ZEBRA MDM Toolkit 1.0.1 - OBSOLETE

MDM Toolkit 1.0.1

NOTE: This document is obsolete. It is published as a courtesy for organizations using the processes for legacy devices.

For modern devices, please refer to:
http://techdocs.zebra.com/emmtk

Table of Contents

Overview

The MDM Toolkit provides a walk-through and sample application for the common tasks and components required for an MDM client to interface with the Zebra MX Management System (MXMS) for staging Zebra Android devices. This toolkit does not describe the means to generate XML required to exchange data with the MXMS. **Zebra does not recommend generating or editing XML code by hand.** All XML should be generated only by using the Zebra StageNow tool and its **Export a Profile to an MDM** feature. Once generated, XML is passed into the MDM client via some transport mechanism and submitted to the MXMS for processing. An XML response is passed from MXMS to the client with the processing results (success or failure) with information about errors (if any) in XML syntax or the requested operation.

MXMS XML Submission Guide

This guide covers:

- **Binding to the MXMS** - All communications to the MXMS on Zebra devices, occur through a common binding interface.

- **Submitting XML** - Within the MDM client, XML will be submitted to apply settings via a simple API.

Generating and transporting XML to and from the MDM client are **NOT** within the scope in this guide.

Requirements

- **MDM Toolkit**
- **Zebra Android Device with MX**

  Note: For instructions to install or update MX on a Zebra Android Device, please see the StageNow Getting Started Documentation.

- **Valid MXMS XML** - XML should be verified to work with your device through StageNow prior use with this toolkit. Features not supported by StageNow are likewise unsupported by the MDM ToolKit.

Binding to the MXMS

1. Create a new Android project with an empty activity.

2. Add the following permissions to the application's manifest file to allow it to access MXMS:

```
<uses-permission
android:name="com.symbol.mxmf.ACCESS_MX_MANAGEMENT_FRAMEWORK_SERVICE" />
<uses-permission
android:name="com.motorolasolutions.emdk.mxframework.ACCESS_MX_MANAGEMENT
_FRAMEWORK_SERVICE"/>
```

3. Create two new packages in the application:

   1. **"com.symbol.mxmf"**

   2. "**com.motorolasolutions.emdk.mxframework.**"

These packages will be used for holding the aidl (Android Interface Definition Language) files. **Note:** When Using Android Studio, create the new packages in the **aidl** folder of the project or the IDE will not automatically generate the necessary interface files.

4. The SimpleMdmToolKitSample project, which is supplied in the MDM Toolkit, contains the IMxFrameworkService.aidl files that should be copied into the application. These AIDL files are located in the SimpleMdmToolKitSample's **com.symbol.mxmf** and

com.motorolasolutions.emdk.mxframework packages. Copy these AIDL files into the respective packages made in Step 3.

5. In the package that contains the empty activity that was created in Step 1, copy and paste the following classes from the SimpleMdmToolKitSample project's **com.symbol.simplemdmtoolkitsample** package:

   – MxNamespace.java
   – MxNamespaceMotorolaSolutions.java
   – MxNamespaceSymbol.java
   – SymbolBrand.java

   Note: The package names in these classes must be edited to match the package name of your application.

6. In the application's activity, the **onCreate** method should call the **init** method that is in the **SymbolBrand** class. This will check that MXMS is installed on the device. If MXMS is detected, the application will be bound to MXMS, allowing interaction to occur between them.

```
// Variable to hold the main activity
static private Activity m_activity;

@Override
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    // Save the main activity so it can be accessed from other threads
    m_activity = this;

    // Call init ONCE to start the process of binding to the MXMS
    // Note: You should call init early (e.g. as shown here in onCreate)
    //       XML cannot be submitted to the MXMS the binding is complete
    SymbolBrand.MXMS.init(this,mMxFrameworkServiceConnection);
}
```

7. The ServiceConnection must be added to the application's activity. This contains the **onServiceConnected** callback that will notify the application that it was successfully bound to MXMS. It also must contain the **onServiceDisconnected** callback that will notify the application when it was successfully unbound with MXMS.

```
// This definition is mandatory to track binding to the MXMS
private ServiceConnection mMxFrameworkServiceConnection = new
ServiceConnection()
{
    // Callback to notify when binding to the MXMS has completed
    public void onServiceConnected(ComponentName className,IBinder
service)
    {

    }

    // Callback to notify when unbinding from the MXMS has occurred
    public void onServiceDisconnected(ComponentName className)
    {

    }
};
```

8. In the **`onServiceConnected`** method, the **`onServiceConnected`** in the SymbolBrand helper class also should be called. This will notify this class that the application was bound to MXMS. At the end of the method, the term method in the **SymbolBrand** class should be called, which will unbind the application from MXMS. The application can also exit at this point. In the **`onServiceDisconnected`** method, the **`onServiceDisconnected`** method in the **SymbolBrand** helper class should also be called, which will notify this class that your application was unbound from MXMS.

```
// This definition is mandatory to track binding to the MXMS
private ServiceConnection mMxFrameworkServiceConnection = new
ServiceConnection()
{
    // Callback to notify when binding to the MXMS has completed
    public void onServiceConnected(ComponentName className,IBinder
service)
    {
        // Pass the binding notification on so the helper class can know
the service was bound
        SymbolBrand.MXMS.onServiceConnected(className,service);

        // Call term
        SymbolBrand.MXMS.term(m_activity,mMxFrameworkServiceConnection);

        // Exit the application
        m_activity.finish();
    }
```

```
        // Callback to notify when unbinding from the MXMS has occurred
        public void onServiceDisconnected(ComponentName className)
        {
            // Pass the unbinding notification on so the helper class can
    know the service was unbound
            SymbolBrand.MXMS.onServiceDisconnected(className);
        }
    };
```

Submitting XML

In the application activity's **`onServiceConnected`** method, add code before the `term` call that would submit XML to MXMS. This example submits XML generated by StageNow directly after calling **`onServiceConnected`**. It also would be possible to submit XML from other locations as long as `term` was not called in **onServiceConnected**. The **`onServiceConnected`** method is the first location where XML submission can occur.

1. To submit XML, first use the **`isReady`** method from the **SymbolBrand** class to check if the application is successfully bound to MXMS, which would mean that it is ready to accept XML from the application.

2. Next, a String containing XML generated by StagNow would be striped of all formatting by using a helper method provided by the XmlParser class. The `xml` variable, in byte array form, is converted to a ByteArrayInputStream and passed to the **XmlParser.`readXml()`** helper method.

3. The String output from **readXml()** in Step 2 is used by the **SymbolBrand** class's **`submitXml`** method, which submits the XML to the MXMS.

4. MXMS will return a Result XML string which should be passed back to a server for processing.

```
    // This definition is mandatory to track binding to the MXMS
    private ServiceConnection mMxFrameworkServiceConnection = new
    ServiceConnection()
    {
        // Callback to notify when binding to the MXMS has completed
        public void onServiceConnected(ComponentName className,IBinder
    service)
        {
            // Pass the binding notification on so the helper class can know
    the service was bound
            SymbolBrand.MXMS.onServiceConnected(className,service);
            // For the purposes of this test application, we call this from
    here because this is the FIRST time it can be done
            // In a real world situation, it might be called repeatedly from
    other locations, as things that need to be done are identified

            if ( SymbolBrand.MXMS.isReady() ){

                //  XML generated by StageNow would be passed in to
    initialize the xml variable. Below is a example of what that xml might
    look like.
                String xml = "<wap-provisioningdoc>
                            <characteristic type="Clock" version="4.2" >
                                <parm name="AutoTime" value="false"/>
                                <parm name="TimeZone" value="GMT-5"/>
                                <parm name="Date" value="2014-12-03"/>
                                <parm name="Time" value="11:00:00"/>
                            </characteristic>
                        </wap-provisioningdoc>";

                // Remove all formatting from xml
                String inXml = XmlParser.readXml(new
    ByteArrayInputStream(xml.getBytes()));


                // Submit the XML to the MXMS for processing
                // Note: This will FAIL unless SymbolBrand.MXMS.isReady(),
    indicating that the binding to the MXMS has completed successfully

                String outXml = SymbolBrand.MXMS.submitXml(inXml);

                //outXml would then be passed back to the server for
    processing


            }
            // Call term
            SymbolBrand.MXMS.term(m_activity,mMxFrameworkServiceConnection);

            // Exit the application
            m_activity.finish();
        }
```

```
        // Callback to notify when unbinding from the MXMS has occurred
        public void onServiceDisconnected(ComponentName className)
        {
            // Pass the unbinding notification on so the helper class can
    know the service was unbound
            SymbolBrand.MXMS.onServiceDisconnected(className);
        }
    };
```

Sample Overview

The included sample provides Helper libraries and Service Interface definition files used in the guide above, and includes many examples of generated XML (located in the project's Assets folder) used to configure a Zebra device using the MXMS framework.

Sample Usage

The attached sample is an Eclipse/ADT project that can be opened directly with Eclipse/ADT or imported into Android Studio.

To use this sample:

1. Open/Import the sample in the prefered IDE.
2. Attach a Zebra Android device via USB and enable its USB debugging feature in Settings > Developer Options.
3. Run the sample as is or choose another XML file to submit by uncommenting/commenting out the prefered **xmlName** global variable definition.

   Note: The sample will likely run and close too fast to see a User interface, so check **logcat** for the application's output.