Creating an Ad component in LWUIT

This article explains how to create and integrate advertisements (Ad) from the InnerActive Ad network into LWUIT apps.

03 Nov 2013

Note: This is an entry in the Asha Touch Competition 2012Q3

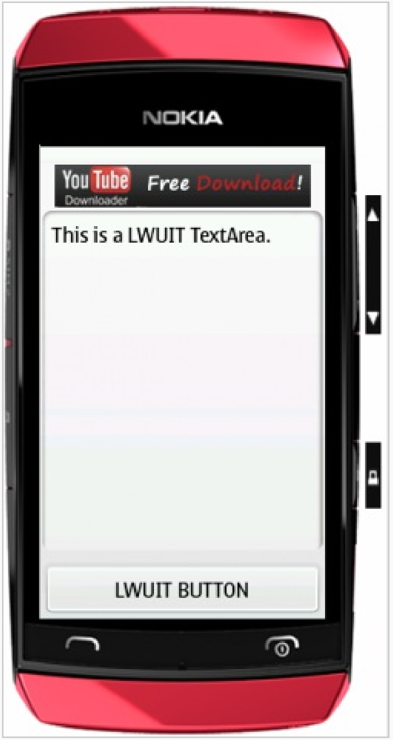
Introduction

In-app advertising is a monetization method used by many apps. Nokia recommends using InnerActive as the Ads provider in their platforms.

Unfortunately, while InnerActive provides a SDK for Java ME, its use requires the Ad (Banner or Interstitial) to have the full screen of the phone (i.e. you cannot embed the Banner Ad in your app UI). As a result, this method is not compatible with LWUIT UI platform.

In this article I show how to create and integrate an InnerActive Ad into an LWUIT-based app, by extending LWUIT and creating a new `AdBanner` component that uses the InnerActive Server API. Please note that in order to use this `AdBanner` component you need to register your app with InnerActive and receive an App ID.

You can see a sample of the end result in the screenshot below.



InnerActive Banner Ad in a LWUIT form application.

The Implementation

I chose to extend the `Label` class in order to be able to display both image and text-based Ads.

The classes for this example are divided as such:

- `AdBanner` - The class that extends Label and responsible for the Ad network interaction and parsing.
- `RMSParams` - This is actually my own utility class which basically help you save & load persistent key/value properties (it is needed because InnerActive generate a client id `cid` for the user that you need to save and send back so it wont get the same Ad if he already clicked on it)
- `Main` - a simple Main MIDLet which creates the form of this example and place the Ad on the form.

When dealing with the server API of InnerActive in its XML form which I chose to use there are 2 main things, one is the contsruction of the request URL and the other is the parsing of the XML returned from that request.

In the request URL there are many parameters some of them are mandatory such as `aid` - the app id, `v` - version of the protocol, `po`

- the used channel which for Nokia store banners is 551 and some are optional which may help increase your fill rate such as "hid" - imei number for nokia device use `System.getProperty("com.nokia.mid.imei")`, `l` - location of the user, etc.

Below is a snippet for the url code formatting.

```
String url = AD_URL + "?aid=" + adID + "&v=" + AD_VERSION + "&po=" + AD_PO;
url += "&w=" + Display.getInstance().getDisplayWidth() + "&h=" +
Display.getInstance().getDisplayWidth();
if (ua != null)
 url += "&ua=" + ua;
if (cid != null)
 url += "&cid=" + cid;
if (imei != null)
 url += "&hid=" + imei;
if (age != "")
 url += "&a=" + age;
if (gender != "")
 url += "&g=" + gender;
if (location != "")
 url += "&l=" + location;
```

You can see my full URL handling in the attached source.

For the returned response XML parsing I chose to use the `XMLParser` class in LWUIT (you can use other XML parsers if you wish but since it comes with the package I thought it would be simpler to use it). The main tags that are needed in the XML are `tns:Response`, `tns:Client`, `tns:Ad` (inside this tag reside the `tns:Text`, tns:Image, `tns:Url` for the banner). The response tag tell us if the response is OK or not, the `Client` tag give us the cid to save for future Ad request and the Ad tag is the banner.

Here is the main part of parsing the returned XML from the HTTP request.

```
con = (HttpConnection) Connector.open(url);
con.setRequestMethod(HttpConnection.GET);
if (con.getResponseCode() != HttpConnection.HTTP_OK)
 return;
XMLParser xml = new XMLParser();
Element node = xml.parse(new InputStreamReader(is = con.openInputStream(), "UTF-8"));
String responseError = node.getAttribute("error").trim().toLowerCase();
if (!responseError.equals("ok"))
 return;
for (int i = 0; i < node.getNumChildren(); i++)
{
 Element n = node.getChildAt(i);
 String tagName = n.getTagName().trim().toLowerCase();
 if (tagName.equals("tns:client"))
 {
  rms.put("cid", n.getAttribute("id").trim());
  rms.save();
 } else if (tagName.equals("tns:ad"))
 {
  for (int j = 0; j < n.getNumChildren(); j++)
  {
   Element adNode = n.getChildAt(j);
   String adTagName = adNode.getTagName().trim().toLowerCase();
   if (adTagName.equals("tns:text"))
   {
    if (adNode.getNumChildren() > 0)
     adText = adNode.getChildAt(0).getText().trim();
   }
   if (adTagName.equals("tns:image"))
   {
    if (adNode.getNumChildren() > 0)
     bannerURL = adNode.getChildAt(0).getText().trim();
   }
   if (adTagName.equals("tns:url"))
   {
    link = adNode.getChildAt(0).getText().trim();
   }
  }
 }
}
```

The `main` class has absolutely nothing special in it, it simply create the lwuit form and place the components in the form.

Summary

In conclusion you can use my ready classes AdBanner & RMSParams in the attached source to integrate InnerActive Ads in any LWUIT based application for any screen size and touch/no touch device. If your application has user information such as Age, Location, Gender etc you can call the proper method for settings those values in the AdBanner class.