Creation web application for Atrjeews

Introduction

Generally Atjeews supports any web application created using J2EE technologies and packaged as .war file. However considering limited resources provided by Android devices and also some APIs limitation, certain things should be taken in consideration as

- Using small foot print servlet or application frameworks
- Do not use features of JDK 6 and above
- Do not use certain heavy weight J2EE features as EJB, JNDI, and JDBC.

Getting started

Most of getting started guides limits a reader to creation Hello World! application which can be a good starting point but still insufficient for starting a real development. This guide will be a bit different, it will guide you to create not only Hello World application, but an application doing something useful and even using Android APIs

What do you need first

First you need to have the following software and equipment

1. Android phone or tablet or an emulator
2. Atjeews installed on an Android device or an emulator (http://tjws.sf.net)
3. PC running Windows, Linux or MacOS with Android development kit installed (http://developer.android.com/sdk/index.html)
4. JDK 1.5 or better installed (check with your OS availability, for Windows it can be http://java.oracle.com)
5. TJWS release 1.106 or better deployed (http://tjws.sf.net)
6. 7Bee building tools downloaded and installed (https://github.com/drogatkin/7Bee)
7. Any text editor or your choice or JDE as Eclipse (http://eclipse.org)

Hello World project

This little project allows you to read contacts from your phone and display them in a browser. Let' name the app as mycontacts. The fastest way is developing the application as a JSP page. Later more advanced technologies for creation a web application will be introduced including using WebBee framework.

Creation project directories structures

Although it isn't a big deal have all project files stored in same directory, it is good to learn structuring project content. So the following directory structure is encouraged to be created.

mycontacts

src

Jsp

conf

A valid web.xml has to be created to be able to deploy application under Atjeews, web.xml can be really minimalistic as below:

web.xml:

```
<web-app version="3.0" metadata-complete="true">
    <welcome-file-list>
        <welcome-file>mycontacts.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

Now mycontacts.jsp can be created. Generally this file may contain just “Hello World!” string, however the purpose of this example to make it doing something more useful, like getting access to Android contacts storage and display it as a web page. Context of Android device has to be obtained first to access contacts storage. Atjeews gives access to underneath Android context as application property ##RuntimeEnv. The rest of the application is really straight forward. So the following content of my mycontacts.jsp has to be placed under jsp directory.

mycontacts.jsp:

```
<%@ page language="java" import="android.content.ContentResolver,
android.database.Cursor,android.provider.ContactsContract, android.content.Context"
contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width, user-scalable=no" />
  <title>My Contacts First Atjeews app demo</title>
</head>
<body>
```

```
<h1>My Android contacts list</h1>
<table>
 <tr><th>Name</th><th>Phone</th><th>E-mail</th></tr>
<%
      Context context = (Context)application.getAttribute("##RuntimeEnv");
       ContentResolver cr = context.getContentResolver();
                  Cursor cur = cr.query(ContactsContract.Contacts.CONTENT_URI, null, null, null, null);
                  if (cur.getCount() > 0) {
                        while (cur.moveToNext()) {
                                    String id =
(cur.getString(cur.getColumnIndex(ContactsContract.Contacts._ID)));
                                    out.print("<tr><td>");
                                    out.print(cur.getString(cur
          .getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME)));
                                    out.print("</td><td>");
                                    if
(Integer.parseInt(cur.getString(cur.getColumnIndex(ContactsContract.Contacts.HAS_PHONE_NUMBER))) >
0) {
                                          Cursor pCur =
cr.query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI, null,
          ContactsContract.CommonDataKinds.Phone.CONTACT_ID + " = ?", new String[] { id }, null);
                                          while (pCur.moveToNext()) {
          out.print(pCur.getString(pCur.getColumnIndex(ContactsContract.CommonDataKinds.Phone.TYPE)));
                                                out.print(" ");
                                                out.print( pCur.getString(pCur
          .getColumnIndex(ContactsContract.CommonDataKinds.Phone.NUMBER)));
```

```
                out.print("<br/>");
                                                }
                                                pCur.close();
                                        }
                                        out.print("</td><td>");

                                        Cursor emailCur =
cr.query(ContactsContract.CommonDataKinds.Email.CONTENT_URI, null,
        ContactsContract.CommonDataKinds.Email.CONTACT_ID + " = ?", new String[] { id }, null);
                                        while (emailCur.moveToNext()) {
                                                // This would allow you get several email addresses
                                                // if the email addresses were stored in an array
                                                String email = emailCur.getString(emailCur
        .getColumnIndex(ContactsContract.CommonDataKinds.Email.DATA));
                                                String emailType = emailCur.getString(emailCur
        .getColumnIndex(ContactsContract.CommonDataKinds.Email.TYPE));
                                                out.print(email);
                                                out.print(" ");
                                                out.print(emailType);
                                                out.print("<br/>");
                                        }
                                        emailCur.close();
          out.print("</td></tr>");
                                }
                        } else {
```

```
                    out.print("<tr><td colspan=3>No contacts</td></tr>");
                }
                cur.close(); // finally
        %>
 </table>
</body>
</html>
```

Note that an application has to be granted a permission to access contacts storage. Since the application runs as part of Atjeews Android application, it inherits all permissions given to it. The access to contacts is requested in Atjeews. Actually Atjeews requests already most useful permissions, however if a new application needs some rare permission, then Atjeews manifest has to be changed to request the permission.

Next step will be a creation of a war file. The best way is using 7Bee tool for that. A simple 7Bee script can be created and placed in root of mycontacts project.

bee-web.xml:

```
<?xml version="1.0" encoding="utf-8"?>
 <!DOCTYPE bee PUBLIC "-//Dmitriy Rogatkin//DTD Bee Project Builder 1.0//EN"
   "http://knitknot.info/DTD/bee.dtd" [
 ]>

<bee name="mycontacts" type="project">
 <target name="check build" dir=".">
   <dependency>
     <expression>
       <operator name="eq">
         <function name ="timestamp">
           <parameter value="build" type="path"/>
         </function>
         <value></value>
```

```
      </operator>
     </expression>
   </dependency>
   <task exec="mkdir">
    <parameter value="build" type="path"/>
   </task>
</target>

<target name="war" dir=".">
  <dependency target="check build"/>
  <dependency>
    <expression>
      <operator name="not">
        <function name ="timestamp">
          <parameter value="build/mycontacts.war" type="path"/>
        </function>
      </operator>
    </expression>
  </dependency>
  <dependency>
    <expression>
      <function name="anynewer">
        <parameter value="src/jsp/*.jsp" type="path"/>
        <parameter value="build/mycontacts.war" type="path"/>
      </function>
    </expression>
  </dependency>
```

```
   <block>
    <echo value="...->build/mycontacts.war"/>
    <function name="warit">
      <parameter value="build/mycontacts.war" type="path"/>
      <parameter>src/conf/web.xml</parameter>
      <parameter>A</parameter>
      <parameter>src/jsp/mycontacts.jsp</parameter>
    </function>
   </block>
  </target>
</bee>
```

The following command will build the application war file:

```
bee –f bee-web.xml
```

Directory build will contain file mycontacts.war.

Now mycontacts.war has to be prepared to be deployed under Atjeews on a real device or emulator. Copy bee-dexwar.xml from Atjeews distribution in the root directory of the project. Open the script and review and edit the top lines:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE bee PUBLIC "-//Dmitriy Rogatkin//DTD Bee Project Builder 1.0//EN"
   "http://knitknot.info/DTD/bee.dtd" [
    <!ENTITY android_sdk "E:\tools\adt-bundle-windows-x86_64-20140321\sdk">
    <!ENTITY jasper_lib "E:\projects\jasper7\build\jasper.jar">
    <!ENTITY servlet_lib "E:\projects\servlet\javax.servlet.jar">
    <!ENTITY target "4.4.2">
]>
<!-- $Id: bee-dexwar.xml,v 1.9 2011/09/02 04:09:46 dmitriy Exp $ Prepare
```

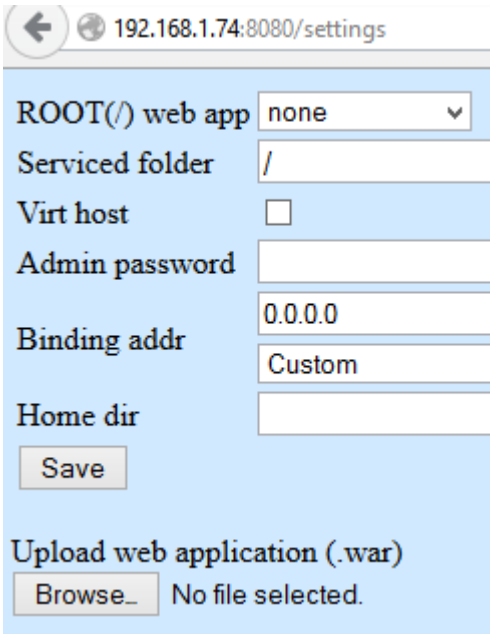
Providing correct location of Android JDK, Jasper and servlet reference implementation directory. Servlet.jar and jasper.jar are part of TJWS installation. It is important to use Atjeews version matching to TJWS version,

since Jasper JSP compiler has to match Jasper runtime as a part of Atjeews. Launch the script using command:

bee -- build\mycontacts.war

Android version of mycontacts.war will be created in the current directory.

Now the application can be deployed under Atjeews. If Atjeews is running on a real device, then the best way will be uploading mycontacts application directly from desktop computer. Make sure that Atjeews is running and then reach setting screen copying its URL in desktop browser. The screen has a button to upload .war, so use it to upload and deploy the application.



Deployment in an emulator is a bit trickier. You need to tell to the emulator to forward certain port from its space to a port of your machine. You can use telnet to do that. Launch telnet first:

```
telnet localhost 5554
```

And then in telnet screen execute forward command

```
redir add tcp:5000:8080
```

Now you can access Atjeews from desktop browser using URL http://localhost:5000m so use **settings** screen to deploy .war.

Now time to start improving the application and learn WebBee to bring more exciting applications to Android device.

Trouble shooting

Most common problems can be failing pre-compilation of JSP. One of the reason can be improper path to Jasper or android.jar.