Spring Design

ScreenShare Service SDK

Instructions

ScreenShare Service

V1.0.8

Change logs

Date	Version	Changes
2013/2/28	1.0.0	First draft
2013/3/5	1.0.1	Redefined some interfaces according to issues raised by Richard Li
2013/3/8	1.0.2	Complete this document
2013/3/13	1.0.3	Revised the part on Google Analytics
2013/4/1	1.0.4	Added sendDataWithPriority interface
2013/4/9	1.0.5	Revised ServiceConfig function name
2013/5/19	1.0.6	Added cancelSendFile interface description
2013/6/3	1.0.7	Revised AppConnection control description
2013/7/12	1.0.8	Added StreamingAPI interface description

Note: This SDK is for Android platform only.

ScreenShareServiceSDKInstructions_v1.0.8_10-17-13

Table of Contents

Abstract

ScreenShareService SDK on Android platform provides an easy to use ScreenShare API call service (i.e. call ScreenShareServiceProxy class function) for third-party applications, so that third-party applications do not need to know Android AIDL technology to achieve the binding ScreenShareService and communicating with a remote device through ScreenShareService. Of course, third-party applications can also use AIDL way to communicate with ScreenShareService.

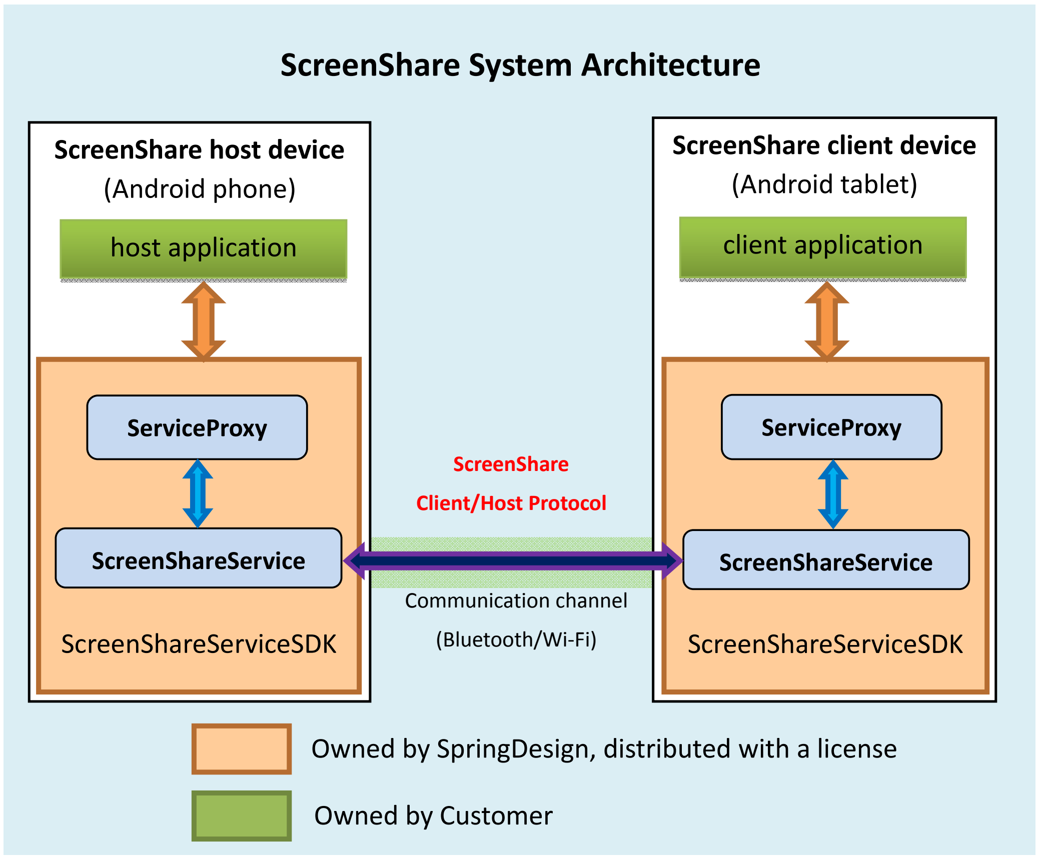
A third-party ScreenShare application contains two apks. One runs on the phone side (host), the other runs on the tablet side (client). Use different packageName for the host and the client (You can use the same packageName if you can distinguish between the host and the client). During ScreenShareService initialization, you need to specify whether it is the host.

ScreenShareService SDK on Android platform is released in the form of Android project (library). Refer SDK usage section for how to use our SDK.

Glossary

ServiceProxy	Abbreviation for ScreenShareServiceProxy

Overall architecture



Description for main classes

a) ServiceApplication

Package name: com.springdesign.screenshare.service

In the onCreate function of application class of third-party applications, it needs to call the onCreate function of ServiceApplication class to initialize ScreenShareService.

b) ServiceConfig

Package name: com.springdesign.screenshare.service

Third-party applications need to call the set function of ServiceConfig class to set the port number used by ScreenShareService (TCP/UDP port or Bluetooth UUID). Different ScreenShare applications cannot use the same port number. The port number must be assigned by Spring Design.

c) ScreenShareServiceProxy(ServiceProxy)

Package name: com.springdesign.screenshare

ScreenShareServiceProxy class provides external ScreenShareService function calls, including the enable/disable AppConnection, sendData, sendFile and other functions.

Third-party applications need to instantiate this class to call its functions to transfer data.

d) IScreenShareServiceCallbackListener

Package name: com.springdesign.screenshare

ScreenShareService callback listener interface. ScreenShareServiceProxy will notify this interface for all data sent from ScreenShareService.

Third-party ScreenShare applications need to implement this interface to receive data sent from ScreenShareService.

Interface description

1. ServiceApplication class

1) ScreenShareService initialization

Interface name	
public static boolean onCreate(Application app, booleanisHost, booleandebugMode)	
Parameter name	**Function**
App	Transfer application context
isHost	True for host; False for client.
debugMode	True for log out; False for no log
Return results	
True denotes it is currently in ScreenShareService process. Third-party applications do not need to initialize ScreenShare service. False denotes it isn't currently in ScreenShareService process. Third-party applications can initialize ScreenShare service based on needs.	

2. ServiceConfig class

1) Set up listen port for file transmission function over Wi-Fi at Host or Client side

Interface name	
public static void setTransferFileWifiPort(intfilePort)	
Parameter name	**Function**
filePort	TCP port number is assigned by Spring Design

2) Set up listen port for search function over Wi-Fi at Host side

Interface name	
public static void setSearchWifiHostPort(intsearchPort)	
Parameter name	**Function**
searchPort	UDP port number is assigned by Spring Design

3) Set up listen port for search function over Wi-Fi at Client side

Interface name	
public static void setSearchWifiClientPort(intsearchPort)	
Parameter name	**Function**
searchPort	UDP port number is assigned by Spring Design

4) Set up UUID for the first connection over Bluetooth at Host side

Interface name	
public static void setFirstUuid(String uuidStr)	
Parameter name	**Function**
uuidStr	UUID string is assigned by Spring design

5) Set up UUID for the second connection over Bluetooth at Host side

Interface name	
public static void setSecondUuid(String uuidStr)	
Parameter name	**Function**
uuidStr	UUID string is assigned by Spring design

6) Set up listen port for the first connection over Wi-Fi at Host or Client side

Interface name

public static void setFirstWifiPort(intfirstPort)	
Parameter name	**Function**
firstPort	TCP port number is assigned by Spring Design

7) Set up listen port for the second connection over Wi-Fi at Host or Client side

Interface name	
public static void setSecondWifiPort(intsecondPort)	
Parameter name	**Function**
secondPort	TCP port number is assigned by Spring Design

3. ScreenShareServiceProxy class

Third-party applications create ScreenShareServiceProxy object, and then you can use ScreenShareServiceProxy object to call functions provided by ScreenShareService.

1) Create ScreenShareServiceProxy instance

Interface name	
public ScreenShareServiceProxy(Context context, String packageName, String remotePackageName)	
Parameter name	**Function**
context	Context
packageName	Package name
remotePackageName	Remote package name

2) Get connected remote device Info

Interface name
final public String getRemoteDeviceInfo()
Return results
Null: denotes service's connection state is Not Connected Json String: {"name":"connected service name","id":"connected service id", "type":0 for WI-FI or 1 for BT, "address":"connected service network address "}

3) Get remote device list

Interface name

final public String getRemoteDeviceList()
Return results
It's a json data {"ServiceName":"My Service", "NetworkType":(0 for WiFi or 1 for BT), "DeviceList":[{"DeviceID":"", "DeviceName":"Name1", "DeviceAddress":"", "IsConnected":(true of false)}, {"DeviceID":"", "DeviceName":"Name1", "DeviceAddress":"", "IsConnected":(true of false)}, ...]}

4) Enable app connection with app of remote device

Interface name	
final public int enableAppConnection(Intent activityIntent, String tipMessage, String downloadUrl)	
Parameter name	**Function**
activityIntent	Service will start activity with this intent on remote device
tipMessage	reserved
downloadUrl	reserved
Return results	
Returns 1 if submitting to ScreenShare Service successes, others denote error. Error code: 0 denotes aidl failed, -1 denotes ScreenShare Service’s state is not connected, -3 denotes ScreenShare service is off, -5 denotes not register to ScreenShare Service, -6 denotes remote package name is wrong.	

5) Disable app connection with app of remote device

Interface name
final public int disableAppConnection()
Return results
Returns 1 if submitting to ScreenShare Service successes, others denote error. Error code: 0 denotes aidl failed, -1 denotes ScreenShare Service’s state is not connected, -3 denotes ScreenShare service is off, -5 denotes not register to ScreenShare Service, -6 denotes remote package name is wrong.

6) Get app connection status with app of remote device

Interface name
final public int getAppConnectionState()
Return results
1 : Not Connected 2 : Connected 3 : Connecting 4 : Disconnecting

7) Send byte array data to app of remote device

Interface name	
final public int sendData(byte[] buff)	
Parameter name	**Function**
buff	The data need be sent to app of remote device
Return results	
Returns 1 if submitting to ScreenShare Service successes, others denote error. Error code: 0 denotes aidl failed, -1 denotes ScreenShare Service's state is not connected, -2 denotes the request application is not in sync mode with remote device's application, -3 denotes ScreenShare service is off, -4 denotes ScreenShare Service's buffer is full, -5 denotes not register to ScreenShare Service, -6 denotes remote package name is wrong.	

8) Send byte array data with priority to app of remote device

Interface name	
final public int sendDataWithPriority(int priority, byte[] buff)	
Parameter name	**Function**
priority	Priority: 1 middle, 2 low
buff	The data need be sent to app of remote device
Return results	
Returns 1 if submitting to ScreenShare Service successes, others denote error. Error code: 0 denotes aidl failed, -1 denotes ScreenShare Service's state is not connected, -2 denotes the request application is not in sync mode with remote device's application, -3 denotes ScreenShare service is off, -4 denotes ScreenShare Service's buffer is full, -5 denotes not register to ScreenShare Service, -6 denotes remote package name is wrong.	

9) Send file to remote device and notify application (compress file during transfer)

Interface name	
final public int sendFile(String localFilePath, String remoteFilePath, String extraInfo)	
Parameter name	**Function**
localFilePath	local file path for transfer
remoteFilePath	remote file path for receiving
extraInfo	extra information for the file
Return results	
Returns 1 if submitting to ScreenShare Service successes, others denote error. Error code: 0 denotes aidl failed, -1 denotes ScreenShare Service's state is not connected, -2 denotes the request application is not in sync mode with remote device's application, -3 denotes ScreenShare service is off, -4 denotes ScreenShare Service's buffer is full, -5 denotes not register to ScreenShare Service, -6 denotes remote	

package name is wrong. After third-party application calls sendFile, ScreenShareService will callback IScreenShareServiceCallbackListener.onCallbackCalled (message.what is ON_FILE_SENT) function and notify application the current file transfer progress. In the Remote device, ScreenShareService will callback the IScreenShareServiceCallbackListener.onCallbackCalled (message.what is ON_RECEIVED_FILE) function of third-party application and notify application the current file receiving progress.

10) Send file to remote device and notify application

Interface name	
final public int sendRawFile(String localFilePath, String remoteFilePath, String extraInfo)	
Parameter name	**Function**
localFilePath	local file path for transfer
remoteFilePath	remote file path for receiving
extraInfo	extra information for the file
Return results	
Returns 1 if submitting to ScreenShare Service successes, others denote error. Error code: 0 denotes aidl failed, -1 denotes ScreenShare Service's state is not connected, -2 denotes the request application is not in sync mode with remote device's application, -3 denotes ScreenShare service is off, -4 denotes ScreenShare Service's buffer is full, -5 denotes not register to ScreenShare Service, -6 denotes remote package name is wrong. After third-party application calls sendFile, ScreenShareService will callback IScreenShareServiceCallbackListener.onCallbackCalled (message.what is ON_FILE_SENT) function and notify application the current file transfer progress. In the Remote device, ScreenShareService will callback the IScreenShareServiceCallbackListener.onCallbackCalled (message.what is ON_RECEIVED_FILE) function of third-party application and notify application the current file receiving progress.	

11) Cancel send file

Interface name	
final public int cancelSendFile(String localFilePath, String remoteFilePath, String extraInfo)	
Parameter name	**Function**
localFilePath	local file path for transfer
remoteFilePath	remote file path for receiving
extraInfo	extra information for the file
Return results	
Returns 1 if submitting to ScreenShare Service successes, others denote error. Error code: 0 denotes aidl failed.	

After third-party application calls cancelSendFile, ScreenShareService will callback IScreenShareServiceCallbackListener.onCallbackCalled (message.what is ON_FILE_SENT) function for the file is being sent. “state=-7” denotes the file transfer has been cancelled. In the Remote device, ScreenShareService will callback the IScreenShareServiceCallbackListener.onCallbackCalled (message.what is ON_RECEIVED_FILE) function of third-party application. “state=-7” denotes the file transfer has been cancelled on remote device.

12) Start http file server

Interface name	
final public String startHttpFileServer(int port, String resourcePath, String contextPath)	
Parameter name	**Function**
port	0 denotes ScreenShareService will auto select for app >0 denotes ScreenShareService will use it. If it has been used, then start will fail.
resourcePath	Resource path, should be absolute disk path. For example: Environment.getExternalStorageDirectory().getAbsolutePath()
contextPath	Context path, example: /
Return results	
Returns baseUrl, null denotes start failed	

13) Stop http file server

Interface name	
final public int stopHttpFileServer(String baseUrl)	
Parameter name	**Function**
baseUrl	The baseUrl returned by startHttpFilServer() Null means stop all started by the app
Return results	
Returns 1 if submitting to ScreenShare Service successes, others denote error. Error code: 0 denotes aidl failed.	

4. IScreenShareServiceCallbackListener class

Third-party applications need to implement ISCreenShareServiceCallbackListener to process received data.

1) Called after callback is received from app on remote device

Interface name	
public void onCallbackCalled(Message message)	
Parameter name	**Function**
message	message.what represents the callback type, Bundle stores parameters. See below: CallbackMethod.ON_RECEIVED_DATA Param: data type: ByteArray CallbackMethod.ON_RECEIVED_FILE Param: filePath type String Param: extraInfo type String Param: state type int (100 denotes receive complete, >=0 denotes receiving percent) CallbackMethod.ON_STATE_CHANGED Param: oldState type int Param: newState type int Param: reason type int Param: extraInfo type String CallbackMethod.ON_METHOD_RESULT Param: methodName type String Param: result type int Param: extraInfo type String CallbackMethod.ON_FILE_SENT Param: filePath type String Param: extraInfo type String Param: state type int (100 denotes send complete, >=0 denotes sending percent) CallbackMethod. ON_APP_CONNECTION_STATE_CHANGED Param: oldState type int Param: newState type int Param: reason type int
Return results	
Void	

2) Called after download http file request is received from app on remote device

Interface name	
public boolean onHttpDownloadFile(String url, String reserved)	
Parameter name	**Function**
url	Whole url, starts with baseUrl
reserved	Reserved for later use
Return results	
True denotes allowing download. False denotes refusing download.	

Case study

Please refer ScreenShareServiceDemo project for more code details. To get the client-side project run on tablet, change the packageName of the demo project manifest file to com.springdesign.screenshare.demo.client, then change the package name of class R imported in the source code.

1. ScreenShareService runtime environment initialization

First configure the port (assigned by Spring Design) in DemoApp.onCreate function for ScreenShareService. Then call ServiceApplication.onCreate. When the return value is false, DemoApp can do its own initialization. During initialization, it needs to create an instance of ScreenShareServiceProxy (or its subclasses) and set a callbackListener for this instance. ScreenShareService will call the callbackListener function to communicate with third-party applications. Third-party applications can actively communicate with ScreenShareService through ScreenShareServiceProxy to transfer data.

For specific codes, please check DemoApp.onCreate function and MyServiceProxy.java file.

2. Opening Service UI in the Activity of third-party apps

```
Intent intent = new Intent (this,
com.springdesign.screenshare.service.activity.MainActivity.class);
startActivity(intent).
```

3. Opening SetupWizard UI in the Activity of third-party apps

```
Intent intent = new Intent (this,
com.springdesign.screenshare.service.activity.SetupWizardActivity.class);
startActivity(intent).
```

4. Adding GoogleAnalytics for Activity

Step1: Place the google_analytics_config.xml file at the res/values directory. Set ga_trackingId with the correct value obtained through applying at GoogleAnalytics website.

All Activities that need to have analytics function must follow below steps. You can create a base class with including below codes for all Activities.

Step2: Add below code to the DemoActivity.onCreate function in DemoActivity:

```
EasyTracker.getInstance().setContext(this).
```

Step3: Add below code to the DemoActivity.onStart function in DemoActivity:

```
EasyTracker.getInstance().activityStart(this);
```

Step4: Add below code to the DemoActivity.onStop function in DemoActivity:

```
EasyTracker.getInstance().activityStop(this);
```

5. Sending byte array data to app on remote device

Step 1: Define Handler in DemoActivity to process Messages from MyServiceCallbackListener. In DemoActivity.onStart, assign Handler to MyServiceProxy instance in DemoApp. In DemoActivity.onStop, set the Handler of the CallbackListener instance in DemoApp with null value. Please refer Handler definition and onStart/onStop functions in DemoApp and DemoActivity.

Step 2: Call the sendData function of ScreenShareServiceProxy instance in DemoApp to send byte array to remote device. On remote device, after service receives data, it will call CallbackListener.onDataReceived function. This function will convert data to string to send to the handler of DemoActivity to process. You can modify the ScreenShareServiceCallbackListener.onDataReceived function code for other tasks. Please refer ScreenShareServiceCallbackListener.java and DemoActivity handler code.

6. Streaming API usage

Step 1: In MyServiceProxy, implement request validation in onHttpDownloadFile function of MyServiceCallbackListener. If the request is legitimate, it will return true. Otherwise, it will return false to deny download.

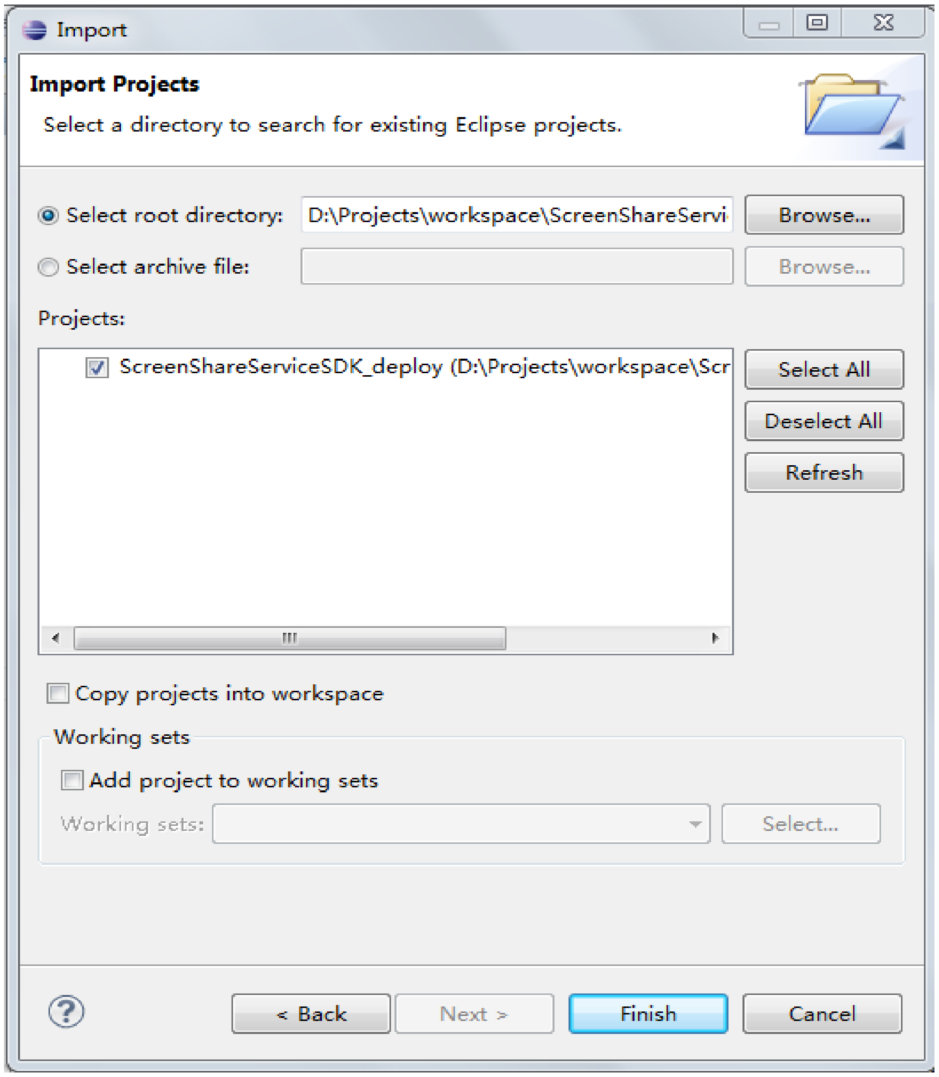
Step 2: In DemoActivity, call the startHttpFileServer of ScreenShareServiceProxy instance in DemoApp to start a file server. After an Http file server is started, app will get baseUrl (excluding IP information). App can start multiple file servers. App can use sendData interface to send the baseUrl to remote app. The remote app can use getRemoteDeviceAddress function of ScreenShareServiceProxy to get the other party's IP. Then the remote app can download files on server via access Url composed by IP and baseUrl.

Step 3: In DemoActivity, call the stopHttpFileServer of ScreenShareServiceProxy instance in DemoApp to stop a file server. If the value of baseUrl parameter is null, all file servers started by this app will be stopped.

SDK usage

1. Import SDK project to eclipse

In eclipse, select File>Import>General>Existing Projects into Workspace. Note: The encoding format in SDK project is UTF-8, as shown in below figure:



Import

Import Projects

Select a directory to search for existing Eclipse projects.

(●) Select root directory: D:\Projects\workspace\ScreenShareServi

( ) Select archive file:

Projects:

[✓] ScreenShareServiceSDK_deploy (D:\Projects\workspace\Scr

[ ] Copy projects into workspace

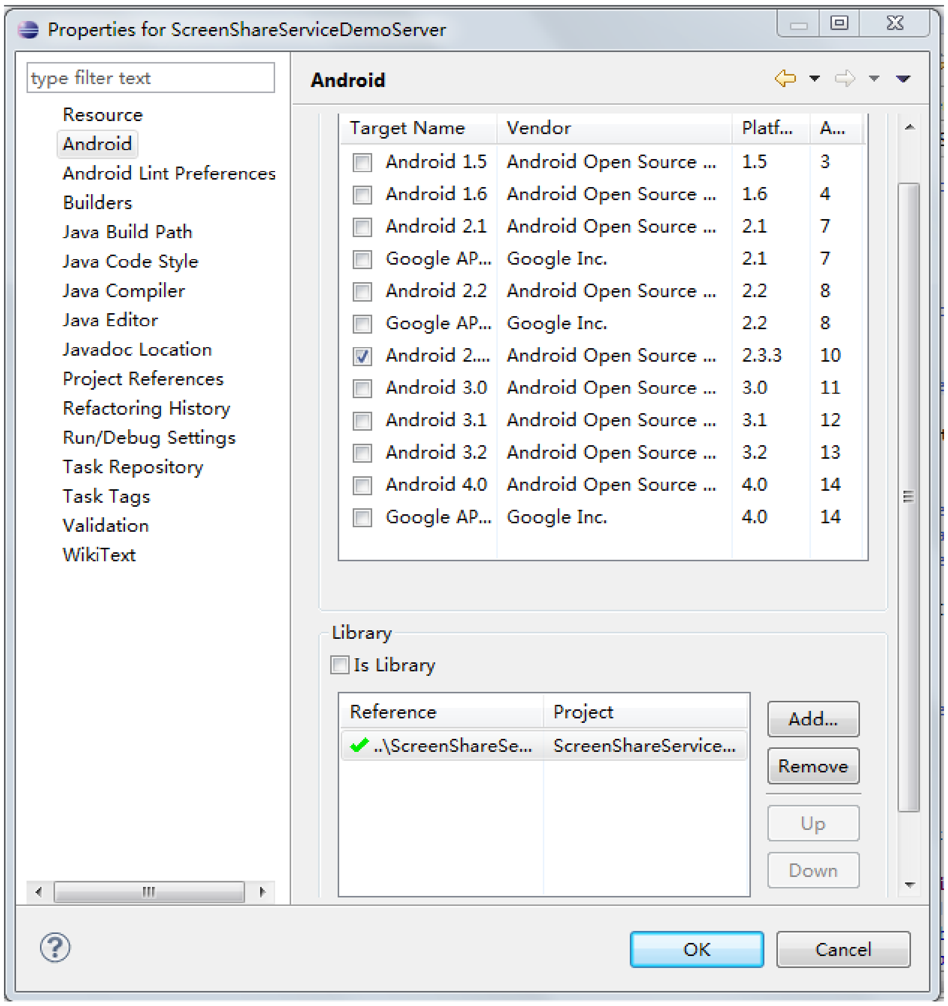
Working sets

[ ] Add project to working sets

Working sets:

2. Add Library to the project that needs to integrate SDK

Right click >Properties>Android. Set Library property, as shown in below figure:



Properties for ScreenShareServiceDemoServer

type filter text

- Resource
- Android
- Android Lint Preferences
- Builders
- Java Build Path
- Java Code Style
- Java Compiler
- Java Editor
- Javadoc Location
- Project References
- Refactoring History
- Run/Debug Settings
- Task Repository
- Task Tags
- Validation
- WikiText

Android

	Target Name	Vendor	Platf...	A...
☐	Android 1.5	Android Open Source ...	1.5	3
☐	Android 1.6	Android Open Source ...	1.6	4
☐	Android 2.1	Android Open Source ...	2.1	7
☐	Google AP...	Google Inc.	2.1	7
☐	Android 2.2	Android Open Source ...	2.2	8
☐	Google AP...	Google Inc.	2.2	8
✓	Android 2....	Android Open Source ...	2.3.3	10
☐	Android 3.0	Android Open Source ...	3.0	11
☐	Android 3.1	Android Open Source ...	3.1	12
☐	Android 3.2	Android Open Source ...	3.2	13
☐	Android 4.0	Android Open Source ...	4.0	14
☐	Google AP...	Google Inc.	4.0	14

Library

☐ Is Library

Reference	Project
✓ ..\ScreenShareSe...	ScreenShareService...

3. Manifest file must have below permission, Activity, receiver and Service disclaims:

```
    <uses-permission android:name="android.permission.DEVICE_POWER" />
<uses-permission android:name="android.permission.INTERNAL_STORAGE" />
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="adnroid.permission.ACCESS_CHECKIN_PROPERTTES" />
```

```
<uses-permission android:name="android.permission.GET_TASKS" />
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.WRITE_SETTINGS" />
<uses-permission android:name="com.android.launcher.permission.INSTALL_SHORTCUT" />
<uses-permission android:name="com.android.vending.CHECK_LICENSE" />

<!--ScreenShareServiceSDKConfig begin -->
<activityandroid:process=":remoteScreenShareService"
android:configChanges="orientation|keyboardHidden"
android:label="@string/ss_service_app_name"
android:launchMode="singleInstance"
android:screenOrientation="sensor"
android:name="com.springdesign.screenshare.service.activity.MainActivity"
android:theme="@android:style/Theme.Translucent.NoTitleBar">
<intent-filter>
<actionandroid:name="com.springdesign.screenshare.SETTINGS"/>
<categoryandroid:name="android.intent.category.DEFAULT"/>
</intent-filter>
</activity>
<activityandroid:process=":remoteScreenShareService"
android:configChanges="orientation|keyboardHidden"
android:label="@string/ss_service_app_name"
android:launchMode="singleTask"
android:screenOrientation="sensor"
android:name="com.springdesign.screenshare.service.activity.SetupWizardActivity"
android:theme="@android:style/Theme.Translucent.NoTitleBar">
</activity>

<activityandroid:process=":remoteScreenShareService"
android:name="com.springdesign.screenshare.service.activity.DeviceListActivity"
android:label="@string/ss_service_app_name"
android:theme="@android:style/Theme.Dialog"
android:configChanges="orientation|keyboardHidden"/>

<receiverandroid:process=":remoteScreenShareService"
android:label="@string/ss_service_app_name"
android:name="com.springdesign.screenshare.service.receiver.ReadmateServiceBootReceiver">
<intent-filter>
<actionandroid:name="android.intent.action.BOOT_COMPLETED"/>
<categoryandroid:name="android.intent.category.LAUNCHER"/>
</intent-filter>
</receiver>
```

```
<serviceandroid:process=":remoteScreenShareService"
android:name="com.springdesign.screenshare.ScreenShareService"
android:icon="@drawable/ss_service_icon">
</service>
<!--ScreenShareServiceSDKConfig end -->
```

4. You can change GoogleAnalyticsConfig parameters in the ss_service_config.xml at the res/values directory under ScreenShareServiceSDK_deploy project.