

# Threads - AsyncTask



**Ever seen?**

**Application not responding**

# Threads - what is a thread

“Multiple threads can exist within one process, executing concurrently and sharing resources such as memory, while different processes do not share these resources. In particular, the threads of a process share its executable code and the values of its variables at any given time.”

[https://en.wikipedia.org/wiki/Thread\\_\(computing\)](https://en.wikipedia.org/wiki/Thread_(computing))

# Threads in Android

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            // a potentially time consuming task
            final Bitmap bitmap =
                processBitmap("image.png");
            mImageView.post(new Runnable() {
                public void run() {
                    mImageView.setImageBitmap(bitmap);
                }
            });
        }
    }).start();
}
```

```
// This implementation is thread-safe: the background operation is done from a separate
// thread while the ImageView is always manipulated from the UI thread.
```

**Huah?**

**A bit raw?**

# **Threads in Android - simplified**

## **Introducing AsyncTask**

# AsyncTask

Let's say you want to download URLs and update the GUI when done

# AsyncTask

```
private class DownloadFilesTask extends AsyncTask< URL, Integer, Long> {  
    ...  
}
```

**URL** - the type of the parameters sent to the task upon execution



# AsyncTask - doing

```
@Override
protected Long doInBackground(URL... urls) {
    int count = urls.length;
    long totalSize = 0;
    for (int i = 0; i < count; i++) {
        totalSize += Downloader.downloadFile(urls[i]);
        publishProgress((int) ((i / (float) count) * 100));
        // Escape early if cancel() is called
        if (isCancelled()) break;
    }
    return totalSize;
}
```

# AsyncTask - starting

```
new DownloadFilesTask().execute(url1, url2, url3);
```

Where url1, url2 and url3 are the URL instances we want to pass to the doInBackground method.

`execute` is a method inherited from `AsyncTask`.

# AsyncTask

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {  
    ...  
}
```

URL - the type of the parameters sent to the task upon execution

Progress - the type of the progress units published during the background computation

**Result**, the type of the result of the background computation

# AsyncTask - when done

```
protected void onPostExecute(Long result) {  
    showDialog("Downloaded " + result + " bytes");  
}
```

# AsyncTask

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {  
    ...  
}
```

URL - the type of the parameters sent to the task upon execution

**Progress** - the type of the progress units published during the background computation

Result, the type of the result of the background computation

# AsyncTask - updating while working

```
protected void onProgressUpdate(Integer... progress) {  
    setProgressPercent(progress[0]);  
}
```

# **AsyncTask - putting the pieces together**

# AsyncTask - putting the pieces together

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
    @Override
    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalSize = 0;
        for (int i = 0; i < count; i++) {
            totalSize += Downloader.downloadFile(urls[i]);
            publishProgress((int) ((i / (float) count) * 100));
            // Escape early if cancel() is called
            if (isCancelled()) break;
        }
        return totalSize;
    }

    @Override
    protected void onProgressUpdate(Integer... progress) {
        setProgressPercent(progress[0]);
    }

    @Override
    protected void onPostExecute(Long result) {
        showDialog("Downloaded " + result + " bytes");
    }
}
```



# AsyncTask - starting

```
new DownloadFilesTask().execute(url1, url2, url3);
```

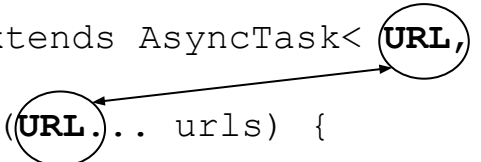
...

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {  
    @Override  
    protected Long doInBackground(URL .. urls) {  
        ...  
    }  
    ...  
}
```

The diagram illustrates the execution flow of the AsyncTask. It shows three call sites: the `execute` method call in the first code block, the `AsyncTask` constructor in the second code block, and the `doInBackground` method in the third code block. Arrows indicate the flow of data: one arrow points from the `execute` call to the `doInBackground` method, another points from the `execute` call to the `AsyncTask` constructor, and a third points from the `doInBackground` method back to the `AsyncTask` constructor. The `URL` parameter in the constructor and the `URL` parameter in the `doInBackground` method are circled, and the `URL` parameter in the `execute` call is also circled, showing the data flow between these elements.

# AsyncTask - doing

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {  
    protected Long doInBackground(URL... urls) {  
        ...  
        publishProgress((int) ((i / (float) count) * 100));  
        ...  
        return totalSize;  
    }  
  
    protected void onProgressUpdate(Integer... progress) {  
        setProgressPercent(progress[0]);  
    }  
  
    protected void onPostExecute(Long result) {  
        showDialog("Downloaded " + result + " bytes");  
    }  
}
```



# AsyncTask - updating while working

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {  
  
    protected Long doInBackground(URL... urls) {  
        ...  
        publishProgress((int) ((i / (float) count) * 100));  
        ...  
        return totalSize;  
    }  
  
    protected void onProgressUpdate(Integer.. progress) {  
        setProgressPercent(progress[0]);  
    }  
  
    protected void onPostExecute(Long result) {  
        showDialog("Downloaded " + result + " bytes");  
    }  
}
```

The diagram illustrates the relationship between the `Integer` type in the `AsyncTask` generic type and the `Integer` parameter in the `onProgressUpdate` method. It also shows that the integer value passed to `publishProgress` is of type `Integer`.

# AsyncTask - when done

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {  
    protected Long doInBackground(URL... urls) {  
        ....  
        publishProgress((int) ((i / (float) count) * 100));  
        ...  
        return totalSize;  
    }  
    protected void onProgressUpdate(Integer... progress) {  
        setProgressPercent(progress[0]);  
    }  
    protected void onPostExecute (Long result) {  
        showDialog("Downloaded " + result + " bytes");  
    }  
}
```

```
graph TD; A((Long)) --> B[return totalSize;]; B --> C[onPostExecute Long result]; B --> D[publishProgress];
```