365Photo Documentation

The app implements an idle timeout for the screens, so the timer resets upon user interaction.

# MainActivity Documentation

This document describes the functionality, components, and purpose of the MainActivity class written in Kotlin. This class utilizes Android's AppCompatActivity as its base and integrates UI elements using Jetpack Compose, TTS via a custom SkillApi, and navigation between activities.

**Class Overview**

The MainActivity class serves as the entry point of the application. It initializes shared preferences, manages app timeout functionality, sets up UI components using Jetpack Compose, and provides functionality for interacting with a robot SDK for voice and activity transitions.

**Key Functionalities**

1. **Timeout Management**
2. **Voice Interaction using SkillApi**
3. **Navigation between activities**
4. **Jetpack Compose-based UI setup**

**Components and Methods**

**Variables**

* **sharedPreferences**: Manages app settings stored in shared preferences.
* **appTimeoutDuration**: Stores the timeout duration (default: 60 seconds).

**onCreate(Bundle?)**

**Description**

The entry point of the activity lifecycle. Initializes shared preferences, configures the timeout manager, sets up the UI using Compose, and triggers a voice prompt.

**Key Operations**

1. Retrieves app\_timeout\_duration from shared preferences.
2. Initializes TimeoutManager with the specified timeout duration.
3. Plays a TTS message asking the user if they are ready to move.
4. Sets the TakePhotoScreen UI.

**@Composable TakePhotoScreen(Context)**

**Description**

Creates a composable UI that allows users to:

* Navigate to the photo-taking screen.
* Access app settings.
* Exit the app.

**UI Components**

1. **Background Image**: Full-screen background image loaded using painterResource.
2. **Navigation Icon**: Top-left icon for exiting the app.
3. **"Let's Go!" Button**: Bottom button navigating to the TakePhotoActivity.
4. **Settings Icon**: Bottom-right icon opening SettingsActivity.

**@Composable LetsGoButton(onClick: () -> Unit)**

**Description**

Creates a circular button labeled "Let's Go!" styled with a custom color and font. Clicking the button triggers a navigation action.

**Parameters**

* onClick: Lambda function executed when the button is clicked.

**Timeout Management**

**onUserInteraction()**

Handles user interaction events and resets the app's timeout timer using TimeoutManager.

**Voice Interaction**

**mSkillCallback**

Defines the callback interface for handling voice recognition and TTS events.

**Callback Methods**:

* onSpeechParResult(String): Temporary speech recognition result.
* onStart(): Triggered when speech recognition starts.
* onStop(): Triggered when speech recognition stops.
* onVolumeChange(int): Handles volume changes in the recognition process.
* onQueryEnded(int): Triggered when a query ends.
* onQueryAsrResult(String): Provides the final ASR (Automatic Speech Recognition) result.

**say(String, Context)**

Plays text-to-speech (TTS) using the robot's voice service.

**Steps**:

1. Connects to the SkillApi.
2. Registers the voice callback (mSkillCallback).
3. Plays the provided text (text) using the robot's TTS service.
4. Displays appropriate toast messages for API connection or disconnection events.

**Parameters**

* text: The text to be spoken by the robot.
* context: The current activity context.

**Navigation**

**Intents**

* **To Photo Module**: Launches TakePhotoActivity when the "Let's Go!" button is clicked.
* **To Settings**: Opens SettingsActivity when the settings icon is clicked.
* **To External App**: Launches an external app using its package name (com.ainirobot.moduleapp).

**Flags**

* Intent.FLAG\_ACTIVITY\_NEW\_TASK: Ensures the new activity is launched in a separate task.

**UI Styling**

**Jetpack Compose Elements**

1. **Box**: Parent layout for stacking composable elements.
2. **IconButton**: Used for navigation and settings icons.
3. **Column**: Organizes the "Let's Go!" button with vertical alignment and padding.
4. **Image**: Sets the background image.

**Styling Features**

* Button colors and shapes are customized.
* Padding and alignment ensure a user-friendly design.
* Icon tint and sizes provide visual consistency.

**Dependencies**

**Robot SDK Integration**

* **SkillApi**: Provides voice services and TTS functionality.
* **TTSEntity**: Represents the TTS text entity.
* **ApiListener**: Manages API connection/disconnection events.

**Jetpack Compose**

* **Composable Annotations**: Used to define UI elements.
* **Modifiers**: Handle layout, padding, and alignment.

**Example Usage**

**Navigating to the Photo Module**

When the user clicks the "Let's Go!" button, an intent launches the TakePhotoActivity.

**Playing TTS**

The say() method triggers the robot to speak a message such as:

say("Are you ready to move to the photo spot?", applicationContext)

# **TakePhotoActivity Documentation**

This document provides an overview and detailed explanation of the TakePhotoActivity class. This class is responsible for managing a photo-taking workflow with countdowns and robot guidance.

**Class Overview**

The TakePhotoActivity class extends AppCompatActivity and is designed to:

* Handle photo capture workflows using camera extensions.
* Manage user permissions and states for the camera.
* Interact with a robot for navigation and positioning.
* Provide a UI for photo capturing, sharing, and retaking.

**Key Functionalities**

1. **Camera Extensions**: Integrates with camera APIs to provide advanced photo-taking capabilities (e.g., HDR, Bokeh).
2. **Robotic Interaction**: Uses a robot API for movement and voice commands.
3. **Countdown Timer**: Displays a countdown before capturing a photo.
4. **Photo Sharing**: Allows users to share captured photos via external apps.
5. **UI Management**: Dynamically updates UI elements based on user actions and system states.

**Components and Methods**

**Variables**

* **Photo and App States**:
  + photoProcessStarted: Indicates if the photo capture process has started.
  + countdownDuration: Countdown duration before photo capture (default: 10 seconds).
  + appTimeoutDuration: App timeout duration (default: 60 seconds).
  + readyPointScript: Voice script for the robot's "ready point" state.
  + readyPointWaitingDuration: Waiting duration at the ready point (default: 5 seconds).
  + retakePhoto: Indicates if the user opted to retake a photo.
  + isPaused: Tracks if the activity is paused.
  + isIdle: Tracks if the system is idle.
* **UI Components**:
  + homeBackButton: Button to navigate back to the home screen.
  + cameraShutter: Button to capture photos.
* **Camera States**:
  + captureUri: URI of the captured photo.
  + progressComplete: Tracks if the photo capture process is complete.
  + cameraExtensionsViewModel: ViewModel for managing camera states.
* **Audio**:
  + mediaPlayer: Media player for countdown sounds.
* **State Management**:
  + captureScreenViewState: Manages the UI state for the capture screen.
  + permissionState: Tracks camera permissions.

**Lifecycle Methods**

**onCreate(Bundle?)**

* Initializes the activity.
* Sets up shared preferences, permissions, and UI components.
* Prepares the robot and media player for interaction.
* Configures the camera ViewModel and state collectors.

**onPause()**

* Sets isPaused to true and closes the activity using finishAffinity().

**onDestroy()**

* Releases mediaPlayer resources to prevent memory leaks.

**onUserInteraction()**

* Resets the app timeout timer via TimeoutManager.resetTimer().

**Photo Workflow**

**startTakePhotoProcess()**

* Manages the photo-taking workflow, including:
  1. Robot navigation to the "stage" and "photo" positions.
  2. Displaying a countdown timer before capturing the photo.
  3. Handling retakes and transitions to the "display" position.

**moveToPosition(String, () -> Unit)**

* Moves the robot to a specified position (stage, photo, or display) and executes a callback when the robot arrives.

**closePhotoPreview()**

* Resets the capture screen to its initial state and restarts the camera preview.

**sharePhoto()**

* Shares the captured photo using Android's Intent system.

**Robot Integration**

**mMotionListener**

* Callback for handling robot navigation commands.

**resetHead()**

* Resets the robot's head position to its default state.

**say(String, Context)**

* Uses the robot's TTS (Text-to-Speech) system to play a voice message.

**UI Management**

**hideButtons()**

* Hides specific UI components during the photo process.

**showButtons()**

* Restores UI components after the photo process.

**startCountdown()**

* Displays a countdown timer before taking a photo and plays countdown audio.

**Permission Management**

**getCurrentPermissionState()**

* Checks the current camera permission status and returns it as a PermissionState.

**permissionState**

* Monitors changes in permission states and updates the UI accordingly.

**Camera Extensions**

**cameraExtensionsViewModel**

* Manages the camera's state and interactions.

**Camera Actions**

* **ShutterButtonClick**: Starts the photo-taking process.
* **SwitchCameraClick**: Switches between front and rear cameras.
* **ClosePhotoPreviewClick**: Closes the photo preview.
* **ProcessProgressComplete**: Updates the UI when the photo process is complete.

**Lifecycle Integration**

* Collects camera states and updates the UI dynamically based on the camera's readiness and capture progress.

**Media Management**

**mediaPlayer**

* Plays audio during the countdown timer and resets after playback.

**shareImage(Uri)**

* Shares the captured image via external apps, excluding specific apps like the default messaging app.

**Callbacks**

**mSkillCallback**

* Handles robot-specific callbacks for speech recognition and TTS events.

# SettingsActivity Documentation

This document provides an overview of the SettingsActivity class. This class allows users to configure application settings, including countdown duration, app timeout, text scripts, and wait duration. The activity integrates with SharedPreferences for persistent storage and validates user input before saving.

**Class Overview**

SettingsActivity is an AppCompatActivity designed to:

* Allow users to configure settings related to countdowns, timeouts, and robot interactions.
* Persist these settings using Android's SharedPreferences.
* Validate inputs to ensure they meet predefined criteria.
* Navigate back to the MainActivity after saving settings.

**Key Functionalities**

1. **Input Fields**: Capture user-configurable settings.
2. **Validation**: Ensure inputs are within acceptable ranges.
3. **Persistence**: Save settings to SharedPreferences.
4. **Navigation**: Return to MainActivity upon saving.

**Components and Methods**

**Variables**

* **UI Components**:
  + countdownInput: Input field for countdown duration.
  + appTimeoutInput: Input field for app timeout duration.
  + textScriptInput: Input field for the robot's text script.
  + waitDurationInput: Input field for wait duration at the ready point.
  + saveButton: Button to save the settings.
* **SharedPreferences**:
  + Stores and retrieves user settings persistently.

**Lifecycle Methods**

**onCreate(Bundle?)**

* Sets the activity's layout to activity\_settings.
* Initializes UI components.
* Retrieves and populates saved settings from SharedPreferences.
* Configures the saveButton to validate and save inputs when clicked.

**onUserInteraction()**

* Resets the app's timeout timer using TimeoutManager.resetTimer().

**Settings Retrieval and Population**

Upon activity creation, the class retrieves saved values from SharedPreferences and populates the corresponding input fields:

* **countdown\_duration**: Default value is 10 seconds.
* **app\_timeout\_duration**: Default value is 60 seconds.
* **ready\_script**: Default value is an empty string.
* **ready\_wait\_duration**: Default value is 5 seconds.

These values are set into their respective UI components using the setText() method.

**Input Validation**

**validateInputs()**

* Validates user input before saving.
* Ensures the following criteria:
  1. **Countdown Duration**: Must be between 5 and 20 seconds.
  2. **App Timeout**: Must be between 300 and 600 seconds.
  3. **Wait Duration**: Must be between 5 and 10 seconds.
  4. **Text Script**: Cannot be empty.
* Displays a Toast message if validation fails.

**Settings Persistence**

**saveSettings(countdown: Int, appTimeout: Int, textScript: String, waitDuration: Int)**

* Saves validated inputs to SharedPreferences using an editor.
* Displays a Toast message confirming the save operation.
* Calls launchMainActivity() to navigate back to the main screen.

**Navigation**

**launchMainActivity(context: Context)**

* Clears the back stack and launches MainActivity.
* Uses intent flags FLAG\_ACTIVITY\_CLEAR\_TOP and FLAG\_ACTIVITY\_NEW\_TASK to ensure a clean navigation transition.

**UI Behavior**

1. **Input Fields**:
   * Allow users to input custom values for countdown duration, app timeout, script text, and wait duration.
2. **Save Button**:
   * Validates inputs upon click.
   * Saves valid settings and navigates back to the main activity.

**Error Handling**

* Displays Toast messages for invalid inputs, such as:
  + "Countdown must be within 5 and 20 seconds."
  + "App timeout must be between 300 and 600 seconds."
  + "Wait duration must be between 5 and 10 seconds."
  + "Script text cannot be empty."

# TimeoutManager Documentation

The TimeoutManager object is a utility for managing global idle timeouts within the Android application. It ensures that when the user is idle for a specified duration, a timeout event is triggered, which can handle context-specific actions, such as navigating to another application or finishing the current activity.

## Class Overview

The TimeoutManager is a singleton object designed to:

* Monitor user activity and reset a timeout timer accordingly.
* Invoke a handler when the user remains idle beyond the specified duration.

### Key Features

1. **Idle State Management**: Tracks whether the application is in an idle state.
2. **Timeout Handling**: Executes a custom action upon timeout.
3. **Context-Specific Behavior**: Uses the provided Context to manage navigation and activity lifecycle.

## Components and Methods

### **Variables**

* **isIdle**: Tracks whether the application is currently idle (true by default).
* **handler**: An Android Handler instance for scheduling and managing timeout tasks.
* **runnable**: A Runnable task executed when the timeout occurs.
* **timeoutDuration**: Duration of the timeout in milliseconds (default: 60,000ms or 1 minute).

### **Methods**

#### initialize(context: Context, appTimeoutDuration: Long)

##### Description

Initializes the TimeoutManager with a specific timeout duration and sets up the handler and runnable for managing timeouts.

##### Parameters

* **context**: The application or activity context used for handling timeout actions.
* **appTimeoutDuration**: The timeout duration in milliseconds.

##### Behavior

1. Sets the global timeoutDuration to the specified value.
2. Initializes the handler for managing the timeout runnable.
3. Defines a runnable that sets the isIdle flag to true and calls handleIdleTimeout().
4. Invokes resetTimer() to start the timeout countdown.

#### resetTimer()

##### Description

Resets the timeout timer, preventing the timeout event from triggering if the user remains active.

##### Behavior

1. Sets isIdle to false to indicate activity.
2. Removes any pending runnable callbacks from the handler.
3. Posts a delayed execution of the runnable based on the timeoutDuration.

#### handleIdleTimeout(context: Context)

##### Description

Executes the timeout behavior when the application enters an idle state.This exits the application and opens XiaoBao.

##### Parameters

* **context**: The application or activity context used for navigation and activity lifecycle management.

##### Behavior

1. Logs a message indicating the idle timeout has been triggered.
2. Attempts to launch an external application specified by its package name (com.ainirobot.moduleapp).
3. If the context is an instance of AppCompatActivity, calls finishAffinity() to close the current activity and its stack.

## Usage Example

### Initialization

To use the TimeoutManager, initialize it in the onCreate method of your main activity:

TimeoutManager.initialize(this, 120 \* 1000L) // Set timeout duration to 2 minutes

### Reset Timer on Interaction

Reset the timer whenever the user interacts with the application:

override fun onUserInteraction() {

super.onUserInteraction()

TimeoutManager.resetTimer()

}

## Default Behavior

1. Upon initialization, the timeout countdown starts based on the timeoutDuration.
2. If no user activity occurs during the countdown:
   * XiaoBao (com.ainirobot.moduleapp) is launched.
   * The current activity and its task stack are finished.
3. Any user interaction during the countdown resets the timer, preventing the timeout.