# WiseMo Host for Android Management Guide

## 1.1 Overview

The WiseMo Host is an application that runs on Android devices to provide management tasks such as remote desktop or file transfer to a Guest user. While being useful on its own, the Host can be a part of a larger solution. This document explains how 3rd party application vendors can integrate the WiseMo Host into their products and solutions.

There are four methods available to manage the WiseMo Host from other Android apps:

- Using the manageability API IHostManagement.aidl as a bound service
- Start the Host app using the `Context.startActivity()` with intent
- Configure the Host with Enterprise Managed Configuration
- Some of the nuanced aspects of Host behavior on Android platform can be configured with option files

## 1.2 What's new

2025-06-12: Corrected typos and some rephrasing.

2025-02-04: There are two new methods onAddonDownloadStart() and onAddonDownloadComplete() and section "Third party add-on installation" were added to this document.

2024-02-13: Two new methods, saveSupportLog() and stopHost(), have been added to IHostManagement.aidl. The first method generates the Host support log and the second method allows a Device Admin app to stop the Host communication.

## 1.3 Host Management Service

The WiseMo Host for Android exposes the management API as an Android bound service component. An external app binds to the Host Management service to be able to send commands to the Host, like restarting the Host, assigning the Host XML configuration file or license key, etc.

There is a package with a sample app and source codes GetHostName-DemoApp-src.zip / GetHostName-DemoApp.apk that be a starting point for testing and integrating the Host Management API into your app.

## 1. Binding to the HostManagement service

First you need the IHostManagement.aidl from GetHostName-DemoApp to be in the AIDL directory of your project, e.g., "`aidl\com\wisemo\host\IHostManagement.aidl`".

To binding to Host Management service, use the Android API for bound services. For example (Java):

```
Intent it = new Intent();
it.setAction("com.wisemo.intent.action.CONNECT_HOST_MANAGEMENT");
it.setComponent(new ComponentName("com.wisemo.host.v10", "com.wisemo.host.HostMana
gementService"));
if (!bindService(it, mServiceConnection, Service.BIND_AUTO_CREATE))
    log("Failed binding to HostManagement. Please verify if Host is installed.");
```

In the `mServiceConnection` callback it is possible to obtain the object with the `IHostManagement` interface that your app can use to manage the Host:

```
@Override
public void onServiceConnected(ComponentName name, IBinder service) {
    mHostManagementService = IHostManagement.Stub.asInterface((IBinder) service);
}
```

With the `IHostManagement` instance obtained, the app can now send commands to the Host:

```
mHostManagementService.restartHost(); // Restart the WiseMo Host Service
```

## 2. IHostManagement API

The description of available API methods can be found in IHostManagement.aidl. Please note that some of the methods require the caller app to be an Android device administrator:

```
// Service API for management Host Application.
interface IHostManagement
{
   const int STARTUP_MODE_UNATTENDED = 0;
   const int STARTUP_MODE_MIXED = 1;

   // Error codes for methods like setLicense/setConfig
   const int ERROR_NONE           = 0; // Completed succesfully
   const int ERROR_GENERAL        = 1; // Other error
   const int ERROR_ADMIN_REQUIRED = 2; // Caller app has to be device admin for this function
   const int ERROR_IO             = 3; // I/O error
   const int ERROR_INVALID_STATE  = 4; // Operation is not possible in current Host state

   // Flag for startActivity() to start Host. With this mode Host ask user about everything
   // required to start Host Service (storage permissions, license, security configuration etc.) and
   // as soon as HostService started activity is stopped automatically. It is recommended to
   // set this flag together with "intent.setFlags(Intent.FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS);"
   const String START_HOST_VIA_ACTIVITY  = "com.wisemo.host.StartHostViaActivity";

   // Flag to startActivity() to start Host. When "true", it creates "opt.no-knox-device-admin-promp
t"
   // and on Samsung device Host do not ask by default to assign it as device admin. When "false",
   // Host removes "opt.no-knox-device-admin-prompt".
   const String START_HOST_NO_ADMIN_KNOX = "com.wisemo.host.StartHostNoAdminKnox";

   // Gets the name of Host
   String getHostName();

   // Restarts the Host. Returns false if Host state is not running.
   // This method does not wait for the Host to be restarted. There is no
   // API yet for Management Application to notify when restart is finished.
   boolean restartHost();

   // Starts the Host service in unattended mode. It must be invoked by device admin. Returns
   // true if host service start has been scheduled.
   boolean startHostUnattended();

   // Stops the Host. Host hast to be in running or connected state. This method must be invoked
   // by device administrator.
   //
   // Returns ERROR_NONE when Host was successfully scheduled to stop. The ERROR_INVALID_STATE is
   // returned when it is not possible for Host stop in current state, for example Host is
   // stopped, stopping or starting. The error ERROR_ADMIN_REQUIRED is returned when caller app is
   // not device admin.
   boolean startHost(String maintenancePassword);

   // Saves the provided data as Host XML configuration file. The caller of this method must
   // be device admin.
   int stopHost(int mode);

   // Saves the provided data as Host XML configuration file. The caller of this method must
   // be device admin.
   //
   // On success returns ERROR_NONE.
   int setConfig(in byte[] config);

   // Saves the provided string as Host license. The caller of this method must be device admin.
   //
   // On success returns ERROR_NONE.
   int setLicense(String license);

   // Saves the support log as *.zip archive.
   //
   // MDM solutions can use this method to collect logs from Host. An MDM Agent creates a file
   // in its private folder or where it has the read/write access and invokes this API method
   // to provide the file descriptor for Host. When the file is provided as file descriptor Host
   // can write the compressed logs into it. Once this method returns, the MDM Agent can submit
   // the zip archive with logs for troubleshooting.
   //
   // On success returns ERROR_NONE.
   int saveSupportLog(in ParcelFileDescriptor zipFileFd);



   // Notifies the Host that third party has started the add-on download process.
   //
   // This is the part of the Third-Party Add-on Installation feature. When Host detects that
   // there is an add-on for this device, it verifies if there’s a known third party app
   // on the device that is able to download and install the add-on. Once such an app is found, Host
   // uses the sendBroadcast() with Intent that contains details of the add-on to be downloaded.
   //
```

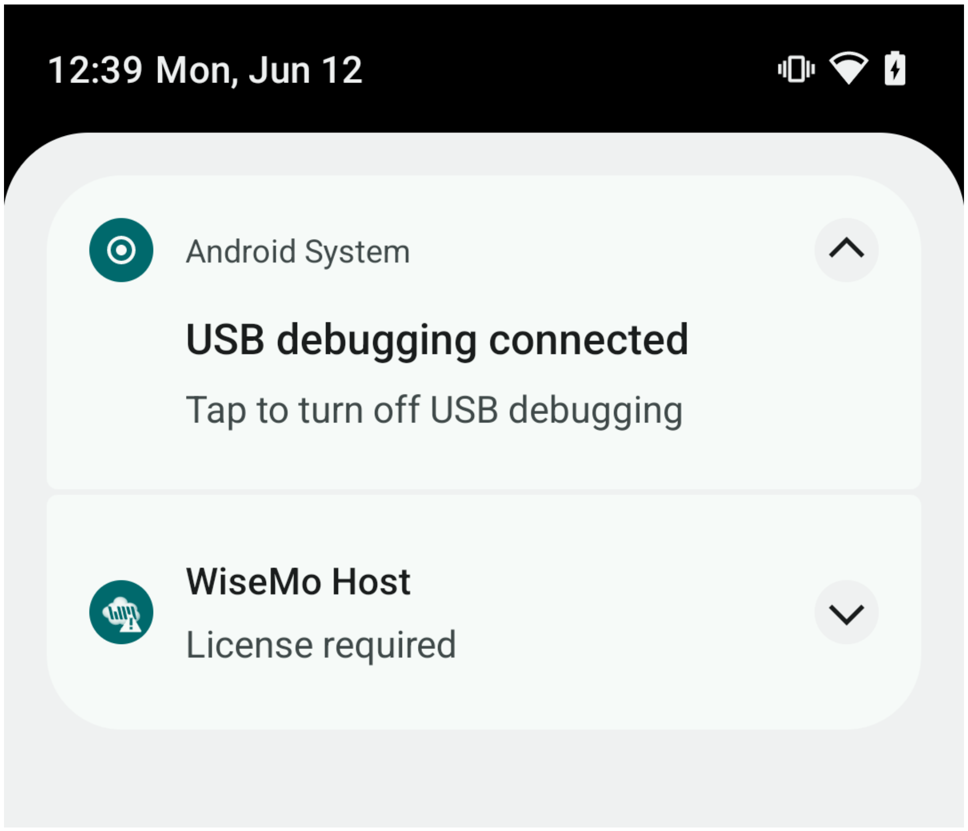
```
    // The third-party app on receiving the broadcast should invoke this API method to confirm that
    // it intends to handle it. Host assigns a rather short timeout (2 sec) for this
    // confirmation. With no confirmation receiving in provided time frame, Host proceeds with its
    // start procedure and issues the user prompt to download and install the add-on.
    //
    // Note that whenever Host does not detect the presence of known third party app, it does not
    // send the broadcast to avoid start up delays.
    //
    // On success returns ERROR_NONE. When Host does not expect this callback, it returns
    // ERROR_INVALID_STATE.
    //
    // More details can be found Android_Host_Management_Guide.pdf.
    int onAddonDownloadStart();


    // Notifies the Host that third party app has completed the add-on download and installation
    // process.
    //
    // Params:
    //    success - indicates if download was completed successfully.
    //    details - reserved for future use. If there is a string value named "message" in the
    //              bundle Host will log it.
    //
    // This is the part of the Third-Party Add-on Installation feature. After getting the
    // onAddonDownloadStart() confirmation, Host assigns a longer timeout (60 seconds) for the app
    // to download and install the add-on and report the completion by invoking the
    // onAddonDownloadComplete().
    //
    // Host would make a user prompt to download and install add-on if the third-party app
    // does not call this API method in time or it reported a failure in the first parameter.
    //
    // When the app reported success, Host continues the start without prompting for add-on
    // installation.
    //
    // It does not verify if the add-on was actually installed as reported.
    //
    // On success returns ERROR_NONE. When Host does not expect this callback, it returns
    // ERROR_INVALID_STATE.
    int onAddonDownloadComplete(in boolean success, in Bundle details);
}
```

## 1.4 Starting the Host Service

Usually, an app would want the Host Service to start in the background without any interaction with the user. However, it may not always be possible, in some cases the user must interactively confirm some requests on first start.

The `IHostManagement` interface has two methods to start the host: `startHostUnattended()` and `startHost(int mode)`. The `startHostUnattended()` and `startHost(IHostManagement.STARTUP_MODE_UNATTENDED)` are equivalent and can be used to start the Host in unattended mode.

In unattended mode, the Host doesn't popup any UI prompts to the user. Whenever the Host needs something to start successfully, it adds a notification:


12:39 Mon, Jun 12

Android System

USB debugging connected

Tap to turn off USB debugging

WiseMo Host

License required

The user must tap the notification to resolve the issue that prevents the Host service from starting.

Unattended mode is not only caused by this API but also in other cases when the Host is started in background, for example on device boot. Android 10 and above restricts UI access from background apps, so notifications are often the only possible way to notify the user.

If you don't want to suppress everything when starting the Host from your app, there are two more methods:

- There is `IHostManagement.startHost()` method with `HostManagement.STARTUP_MODE_MIXED` parameter.
- App can start the Host main activity using Android `Context.startActivity()` with `IHostManagement.START_HOST_VIA_ACTIVITY` boolean flag (see below)

The `HostManagement.STARTUP_MODE_MIXED` parameter starts the Host service in the background, but whenever there is a startup issue, it is shown to user as a full screen activity or alert dialog prompting for immediate action. Please note that this method may no longer works from Android 10 where the restrictions on starting activities from the background were introduced.

Another method is to start the Host's main activity with `IHostManagement.START_HOST_VIA_ACTIVITY` extra flag:

```
final Bundle extras = new Bundle();
// The name of the extra bool flag to launch Host service with startActivity().
extras.putBoolean(IHostManagement.START_HOST_VIA_ACTIVITY, true);

final Intent intent = new Intent(Intent.ACTION_MAIN);
intent.setComponent(new ComponentName("com.wisemo.host.v10","com.wisemo.host.Host"
));
// It is better to exclude Host from recent when starting activity this way,
// otherwise the activity is seen in Recent and when user attempts to launch it
// it hides immediately.
intent.setFlags(Intent.FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS);
intent.putExtras(extras);

startActivity(intent);
```

If the Host's main activity is started with `IHostManagement.START_HOST_VIA_ACTIVITY`, the main activity is automatically closed when there are no more starting issues, and the Host service has been started successfully. However, when there is an issue that prevents the Host Service from starting, it will be shown to user as an activity or alert dialog.

On Samsung devices remote control is provided by the Knox API. Your app can control if the Host should prompt the user to assign it as Device Admin. You can use the `IHostManagement.START_HOST_NO_ADMIN_KNOX` in the same way as `START_HOST_VIA_ACTIVITY`:

- true: device admin is optional, Host does not prompt user about assigning Host as device admin on first start. Without device admin priviledge the Host is unable to do a remote reboot. The Host still has to prompt the user to activate the Knox license in order to get access to the Samsung RC API.
- false: device admin is required before KNOX license can be activated.
- none (no START_HOST_NO_ADMIN_KNOX in Intent's extras): use Host default mode.

## 1.5 Android Managed Configuration

If an app is a device admin app, it can assign enterprise managed configurations to other apps (previously known in Android documentation as *application restrictions*). The Host supports the assignment of key configuration options through the Enterprise Management Configuration API.

For more info about managed configuration support in Android please visit https://developer.android.com/work/managed-configurations

It is possible to test managed configuration with the Test DPC tool developed by Google.

This is how the Host handles managed configuration:

- On Host start, the Host reads the values from managed configuration and saves it to the Host XML config file.
- Values from managed configuration overwrites values in the Host XML config. For example, if the Host is configured via an EMM solution and the user changes some of these settings with the built-in UI or WiseMo Host Manager, these changes will be reverted on next Host restart.
- When the Host is running, it will receive an event if the EMM config has been updated by the EMM tool. In this case the Host restarts and applies the new settings.

The specification management configuration parameters can be found in Appendix A in our guide for Intune Deployment and Configuration for Host: WiseMo_Intune_Android_Host_Deployment.pdf.

## 1.6 Host option files

The Host can be customized using option files. For instance, if you want to prevent the Host from prompting for add-on installation at startup, your management app can create the following file on the system:

- /sdcard/Android/data/com.wisemo.host.v10/files/WsmHost/opt.no-rcbridge-ui

When Host starts it checks if this file exists and suppresses the add-on installation UI prompt.

Your management app must have external storage write access to be able to create such option files for the Host.

The full list of the option files can be found in our guide for Host advanced installation and configuration: Android Host advanced installation.pdf

## 1.7 Third party add-on installation

The host can work with an MDM tool on the device to download and install the Remote Control Add-on for the host:

1. At startup, the Host checks for an add-on that supports remote control of the current device.
2. The host checks for a third-party management app that can download and install the add-on automatically. If none is found, the host prompts the user to download and install the add-on manually.
3. With the known third-party management app detected, Host sends a broadcast with the Intent object to this app. The intent has specific action name and the extras parameters like “url” and “filename” attached, for example:

```
var INSTALL_APP_ACTION = "..."
var APPLICATION_NAME = "..."

var COMPONENT_NAME = "..."

var intent = Intent(INSTALL_APP_ACTION)

intent.component = ComponentName(APPLICATION_NAME, COMPONENT_NAME)

intent.putExtra("url","https://www.wisemo.com/patcher/Content/RcBridge/RcBridge.crosscall_m5.a
pk")

intent.putExtra("filename","addon.apk")

sendBroadcast(intent)
```

The constants INSTALL_APP_ACTION, APPLICATION_NAME and COMPONENT_NAME are provided by the MDM app developer for integration to Host.

1. Whenever the management app receives the broadcast, the app should invoke the onAddonDownloadStart() method from the IHostManagement interface.

   The Host assigns a timeout for the app to invoke onAddonDownloadStart(), see description of this method in AIDL file for details.

2. After the app finishes downloading and installing the add-on, it should call the onAddonDownloadComplete() method. The host sets a longer timeout for this process. For more details, refer to the onAddonDownloadComplete() method in the AIDL file.

3. As soon as the Host is notified with onAddonDownloadComplete(), it continues the start process.