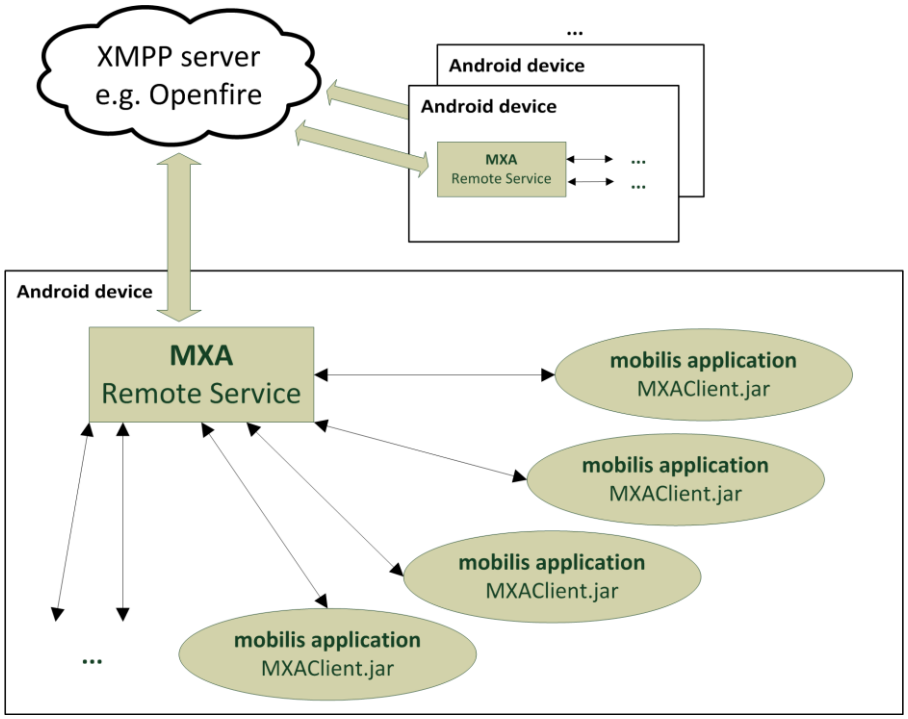
How to use MXA in own applications

General

MXA (Mobilis XMPP for Android) was developed to facilitate the use of xmpp communication on Android devices. So it allows a faster development of Android applications for the mobilis platform. On every Android device multiple applications can be installed, which all use the central ***MXA Remote Service*** of the device. Each mobilis application uses the library MXAClient.jar to connect to the *MXA Remote Service* (see Picture 1).



Picture 1: Overview of MXA components

Start the XMPP Preferences Activity

One of the first things to do when developing a mobilis application is to make the XMPP Preferences of the MXA available to the user. Therefor you only have to let one of your activities start a new activity with the intent seen in the following code snippet.

```
Intent i = new Intent(ConstMXA.INTENT_PREFERENCES);
this.startActivity(Intent.createChooser(i, "MXA not found. Please install."));
```

The *Intent Chooser* is used because your application would crash if MXA was not installed on the Android device. So, if there is no application reacting to the intent, at least an error message is displayed.

Connection to MXA Remote Service

From the *MXAClient* library you have to use ***MXAController*** (Singleton). In a user interface class the method ***connectMXA()*** from *MXAController* can be invoked. You have to pass the current context and the listener, which should be notified, if the connection between mobilis application and *MXA Remote Service* was established or broken up.

```
mMXAController = MXAController.get();
mMXAController.connectMXA(ctx, mMXAListener);
```

This listener must have the type ***MXAListener***. The methods *onMXAConnected()* and *onMXADisconnected()* must be implemented. (In the following code snippet a local method is called, which show a Toast message on the screen.)

```
mMXAListener = new MXAListener() {
    @Override
    public void onMXADisconnected() {
        makeToast("Connection to MXA lost.");
    }
    @Override
    public void onMXAConnected() {
        makeToast("Connection to MXA established.");
    }
};
```

Connection to XMPP server

If the connection between mobilis application and MXA Remote Service was properly established, the next step is connecting to the xmpp server. For that purpose the *XMPPService* is fetched from *MXAController* with the method ***getXMPPService()***. After that follows the call of the ***connect()*** method of the *XMPPService*.

```
@Override
public void onMXAConnected() {
    Log.i(TAG,"Connection to MXA Remote Service established.");
    iXMPPService = mMXAController.getXMPPService();
    try {
        /** Connect the MXA Remote Service to the XMPP Server */
        iXMPPService.connect(mConnectMessenger);
    } catch (RemoteException e) {
        Log.e(TAG, "MXA Remote Service couldn't connect to XMPP Server");
    }
}
```

On this XMPPService object you can also call other important methods:

- *connect, disconnect, sendMessage, sendIQ, sendPresence*
    - All these methods require a messenger with a handler included. The handler will be notified about the outcome of the method. This approach can guarantee a high level of asynchrony.
- *isConnected, getUsername*
    - These methods do not require such a messenger, because they are not called asynchronous, but they return the result immediately.

Sending IQ's

How to send XMPP-IQ's (Info Query) is demonstrated in the following code snippet. First you have to create the IQ of the type ***XMPPIQ*** and attach all the necessary parameters: ***sender***, ***receiver***, ***IQ type*** (get, set or result), ***IQ element name***, ***namespace*** and the ***payload*** in xml format.
To finally send the IQ, call the method ***sendIQ()*** of the *XMPPService.* It requires an ***acknowledgement messenger*** (*which is notified upon the delivery of the IQ*), a ***result messenger*** (*which is notified upon the arrival of the correspondent result IQ*), a ***request code*** (*which …*) and the ***IQ*** itself.

```
    private void sendTestIQ() {
        XMPPIQ iq = new XMPPIQ();
        try {
            iq.from = "alice@server-a.com/PC";
            iq.to = "bob@server-b.com/Phone";
            iq.type = XMPPIQ.TYPE_GET;
            iq.element = "query";
            iq.namespace = "mobilis:iq:station";
            iq.payload = "<Station id =\"hbf\"><name>Hauptbahnhof</name><Location><longitude>" +
                    "12345678</longitude><latitude>87654321</latitude></Location></Station>";
            //The IQ should now look like that in XML format:

//          <iq id="mobilis_1" to="bob@server-b.com/Phone" from="alice@server-a.com/PC" type="get">
//              <query xmlns="mobilis:iq:station">
//                  <Station id ="hbf">
//                      <name>Hauptbahnhof</name>
//                      <Location>
//                          <longitude>12345678</longitude>
//                          <latitude>87654321</latitude>
//                      </Location>
//                  </Station>
//              </query>
//          </iq>

            xmppService.sendIQ(new Messenger(ackHandler), new Messenger(testIQResultHandler), 1, iq);
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }
```

Further details:

- MXA cannot guarantee the delivery of XMPP packets. Die *MXAClients* have to care about that issue themselves. The delivery cannot be seen as successful until the acknowledgement reaches the correspondent handler.
- The handler (*acknowledgement* and *result handler*) can also receive *Negative Acknowledgements (NACK)*.

Receiving IQ's

To receive and process IQ's that where sent by other communication partners you have to call the method ***registerIQCallback()*** from *XMPPService*. You have to pass the ***callback*** which should be notified upon arrival of an IQ, the IQ's ***element name*** and the IQ's ***namespace***.

```
        try {
            xmppService.registerIQCallback(testIQCallback, "query", "mobilis:iq:station");
        } catch (RemoteException e) {
            e.printStackTrace();
        }
```

The Callback has to be of the type ***IXMPPIQCallback***. You have to override the method processIQ() to finally get and work with the IQ.

```
    public IXMPPIQCallback testIQCallback = new IXMPPIQCallback.Stub(){
        @Override
        public void processIQ(XMPPIQ xiq) throws RemoteException {
            Log.i(TAG, "SENDER: " + xiq.from);
            Log.i(TAG, "PAYLOAD: " + xiq.payload);
            // Process the IQ
            // ...
        }
    };
```