GUI code

/*spinner is the drop down list*/

```
  Spinner spinner = (Spinner) findViewById(R.id.spinner);

  ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(
              this, R.array.websites, android.R.layout.simple_spinner_item);

  adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)
;
  spinner.setAdapter(adapter);
  spinner.setOnItemSelectedListener(new MyOnItemSelectedListener());


/*Button to connect*/

  final Button Broker = ((Button) findViewById(R.id.broker_button));

   Broker.setOnClickListener(new OnClickListener() {
               @Override
  public void onClick(View v) {
                     Log.d("ClientActivity", "C:in activity...");

  AndroidParser_service.actionStart(getApplicationContext());

      }
```

});

/*event handlers

```
public class MyOnItemSelectedListener implements OnItemSelectedListener{
      public void onItemSelected(AdapterView<?> parent,
              View view, int pos, long id)
      {
            Log.d("ClientActivity in selecting the menu", "C:in activity...");
            String str=parent.getItemAtPosition(pos).toString();
            Log.d("ClientActivity in the selected menu value", str);
            System.out.println(str);

            SharedPreferences settings =  getSharedPreferences
                        (AndroidParser_service.APP_ID, 0);
            SharedPreferences.Editor editor = settings.edit();
            editor.putString("broker", "172.16.33.142");

            if(str.equalsIgnoreCase("nitk"))
            {
                  Log.d("ClientActivity inside nitk", "nitk");
                  editor.putString("topic","nitk");
                  Log.d("ClientActivity inside nitk", "nitk");
                  editor.commit();

            }
            else if(str.equalsIgnoreCase("iisc"))
            {
                  Log.d("ClientActivity inside iisc", "iisc");
                  editor.putString("topic","iisc");
                  editor.commit();
            }
            else if(str.equalsIgnoreCase("nptel"))
            {
                  Log.d("ClientActivity inside nptel", "nptel");
```

```
                editor.putString("topic","nptel");
                editor.commit();
            }
            else
            {
                Log.d("ClientActivity inside gre", "gre");
                editor.putString("topic","gre");
                editor.commit();
            }


                //if(str.equalsIgnoreCase("nptel"))
                //editor.putString("topic","nptel");
            //editor.commit();






 //              AndroidParser_service.actionStart(getApplicationContext());
             Toast.makeText(parent.getContext(), "The site is " +
            parent.getItemAtPosition(pos).toString(), Toast.LENGTH_LONG).show();

        }
         public void onNothingSelected(AdapterView<?> parent)
        {
            //do nothing
        }
    }
```

Connect code

```
/*
        * (Re-)connect to the message broker
        */
      private boolean connectToBroker() {
            Log.d("ClientActivity", "C: inside connectToBroker");
            try {
                  // try to connect
                  mqttClient
                              .connect(generateClientId(), cleanStart,
      keepAliveSeconds);

                  //
                  // inform the app that the app has successfully connected
                  broadcastServiceStatus("Connected");

                  // we are connected
                  connectionStatus = MQTTConnectionStatus.CONNECTED;

                  // we need to wake up the phone's CPU frequently enough so
      that the
                  // keep alive messages can be sent
                  // we schedule the first one of these now
                  scheduleNextPing();

                  return true;
            } catch (MqttException e) {
                  // something went wrong!

                  connectionStatus =
      MQTTConnectionStatus.NOTCONNECTED_UNKNOWNREASON;

                  //
                  // inform the app that we failed to connect so that it can
      update
                  // the UI accordingly
                  broadcastServiceStatus("Unable to connect");

                  //
                  // inform the user (for times when the Activity UI isn't
      running)
                  // that we failed to connect
                  notifyUser("Unable to connect", "MQTT",
                              "Unable to connect - will retry later");

                  // if something has failed, we wait for one keep-alive period
      before
                  // trying again
                  // in a real implementation, you would probably want to keep
      count
                  // of how many times you attempt this, and stop trying after a
                  // certain number, or length of time - rather than keep trying
                  // forever.
                  // a failure is often an intermittent network issue, however,
      so
                  // some limited retry is a good idea
                  scheduleNextPing();

                  return false;
            }
      }
```

Publish code

/*pinging once every hour

```
while(true)
    {
        try
        {

            obj.publishToTopic("iisc");
            obj.publishToTopic("nitk");
            obj.publishToTopic("gre");
            Thread.sleep ( 60 * 60 * 1000 ) ;
        }
        catch(Exception e)
        {
            System.out.println(e);
        }

    }
    }
```

/*finding the difference between two updates

```
	public static String diff(String str1, String str2)
	{
		int index = str1.lastIndexOf(str2);

		if (index == 0)
		{
			return str1.substring(str2.length());
		}


		return str1.substring(0,index);


	}
/*checking for nitk website, the same holds for the other sites

	if(nitk_flag == 1)
		{

			nitk_str1=list.elementAt (0).toPlainTextString();

			//Because only nitk news has garbage
			if(topicname == "nitk")
			{
				nitk_str1 = nitk_str1.replaceAll("&nbsp;", " ");
				nitk_str1 = nitk_str1.replaceAll("\\(.*?\\)", "");
			}

			nitk_flag = 0;
		}
		else
		{

			nitk_str2=list.elementAt (0).toPlainTextString();

			//Because only nitk news has garbage
			if(topicname == "nitk")
			{
				nitk_str2 = nitk_str2.replaceAll("&nbsp;", " ");
				nitk_str2 = nitk_str2.replaceAll("\\(.*?\\)", "");
			}

			nitk_flag = 1;
		}

		if(nitk_str1.equals(nitk_str2))
		{

			equalFlag=1;
			//maintain some flag here and don't call mqttClient.publish
method
		}
		else
		{
			//there is some update. difference between the two strings
			//send that update to broker through mqttClient.publish method
			equalFlag = 0;
```

```
                if(nitk_str1.length() < nitk_str2.length())
                {
                      htmlString = diff(nitk_str2,nitk_str1);
                }
                else
                {
                      htmlString = diff(nitk_str1,nitk_str2);
                }
          }
}
```

Subscribe code

```
/*
       * Send a request to the message broker to be sent messages published with
       * the specified topic name. Wildcards are allowed.
       */
      private void subscribeToTopic(String topicName) {
            Log.d("ClientActivity", "C: inside subscribeToTopic");
            boolean subscribed = false;

            if (isAlreadyConnected() == false) {
                  // quick sanity check - don't try and subscribe if we
                  // don't have a connection

                  Log.e("mqtt", "Unable to subscribe as we are not connected");
            } else {
                  try {
                        String[] topics = { topicName };
                        mqttClient.subscribe(topics, qualitiesOfService);

                        subscribed = true;
                  } catch (MqttNotConnectedException e) {
                        Log.e("mqtt", "subscribe failed - MQTT not connected",
      e);
                  } catch (IllegalArgumentException e) {
                        Log.e("mqtt", "subscribe failed - illegal argument", e);
                  } catch (MqttException e) {
                        Log.e("mqtt", "subscribe failed - MQTT exception", e);
                  }
            }

            if (subscribed == false) {
                  //
                  // inform the app of the failure to subscribe so that the UI
      can
                  // display an error
                  broadcastServiceStatus("Unable to subscribe");

                  //
                  // inform the user (for times when the Activity UI isn't
      running)
                  notifyUser("Unable to subscribe", "MQTT", "Unable to
      subscribe");
            }
      }



/*
```

```
         * callback - called when we receive a message from the server
         */
        public void publishArrived(String topic, byte[] payloadbytes, int qos,
                    boolean retained) {
              Log.d("ClientActivity", "C: inside publish arrived");
              // we protect against the phone switching off while we're doing this
              // by requesting a wake lock - we request the minimum possible wake
              // lock - just enough to keep the CPU running until we've finished
              PowerManager pm = (PowerManager) getSystemService(POWER_SERVICE);
              WakeLock wl = pm.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK,
        "MQTT");
              wl.acquire();

              //
              // I'm assuming that all messages I receive are being sent as
        strings
              // this is not an MQTT thing - just me making as assumption about
        what
              // data I will be receiving - your app doesn't have to send/receive
              // strings - anything that can be sent as bytes is valid
              String messageBody = new String(payloadbytes);

              //
              // for times when the app's Activity UI is not running, the Service
              // will need to safely store the data that it receives
              if (addReceivedMessageToStore(topic, messageBody)) {
                    Log.d("ClientActivity", "C: inside addReceiveMessageToStore");
                    // this is a new message - a value we haven't seen before

                    //
                    // inform the app (for times when the Activity UI is running)
        of the
                    // received message so the app UI can be updated with the new
        data
                    broadcastReceivedMessage(topic, messageBody);

                    //
                    // inform the user (for times when the Activity UI isn't
        running)
                    // that there is new data available
                    notifyUser("New data received", topic, messageBody);
              }

              // receiving this message will have kept the connection alive for
        us, so
              // we take advantage of this to postpone the next scheduled ping
              scheduleNextPing();

              // we're finished - if the phone is switched off, it's okay for the
        CPU
              // to sleep now
              wl.release();
        }
```

Connection lost code

```
/*
         * callback - method called when we no longer have a connection to the
         * message broker server
         */
        public void connectionLost() throws Exception {
              Log.d("ClientActivity", "C: inside connectionLost");
              // we protect against the phone switching off while we're doing this
              // by requesting a wake lock - we request the minimum possible wake
              // lock - just enough to keep the CPU running until we've finished
              PowerManager pm = (PowerManager) getSystemService(POWER_SERVICE);
              WakeLock wl = pm.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK,
  "MQTT");
              wl.acquire();

              //
              // have we lost our data connection?
              //

              if (isOnline() == false) {
                    connectionStatus =
        MQTTConnectionStatus.NOTCONNECTED_WAITINGFORINTERNET;

                    // inform the app that we are not connected any more
                    broadcastServiceStatus("Connection lost - no network
        connection");

                    //
                    // inform the user (for times when the Activity UI isn't
        running)
                    // that we are no longer able to receive messages
                    notifyUser("Connection lost - no network connection", "MQTT",
                                "Connection lost - no network connection");

                    //
                    // wait until the phone has a network connection again, when
        we
                    // the network connection receiver will fire, and attempt
        another
                    // connection to the broker
              } else {
                    //
                    // we are still online
                    // the most likely reason for this connectionLost is that
        we've
                    // switched from wifi to cell, or vice versa
                    // so we try to reconnect immediately
                    //

                    connectionStatus =
        MQTTConnectionStatus.NOTCONNECTED_UNKNOWNREASON;

                    // inform the app that we are not connected any more, and are
                    // attempting to reconnect
                    broadcastServiceStatus("Connection lost - reconnecting...");

                    // try to reconnect
                    if (connectToBroker()) {
                          subscribeToTopic(topicName);
                    }
              }

              // we're finished - if the phone is switched off, it's okay for the
        CPU
              // to sleep now
              wl.release();     }
```