



Articles » Mobile Development » Android » General

Painless AsyncTask and ProgressDialog Usage

By **Mike114**, 25 Feb 2011

★★★★★ 4.92 (13 votes)

[Download source code - 22.7 KB](#)

Introduction

In this article, I'll try to show how to create a small framework which can help you develop applications with asynchronous operations and progress dialogs without code duplication and with configuration changes handling.

Background

To avoid application freezing on complex tasks, their execution should be moved to a separate thread. As mentioned in the Android Developer reference, **AsyncTask** usage is the best way to do this. To wait for task completion and report about the current state, a **ProgressDialog** is often used.

In the perfect world, they work really well and you do not need to write a lot of code for this. Several snippets will be enough. But this fairytale ends when you start to change the configuration (at least rotate screen) because Android recreates the activity, or when you try to repeat the same functionality in different activities.

Activity cleanup

So I decided to get rid of supportive code from the activity and move it to a separate class to reuse it in the project. The activity should only create a particular task, pass it to this class, and provide a callback to process task completion. The rest of the functionality (show dialog, update progress message, close dialog, call completion handler) is common, and should be implemented in this new class **AsyncTaskManager**.

But besides task creation and completion handling, the activity should keep this task between recreations on configuration changes. This functionality can also be delegated to **AsyncTaskManager**.

Finally, the activity should contain several lines of code to do the following things:

1. Create **AsyncTaskManager** on activity creation

```
mAsyncTaskManager = new AsyncTaskManager(this, this);
```

2. Let **AsyncTaskManager** handle the retained task and run it automatically

```
mAsyncTaskManager.handleRetainedTask(getLastNonConfigurationInstance());
```

3. Create new **AsyncTask**, setup **AsyncTaskManager** with it, and run it

```
mAsyncTaskManager.setupTask(new Task(getResources()));
```

4. Let **AsyncTaskManager** to retain the task

```
return mAsyncTaskManager.retainTask();
```

5. Process task completion

To reduce coupling between **AsyncTaskManager** and the activity, I created an interface that should be implemented by any activity which uses the same approach.

```
public interface OnTaskCompleteListener {
    void onTaskComplete(Task task);
}
```

Task in parameters is initially created as an asynchronous task, so I can get results or check if this task was cancelled.

AsyncTaskManager implementation

The **AsyncTaskManager** is responsible now for **AsyncTask** and **ProgressDialog** management, and its logic can be implemented as follows:

1. Create dialog on start
2. On task assignment, somehow bind the task state (progress message) to the dialog and run the task
3. Unbind the dialog from the task when the task should be retained and bind back on restore
4. Cancel task on dialog cancel
5. Dismiss the dialog on task completion
6. Report about completion to activity (via **OnTaskCompleteListener**) on task cancel or complete

To organize such a communication, I introduced another interface to bind the task to **AsyncTaskManager**:

```
public interface IProgressTracker {
    void onProgress(String message);
    void onComplete();
}
```

And implemented it:

```
@Override
public void onProgress(String message) {
```

```

if (!mProgressDialog.isShowing()) {
    mProgressDialog.show();
}
mProgressDialog.setMessage(message);
}

@Override
public void onCancel(DialogInterface dialog) {
    mAsyncTask.cancel(true);
    mTaskCompleteListener.onTaskComplete(mAsyncTask);
    mAsyncTask = null;
}

```

My implementation of **AsyncTask** has a special method to assign the **IProgressTracker** instance to allow to easily attach and detach a task from the rest of code that is recreated every time on configuration changes:

```

public void setProgressTracker(IProgressTracker progressTracker) {
    mProgressTracker = progressTracker;
    if (mProgressTracker != null) {
        mProgressTracker.onProgress(mProgressMessage);
        if (mResult != null) {
            mProgressTracker.onComplete();
        }
    }
}

```

As you can see, **AsyncTask** keeps the progress message and operation result in fields and calls the specific method on attach. So if your task has completed when the activity was destroyed, on new **setProgressTracker** call, all callbacks will be run and the activity will receive a completion notification.

AsyncTask implementation

Custom **AsyncTask** implementation is pretty straightforward and looks like a standard solution except the mentioned setup method. Besides that, the **onProgressUpdate** method should call **IProgressTracker.onProgress**, and the **onPostExecute** method should call **IProgressTracker.onComplete**.

Conclusion

Now any activity can use this solution. Just add five lines of code to it and use **Task** from the sample code as the super class for your own tasks. You can rotate your phone as you wish in any direction, but all these configuration changes will be correctly handled.

This approach was implemented in my new application [Lingo Quiz Full](#) and [Lingo Quiz Lite](#) when it imports words from a file or request translations from Google or setup the initial set of dictionaries.

Useful links

- [AsyncTask class](#)
- [Painless threading](#)
- [Android dialogs](#)

License

This article, along with any associated source code and files, is licensed under [The Code Project Open License \(CPOL\)](#)

About the Author



Mike114


Technical Lead BCS Technology

Australia 

Member

No Biography provided

Comments and Discussions

 **36 messages** have been posted for this article Visit <http://www.codeproject.com/Articles/162201/Painless-AsyncTask-and-ProgressDialog-Usage> to post and view comments on this article, or click [here](#) to get a print view with messages.

[Permalink](#) | [Advertise](#) | [Privacy](#) | [Mobile](#)
Web01 | 2.6.121130.2 | Last Updated 25 Feb 2011

Article Copyright 2011 by Mike114
Everything else Copyright © [CodeProject](#), 1999-2012
[Terms of Use](#)