## ntroduction

Android updates its User Interface via the main UI Thread only. Main Thread handles dispatching of events, updating UI components and interacting with other components.

Android Main Thread (UI Thread) cannot be used for performing time consuming operations or blocking operations since:

- These block the UI. This is important as Android displays an "Application not responding" (ANR) dialog if activities do not react within 5 seconds.
- Android UI toolkit is not thread safe. Updating UI via background threads can create problems.

Android provides the following mechanisms for performing time consuming operations (Threads) and updating the UI.

- Activity.runOnUiThread(Runnable)
- View.post(Runnable)
- View.postDelayed(Runnable, long)
- Handlers
- AsyncTask

## Threads

Android supports the standard Java Thread. It supports both ways of creating Threads, either providing a new class that extends the Thread and overriding its run() method, or by providing a new Thread instance with a Runnable object during its creation.

```
new Thread(new Runnable() {
    public void run() {
      //do time consuming operations
      });
    }
}).start();
```

Threads can only be used to do time consuming operations; they cannot be used to update the UI. For updating the UI, Handlers and AsyncTask are used.

## Handlers

The Handler class present in the android.os package can be used to update the UI, as each Handler instance is associated with a main thread and its message queue. When you create a new Handler, it is bound to the thread / message queue of the thread that is creating it

The Handler allows you to send and process Message (containing description and arbitrary data object that can be sent to a Handler), and Runnable objects, associated with a thread's MessageQueue.

There are two main uses for a Handler:

- To schedule Messages and Runnables to be executed at some point in the future

- To enqueue an action to be performed on a different thread than your own.

Scheduling messages is accomplished with the

- post(Runnable)
- postAtTime(Runnable, long)
- postDelayed(Runnable, long)
- sendEmptyMessage(int)
- sendMessage(Message)
- sendMessageAtTime(Message, long)
- sendMessageDelayed(Message, long)

Post methods are used to equeue Runnable objects to the thread message queue and send methods to enqueue Message objects to the thread message queue.

The following example shows how to use the Handler to update the UI by using the post and send methods. It shows the code required to update the progressbar every 500 milliseconds.

## Using post Methods

```
package com.test.progress;

import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ProgressBar;

public class ProgressActivity extends Activity implements OnClickListener {
        private Handler handler;
        private ProgressBar progressBar;
        private Button progressButton;
        /** Called when the activity is first created. */
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.main);
                progressBar = (ProgressBar) findViewById(R.id.progressBar);
                progressButton = (Button) findViewById(R.id.progBtn);
                progressButton.setOnClickListener(this);
        }
```

```
         public void startProgress() {
                  Runnable runnable = new Runnable() {
                           @Override
                           public void run() {
                                    for (int i = 0; i <= 10; i++) {
                                             final int value = i;
                                             try {
                                                      Thread.sleep(500);
                                             } catch (InterruptedException e) {
                                                      e.printStackTrace();
                                             }
                                             handler.post(new Runnable() {
                                                      @Override
                                                      public void run() {

progressBar.setProgress(value);
                                                      }
                                             });
                                    }
                           }
                  };
                  new Thread(runnable).start();
         }

         @Override
         public void onClick(View v) {
                  // TODO Auto-generated method stub
                  if (v == progressButton) {
                           startProgress();
                  }
         }
}
```

## Using send Methods

Here we create a Handler object with an associated callback method to handle the received messages (this is the method that will perform the UI update). From the background thread you will need to send messages to the handler.

```
package com.test.progress;
import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ProgressBar;

public class ProgressActivity1 extends Activity implements OnClickListener {
        private Handler handler;
        private ProgressBar progressBar;
        private Button progressButton;
        /** Called when the activity is first created. */
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.main);
                progressBar = (ProgressBar) findViewById(R.id.progressBar);
                progressButton = (Button) findViewById(R.id.progBtn);
                progressButton.setOnClickListener(this);
        }

        public void startProgress() {
                handler = new Handler(){
                        @Override
                        public void handleMessage(Message msg) {
                                // TODO Auto-generated method stub
                                super.handleMessage(msg);
                                int value  = progressBar.getProgress();
                                progressBar.setProgress(value + 5);
                        }
                };

                Runnable runnable = new Runnable() {
                        @Override
                        public void run() {
                                for (int i = 0; i <= 10; i++) {
                                        try {
                                                Thread.sleep(500);
                                        } catch (InterruptedException e) {
                                                e.printStackTrace();
                                        }
                                        handler.sendMessage(handler.obtainMessage());
                                }
                        }
                };
                new Thread(runnable).start();
        }

        @Override
        public void onClick(View v) {
                // TODO Auto-generated method stub
                if (v == progressButton) {
                        startProgress();
                }
        }
}
```

## AsyncTask

AsyncTask is present in the android.os package and allows background operations to perform and publish results on the UI thread without having to manipulate threads and/or handlers.

An asynchronous task is defined by a computation that runs on a background thread whose result is published on the UI thread.

An asynchronous task is defined by:

- 3 generic types, called Params, Progress and Result, and
- 4 methods called onPreExecute, doInBackground, onProgressUpdate and onPostExecute.

The three types used by an asynchronous task are the following:

- Params - 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.

Not all types are used by an asynchronous task. To mark a type as unused, simply use the type Void:

```
private class MyTask extends AsyncTask { ... }
```

When an asynchronous task is executed, the task goes through 4 steps:

- onPreExecute() - invoked on the UI thread immediately after the task is executed. This step is normally used to setup the task, for instance by showing a progress bar in the user interface.
- doInBackground(Params...) - invoked on the background thread immediately after onPreExecute() finishes executing. This step is used to perform background computation that can take a long time.
- The parameters of the asynchronous task are passed to this step, with the result of the computation being returned by this step and getting passed back to the last step.
- This step can also use publishProgress(Progress...) to publish one or more units of progress. These values are published on the UI thread, in the onProgressUpdate(Progress...) step.
- onProgressUpdate(Progress...) - invoked on the UI thread after a call to publishProgress(Progress...).
- The timing of the execution is undefined, and this method is used to display any form of progress

in the user interface while the background computation is still executing. For instance, it can be used to animate a progress bar or show logs in a text field.

- onPostExecute(Result) invoked on the UI thread after the background computation finishes. The result of the background computation is passed to this step as a parameter.

There are a few threading rules that must be followed for this class to work properly:

- The task instance must be created on the UI thread.
- execute(Params...) must be invoked on the UI thread.
- Do not call onPreExecute(), onPostExecute(Result), doInBackground(Params...), onProgressUpdate(Progress...) manually.
- The task can be executed only once (an exception will be thrown if a second execution is attempted.)

ref:http://developer.samsung.com/android/technical-docs/Using-Threads-Handlers-and-AsyncTask