// Create an Animation with a QiContext.
Animation animation = AnimationBuilder.with(qiContext)
                                      .withResources(R.raw.elephant_a001)
                                      .build();
```

Access to services service:

```
// Get the HumanAwareness service with a QiContext.
HumanAwareness humanAwareness = qiContext.getHumanAwareness();
```

In this callback, you can run your actions:

run action:

```
// Run a synchronisedaction.
say.run();
```

```
// Run an asynchronousaction.
say.asyncv().run();
```

```
<div class="section" id="losing-the-robot-focus">
  <h3>Loss of robot focus<a class="headerlink" href="#losing-the-robot-focus" title="Permalink to t
  <p>If `<span class="pre">Activity</span>` Loss of robot focus, `<span class="pre">onRobotFocusLost</span>` The c
```

```
void onRobotFocusLost();
```

Warning

This callback runs in a **background thread**, so you can't manipulate the UI components directly in it. See also return to the UI thread to run your code in the UI thread.

```
    <p>About when `<span class="pre">Activity</span>` For more information on how you can lose focus.
```

Impact on actions

When this callback is called, `Activity` cannot run on Pepper until it regains focus on the robot actions:

```
// This will fail if robot focus is lost.
say.run();
```

In addition, if `onRobotFocusLost` is called while the operation is running, the execution of the operation will stop and an exception will be thrown: the

```
// This will raise an exception.
say.run();
```

```
say.asyncv().run().thenConsume(future -> {
    if (future.isSuccess()) {
        // This will not be called.
    } else if (future.hasError()) {
        // This will be called.
    }
});
```

Because Listeners can be triggered without robot focus, it should be removed `onRobotFocusLost` Callbacks under all Listeners:

```
// Remove listeners from LookAt.
if (lookAt != null) {
    lookAt.removeAllOnPolicyChangedListeners();
}
```

```
    <div class="section" id="impact-on-services">
      <h4>Impact on services services<a class="headerlink" href="#impact-on-services" title="Permal
      </h4>
      <p>Robot services are not affected when the robot focus is lost.</p>
      <p>This means that with the services created by the `<span class="pre">Future</span>` will continue
      </p>
```

```
// The Future will continue its execution.
Future<List<Human>> humansAroundFuture = humanAwareness.asyncv().getHumansAround();
```

```
// The listener will still be triggered.
humanAwareness.addOnHumansAroundChangedListener(listener);
```

```
    <p>Because Listeners can be triggered without the robot's focus, you should remove all the ob
      listeners:</p>
    <div class="highlight-java notranslate">
      <div class="highlight">
        <pre><span></span><span class="c1">// Remove listeners from HumanAwareness.</span>
```

if (humanAwareness != null) { humanAwareness.removeAllOnHumansAroundChangedListeners(); }

If a service listener is not removed from the `onRobotFocusLost` callback, the listener will be triggered for all application lifecycles. Listener is triggered for all application lifecycles.

Rejecting the Robot Spotlight

The focus can reject your `<span class="pre">Activity</span>` , in certain situations. For example, if the robot is in an unstable state. `<span class="pre">onRobotFocusRefused</span>` The callbacks will be used by every callback registered to the `<span class="pre">Activity</span>` upper `<span class="pre">RobotLifecycleCallbacks</span>` invocations:

```
void onRobotFocusRefused(String reason);
```

The reason for rejecting the focus is provided as a parameter.

Performance and limitations

incapable of guaranteeing `onRobotFocusGained` or `onRobotFocusRefused` will be called. In some cases, the `RobotLifecycleCallbacks` may never be notified of focus acquisition or focus rejection. The application should implement its own end-of-time mechanism to handle this contingency and avoid waiting indefinitely.

```
<div class="section" id="get-back-on-ui-thread">
```

Return to UI thread

You may want to update the UI in robot lifecycle callbacks. For example, you want to update the `TextView` .

To achieve this, you have two common possibilities .

Android-based programmes

`Activity` class providing `runOnUiThread` method runs a `Runnable` In the UI thread:

```
@Override
public void onRobotFocusGained(QiContext qiContext) {

  Say say = SayBuilder.with(qiContext)
                      .withText("Hello")
                      .build();

  // Synchronous call.
    say.run();
    // Update the TextView to notify that the Say action is done.
    runOnUiThread(() -> textView.setText("Done!"));
}
```