COGNEX

MX-1000

SDK Reference Manual

3/22/2016
Version 5.5.3

Legal Notices

The software described in this document is furnished under license, and may be used or copied only in accordance with the terms of such license and with the inclusion of the copyright notice shown on this page. Neither the software, this document, nor any copies thereof may be provided to, or otherwise made available to, anyone other than the licensee. Title to, and ownership of, this software remains with Cognex Corporation or its licensor. Cognex Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by Cognex Corporation. Cognex Corporation makes no warranties, either express or implied, regarding the described software, its merchantability, non-infringement or its fitness for any particular purpose.

The information in this document is subject to change without notice and should not be construed as a commitment by Cognex Corporation. Cognex Corporation is not responsible for any errors that may be present in either this document or the associated software.

Companies, names, and data used in examples herein are fictitious unless otherwise noted. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, nor transferred to any other media or language without the written permission of Cognex Corporation.

Portions of the hardware and software provided by Cognex may be covered by one or more U.S. and foreign patents, as well as pending U.S. and foreign patents listed on the Cognex web site at: http://www.cognex.com/patents.

The following are registered trademarks of Cognex Corporation:

Cognex, 2DMAX, Advantage, Alignplus, Assemblyplus, Check it with Checker, Checker, Cognex Vision for Industry, Cognex VSOC, CVL, DataMan, DisplayInspect, DVT, EasyBuilder, Hotbars, IDMax, In-Sight, Laser Killer, MVS-8000, OmniView, PatFind, PatFlex, PatInspect, PatMax, PatQuick, SensorView, SmartView, SmartAdvisor, SmartLearn, UltraLight, Vision Solutions, VisionPro, VisionView

The following are trademarks of Cognex Corporation:

The Cognex logo, 1DMax, 3D-Locate, 3DMax, BGAII, CheckPoint, Cognex VSoC, CVC-1000, FFD, iLearn, In-Sight (design insignia with cross-hairs), In-Sight 2000, InspectEdge, Inspection Designer, MVS, NotchMax, OCRMax, ProofRead, SmartSync, ProfilePlus, SmartDisplay, SmartSystem, SMD4, VisiFlex, Xpand

Other product and company trademarks identified herein are the trademarks of their respective owners.

Table of Contents

Symbols

The following symbols indicate safety precautions and supplemental information.

WARNING: This symbol indicates the presence of a hazard that could result in death, serious personal injury or electrical shock.

CAUTION: This symbol indicates the presence of a hazard that could result in property damage.

Note: Notes provide supplemental information about a subject.

Tip: Tips provide helpful suggestions and shortcuts that may not otherwise be apparent.

Using the DataMan Mobile SDK

The following example demonstrates the use of a sample application as part of the DataMan Mobile SDK for Android. The DataMan Mobile SDK is used to develop your own mobile application for your DataMan device. The example included in this document shows the basic use of the DataMan SDK.

Tip: The DataMan Mobile SDK available at http://cognex.com/tech_supp/dataman/Android_DataMan_SDK_v1.1.0_rc2.zip.

The application can do the following:

- Connect to a device with a manually entered IP address.
- Discover DataMan devices on the network and connect to them.
- Trigger the connected DataMan device.
- Receive and display images or read strings from the device and display them.
- Enable and disable live image mode on the device and display the image stream arriving from the device.

Note: The following sample application is part of the DataMan SDK for Android and can be deployed to any Android phone that runs Android OS v4.0.3.

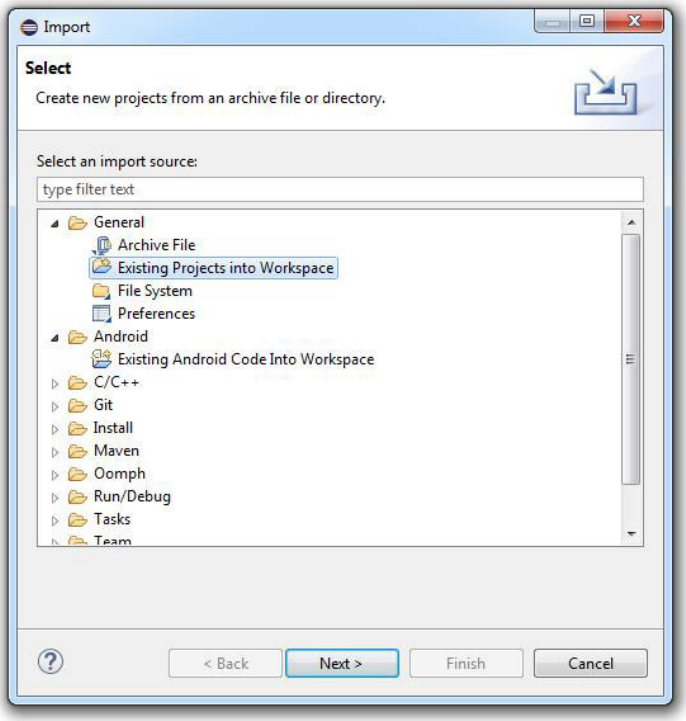
The sample applicaion, ***DataManSampleApp***, can be compiled and deployed from Eclipse 4.5.0 or from Android Studio v1.0. In our example, Eclipse is used to show how to open and deploy the application on an Android phone.

The following requirements must be met to use the application:

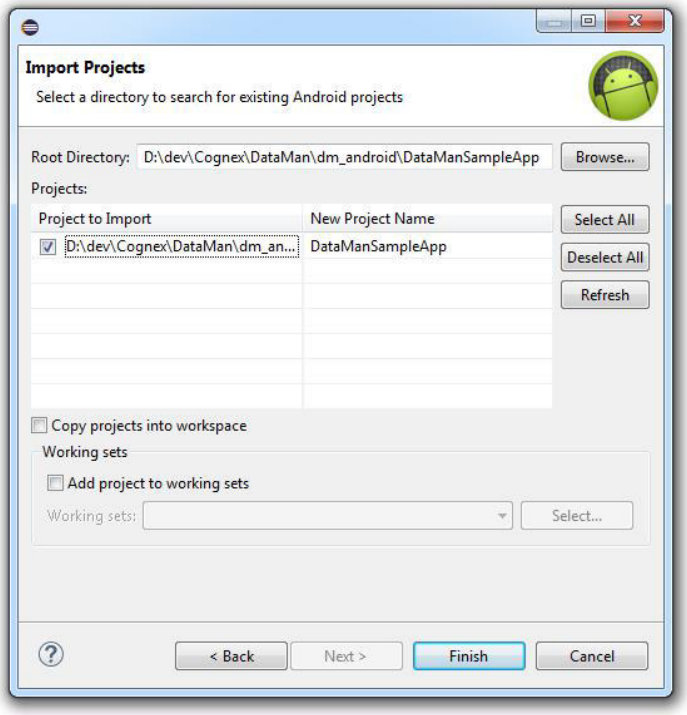
- **Operating system**: Windows 7 (or later), or Mac OS X v10.10 (or later)
- **Development environment**: Eclipse 4.5.0 installed and configured
- **Mobile device**: Samsung Galaxy S4
- **DataMan device**: Cognex MX-1000 Vision-Enabled Mobile Terminal (henceforth referred to as "MX-1000")
- **Software Development Kit**: DataMan Mobile SDK for Android

Follow the steps below to deploy the application.

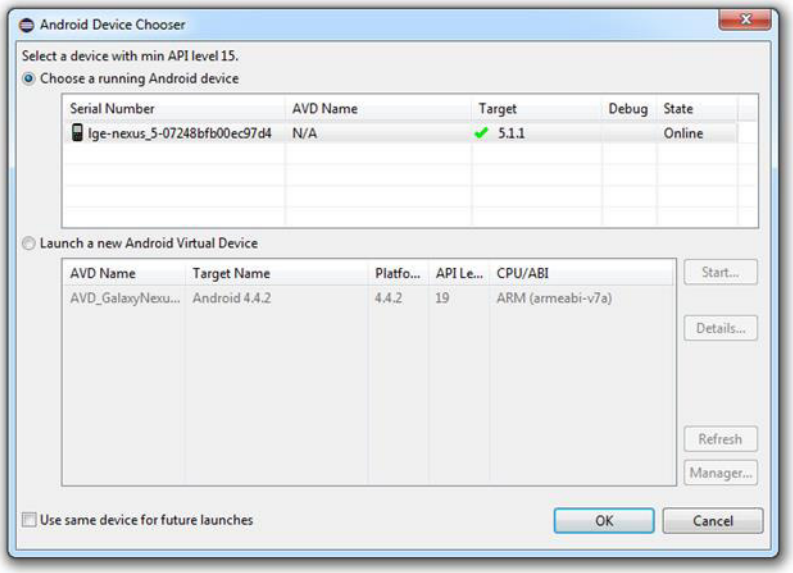
1. To open the sample project, first start Eclipse on your computer.
2. Select **File** -> **Import**, then select **General** -> **Existing Projects into Workspace** to import the existing project from the SDK folder.



3. Browse to the sample application folder and click **Finish**.



4. Once the project is loaded, select **Run -> Run**, select the device on which you want to run the application, and click **OK**.



5. After the application is launched on the mobile device, connect the MX-1000 to it by tapping the **Connect** button.

6. If the device is connected, you will see messages printed on the lower half of the screen.
   - Tap the **Trigger** button to trigger the reader.
   - Tap **Live On** to enable live image mode and tap it once again to disable live image mode.

DataManSample Walkthrough

This section explains the most important steps of the ***DataManSample*** application.

1. Enumerating USB Device.
   You can connect to the MX-1000 engine via USB. For this, you will need to use the **`UsbDiscoverer`** that is provided by the DataMan SDK. The code below demonstrates how to use this class:

```
HashMap<String, UsbDevice> usbDevices = UsbDiscoverer
        .getDataManDevices(this);

addLog("USB Device count:" + String.valueOf(usbDevices.size()));

if  (usbDevices.size() > 0) {
   UsbDevice usbDevice = usbDevices.entrySet().iterator().next().getValue();
   UsbManager usbManager = (UsbManager) getSystemService(Context.USB_SERVICE);
   usbManager.requestPermission(usbDevice, PendingIntent.getBroadcast(this, 0,
new Intent(ACTION_USB_PERMISSION), 0));
}
```

> **Note**: The value of *`ACTION_USB_PERMISSION`* is defined in the application and can be anything. We recommend that you use a the following format and replace *companyname* with your company's name: *com.companyname.appname.USB_PERMISSION* .

In the code above, the UsbDiscoverer is asked to provide the list of DataMan currently connected to the phone. If there is a DataMan usb device connected then the app requests the usb permission from the system. In order to

process the granting of the permission we need to declare the following receivers:

```
private BroadcastReceiver usbDevicePermissionReceiver = new BroadcastReceiver() {

    @Override
   public void onReceive(Context context, Intent intent) {
     UsbDevice device =
(UsbDevice)intent.getParcelableExtra(UsbManager.EXTRA_DEVICE);
      if (intent.getBooleanExtra(UsbManager.EXTRA_PERMISSION_GRANTED, false))
{

   connect(DataManSystem.createDataManSystemOverUsb(MainActivity.this, device));
      } else {
        addLog("Permission not granted by the user.");
      }
   }
};
```

To receive notification when an MX-1000 engine is detached from the phone, a receiver will be needed:

```
private BroadcastReceiver usbDeviceDetachedReceiver = new BroadcastReceiver()
{

    @Override
   public void onReceive(Context context, Intent intent) {
 if (dataManSystem != null && dataManSystem.isConnected()) {
           dataManSystem.disconnect();
      }
   }
};
```

2. Before connecting to an MX-1000 engine, perform the following:

   i. Set the result types

   The engine can send different type of information back to the application. It is possible to tell the engine what kind of information our application is interested in by setting the result types:

```
 dataManSystem.setResultTypes(EnumSet.of(
ResultType.READ_STRING,
ResultType.READ_XML,
ResultType.IMAGE,
ResultType.IMAGE_GRAPHICS));
```

   Here, we are telling the engine to send back the following types of information:

   - READ_STRING: the string from the barcode
   - READ_XML: an XML that contains more information about what was actually read
   - IMAGE: a jpg image sent by the engine
   - IMAGE_GRAPHICS: an overlay in SVG format showing a highlighted rectangle of the read code.

   ii. Set up listeners

   There are number of events that the app can receive notifications from the engine. The app can declare listeners for these events and it needs to set up these listeners before connecting to the engine:

```
dataManSystem.setOnConnectedListener(this);
dataManSystem.setOnDisconnectedListener(this);
dataManSystem.setOnReadStringArrivedListener(this);
dataManSystem.setOnImageArrivedListener(this);
dataManSystem.setOnImageGraphicsArrivedListener(this);
dataManSystem.setOnConnectionErrorListener(this);
dataManSystem.setOnHeartbeatResponseMissedListener(this);
```

   Tip: For further information regarding the listeners, please consult the DataManSDK documentation or review the SampleApplication source code in the SDK.

3. The application can now connect to the MX-1000 engine by calling the connect method:

```
dataManSystem.connect();
```

Note: After calling *connect()*, the *OnConnectedListener* method will be invoked if the connection was successful.

4. When a barcode is read, the engine will send you the read string. The following code snippet demonstrates how to receive the read string:

```
 @Override
 public void onReadStringArrived(DataManSystem dataManSystem, int resultId,
      String readString) {
   this.addLog("String read: " + readString);
}
```

The image also sends an image when the trigger is finished. The following is a snippet showing how to receive the image sent by the engine:

```
 @Override
 public void onImageArrived(DataManSystem dataManSystem, int resultId,
      Bitmap image) {
   this.addLog("Image arrived from the device.");

   scanImage.setImageBitmap(image);
}
```

If the heartbeat functionality is enabled before connecting, the SDK will send a heartbeat to the engine from time to time to find out if the engine still responds. If there is no answer received from the engine, the SDK provides a way for the application to be notified. One such event can be the engine restart after a firmware update. The following snippet shows how to detect this event and how to react to it:

```
 @Override
public void onHeartbeatResponseMissed(DataManSystem dms) {
   this.addLog("Heartbeat missed.");

   dataManSystem.disconnect();

 connectButton.setEnabled(false);
}
```

5. When the application is connected to the engine, it is required to report the phone's battery status regularly. The following code snippet demonstrates how to send the battery status to the engine. With this information, the engine can decide if the phone needs to be charged or not.

```
IntentFilter intentFilter = new IntentFilter(Intent.ACTION_BATTERY_CHANGED);
Intent batteryStatus = registerReceiver(null, intentFilter);
int level = batteryStatus.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
int scale = batteryStatus.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
 final int batteryPct = (int)(level / (float)scale * 100);
 dms.sendCommand("SET MOBILE-DEVICE.BATTERY-STATE " + batteryPct, new
OnResponseReceivedListener() {

    @Override
 public void onResponseReceived(DataManSystem dms, DmccResponse resp) {
 if (resp.getError() == null) {
          Log.d("BatteryReporterService", "Sent battery percent " + batteryPct);
 } else {

         Log.w("BatteryReporterService", "Error Sending battery percent " +
batteryPct, resp.getError());
      }

      dms.disconnect();
   }
});
```

The DMCC command to use is **SET MOBILE-DEVICE.BATTERY-STATE**. For example, sending **SET MOBILE-DEVICE.BATTERY-STATE 95** to the engine reports that currently, the phone battery is at 95 percent.

Useful Code Snippets

DMCC Commands

This section demonstrates how to send information to the engine or get information from it.

Tip: The full documentation of these commands can be found in the ***DataMan Control Commands*** document.

1. Sending a DMCC to get the device type:
   In the following snippet, the **GET DEVICE.TYPE** command will be sent to the engine and the response will be logged:

```
dataManSystem.sendCommand("GET DEVICE.TYPE", new OnResponseReceivedListener() {

       @Override
       public void onResponseReceived(DataManSystem dataManSystem, DmccResponse
response) {
              addLog("Device type: " + response.getPayLoad());
      }
});
```

2. Query and Enable/Disable symbology:
   The following snippet demonstrates how to query the current status of DataMatrix symbology on the connected engine:

```
dataManSystem.sendCommand("GET SYMBOL.DATAMATRIX", new OnResponseReceivedListener() {
       @Override
       public void onResponseReceived(DataManSystem dataManSystem, DmccResponse
response) {
   addLog("DataMatrix symbol is: " + response.getPayLoad());

   // Logic comes here
      }
});
```

   The following snippet enables DataMatrix on the connected MX-1000 engine:

```
dataManSystem.sendCommand("SET SYMBOL.DATAMATRIX ON", new
OnResponseReceivedListener()
{
       @Override
       public void
onResponseReceived(DataManSystem dataManSystem, DmccResponse response) {
 if (response.getError() != null) {
            // Error

      }

      // Logic comes here
   }
});
```

3. Turning off the aimer:
   The command you can use to turn on the aimer light on the connected engine is **LIGHT.AIMER**.

```
dataManSystem.sendCommand("SET LIGHT.AIMER 1", new OnResponseReceivedListener()
{
    @Override
    public void onResponseReceived(DataManSystem dataManSystem, DmccResponse response)
{
   if (response.getError() != null) {
            // Error

      }

      // Logic comes here
   }
});
```

4. Saving the changes to the Engine:

   In order to persist the configuration changes that our application made on the connected engine we need to issue a command called CONFIG.SAVE.

```
dataManSystem.sendCommand("CONFIG.SAVE", new OnResponseReceivedListener()
{
    @Override
    public void onResponseReceived(DataManSystem dataManSystem, DmccResponse response)
{
 if (response.getError() != null) {
            // Error

      }

      // Logic comes here
   }
});
```

Precautions

Observe these precautions when installing the Cognex product, to reduce the risk of injury or equipment damage:

- To reduce the risk of damage or malfunction due to over-voltage, line noise, electrostatic discharge (ESD), power surges, or other irregularities in the power supply, route all cables and wires away from high-voltage power sources.
- Changes or modifications not expressly approved by the party responsible for regulatory compliance could void the user's authority to operate the equipment.
- Cable shielding can be degraded or cables can be damaged or wear out more quickly if a service loop or bend radius is tighter than 10X the cable diameter. The bend radius must be at least six inches from the connector.
- Class A Equipment (broadcasting and communication equipment for office work): Seller and user shall be notified that this equipment is suitable for electromagnetic equipment for office work (Class A) and can be used outside the home.
- This device should be used in accordance with the instructions in this manual.
- All specifications are for reference purpose only and may be changed without notice.