You need to setup multiple components to be able to create an account programmatically. You need:

* an AccountAuthenticator
* a Service to provide access to the AccountAuthenticator
* some permissions

**The authenticator**

The authenticator is an object that will make the mapping between the account type and the autority (i.e. the linux-user) that have rights to manage it.

**Declaring an authenticator** is done in xml :

* create a file res/xml/authenticator.xml

with the following content :

<?xml version="1.0" encoding="utf-8"?>

<account-authenticator xmlns:android="http://schemas.android.com/apk/res/android"

android:accountType="com.company.demo.account.DEMOACCOUNT"

android:icon="@drawable/ic\_launcher"

android:smallIcon="@drawable/ic\_launcher"

android:label="@string/my\_custom\_account"/>

Note the accountType : it must be reused in code when you create the Account. The icons and label will be used by the "Settings" app to display the accounts of that type.

**Implementing the AccountAuthenticator**

You must extends AbstractAccountAuthenticator to do that. This will be use by third party app to access Account data.

The following sample don't allow any access to 3rd-party app and so the implementation of each method is trivial.

public class CustomAuthenticator extends AbstractAccountAuthenticator {

public CustomAuthenticator(Context context) {

super(context);

}

@Override

public Bundle addAccount(AccountAuthenticatorResponse accountAuthenticatorResponse, String s, String s2, String[] strings, Bundle bundle) throws NetworkErrorException {

return null; //To change body of implemented methods use File | Settings | File Templates.

}

@Override

public Bundle editProperties(AccountAuthenticatorResponse accountAuthenticatorResponse, String s) {

return null; //To change body of implemented methods use File | Settings | File Templates.

}

@Override

public Bundle confirmCredentials(AccountAuthenticatorResponse accountAuthenticatorResponse, Account account, Bundle bundle) throws NetworkErrorException {

return null; //To change body of implemented methods use File | Settings | File Templates.

}

@Override

public Bundle getAuthToken(AccountAuthenticatorResponse accountAuthenticatorResponse, Account account, String s, Bundle bundle) throws NetworkErrorException {

return null; //To change body of implemented methods use File | Settings | File Templates.

}

@Override

public String getAuthTokenLabel(String s) {

return null; //To change body of implemented methods use File | Settings | File Templates.

}

@Override

public Bundle updateCredentials(AccountAuthenticatorResponse accountAuthenticatorResponse, Account account, String s, Bundle bundle) throws NetworkErrorException {

return null; //To change body of implemented methods use File | Settings | File Templates.

}

@Override

public Bundle hasFeatures(AccountAuthenticatorResponse accountAuthenticatorResponse, Account account, String[] strings) throws NetworkErrorException {

return null; //To change body of implemented methods use File | Settings | File Templates.

}

}

**The Service exposing the Account Type**

**Create a Service to manipulate the Accounts of that type :**

public class AuthenticatorService extends Service {

@Override

public IBinder onBind(Intent intent) {

CustomAuthenticator authenticator = new CustomAuthenticator(this);

return authenticator.getIBinder();

}

}

**Declare the service in your manifest** :

<service android:name="com.company.demo.account.AuthenticatorService" android:exported="false">

<intent-filter>

<action android:name="android.accounts.AccountAuthenticator"/>

</intent-filter>

<meta-data

android:name="android.accounts.AccountAuthenticator"

android:resource="@xml/authenticator"/>

</service>

Here, the filter and the meta-data referring to the xml resource declaring the authenticator are the key points.

**The permissions**

In your manifest be sure to declare the following permissions

<uses-permission android:name="android.permission.AUTHENTICATE\_ACCOUNTS"/>

<uses-permission android:name="android.permission.GET\_ACCOUNTS"/>

<uses-permission android:name="android.permission.MANAGE\_ACCOUNTS"/>

(not all required for the sample code presented in this post, but you will probably have some more code about account management and at the end all of them will be useful)

**Create an account in code**

Now that everything is ready you create an account with the following code. Note the boolean returned by addAccountExplicitly informing you about the success or failure.

AccountManager accountManager = AccountManager.get(this); //this is Activity

Account account = new Account("MyAccount","com.company.demo.account.DEMOACCOUNT");

boolean success = accountManager.addAccountExplicitly(account,"password",null);

if(success){

Log.d(TAG,"Account created");

}else{

Log.d(TAG,"Account creation failed. Look at previous logs to investigate");

}

**Final tips**

**Don't install your app on external storage**

If your app is installed on external storage, there are good chance that Android delete your Account data when sdcard is unmounted (since the authenticator for that account will not be accessible anymore). So to avoid this loss (on every reboot !!!) you must install the App declaring the authenticator on internal storage only :

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

android:installLocation="internalOnly"

...

**In case of trouble**

Read the logs carefully, The AccountManger is outputing many logs to help you to debug your code.