MMDAgent Developer Reference
Ver. 1.01

September 29, 2016
Nagoya Institute of Technology
## Contents

1. About this document ............................................................................................................. 4  
2. Building the development and run-time environments (Windows) ............................... 5  
   Overview ............................................................................................................................... 5  
   Building the development environment ........................................................................... 6  
      Downloading Visual Studio Community 2013 ............................................................... 6  
      Installing Visual Studio Community 2013 ................................................................. 7  
      Installing the Visual Studio 2013 Language Pack ..................................................... 8  
      Initial configuration of Visual Studio Community 2013 ........................................... 9  
      Setting Visual Studio Community 2013 to Japanese ............................................... 10  
   Building the run-time environment ................................................................................. 11  
      Getting the source code ............................................................................................... 11  
      Building and running the source code ....................................................................... 12  
   Developing a new plugin .................................................................................................. 17  
      Adding and configuring a new project ........................................................................ 17  
      Creating, building and running a file ........................................................................ 26  
      Implementable function set ......................................................................................... 29  
      Simple implementation example ................................................................................ 35  
      Plugin template ............................................................................................................ 38  
3. Building the development and run-time environments (Android) ................................. 39  
   Overview ............................................................................................................................... 39  
   Building the development environment ........................................................................ 40  
      Downloading Java SE Development Kit 7 .................................................................. 40  
      Installing the Java SE Development Kit 7 ................................................................. 42  
      Configuring environment variables (JAVA_HOME) .................................................... 44  
      Downloading Android Studio ...................................................................................... 47  
      Installing Android Studio ............................................................................................. 49  
      Downloading Android NDK .......................................................................................... 53  
      Installing Android NDK ............................................................................................... 54  
   Building the run-time environment ................................................................................ 55  
      Getting the source code and sample script ................................................................ 55  
      Creating a new project ................................................................................................... 56  
      Changing how projects are displayed ......................................................................... 59  
      Creating a JNI folder ..................................................................................................... 60  
      Importing source code ................................................................................................. 61  
      Creating system and content directories on an Android terminal ............................ 63
Editing files .................................................................................................................. 65
Building and running source code ........................................................................ 76
1. About this document

This document provides specifications for developers extending MMDAgent functionality. It describes how to build the development and run-time environments on Windows, how to develop a new plugin, and how to build development and run-time environments on Android.

This document deals with the following versions of MMDAgent.

▼ Versions

<table>
<thead>
<tr>
<th>Software</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMDAgent.exe</td>
<td>1.6.1</td>
</tr>
<tr>
<td>MMDAgent_Example</td>
<td>1.6</td>
</tr>
</tbody>
</table>
2. Building the development and run-time environments (Windows)

Overview

This section summarizes procedures to develop and run published MMDAgent source code on Windows and to develop a new plugin for MMDAgent.

This section was written based on the following environment, but development can also be done on Windows 8. In doing so, adjust the document details according to your own environment.

<table>
<thead>
<tr>
<th>Software</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development OS</td>
<td>Windows 7 64 bit</td>
</tr>
<tr>
<td>Development software</td>
<td>Visual Studio Community 2013</td>
</tr>
</tbody>
</table>

[Notes]

The published MMDAgent solution file (.sln) and project file (.vcxproj) were created for Visual Studio 2010, but later versions can be used by upgrading the project (when the solution is first opened, a request to upgrade will automatically appear).
Building the development environment

Downloading Visual Studio Community 2013

Download Visual Studio Community 2013 and the Japanese Language Pack from Microsoft.

https://www.visualstudio.com/downloads/download-visual-studio-vs

Procedure

(1) Select Visual Studio Community 2013

(2) Download Visual Studio Community 2013 (ISO format is recommended due to size)

(3) Download the Visual Studio 2013 Language pack (Japanese)
Installing Visual Studio Community 2013

Run the downloaded install file (ISO or Web installer) to install Visual Studio Community 2013.

▼ Procedure

1. Select checkbox
2. Select next
3. Select INSTALL
Installing the Visual Studio 2013 Language Pack
Run the downloaded Language Pack (Japanese) installer to install it. Note that Visual Studio settings are required to complete the conversion to Japanese, and these changes are described below.

▼ Procedure

(1) Select checkbox

(2) Select Install
Initial configuration of Visual Studio Community 2013
Run Visual Studio Community 2013 after installation to perform initial configuration.

▼ Procedure (the configuration window appears automatically upon first launch)

![Visual Studio configuration window](image)

(1) Select Not now, maybe later
(2) Select Start Visual Studio

[Notes]
To continue using it beyond the evaluation period (30 days) requires signing in with a Microsoft account.
Setting Visual Studio Community 2013 to Japanese

Visual Studio Community 2013 can be set to Japanese in the Options settings.

▼ Procedure (the change is applied upon restarting Visual Studio)

(1) Select Options
(2) Select Environment
(3) Select International Settings
(4) Select Japanese
(5) Select OK
Building the run-time environment

Getting the source code

Download the MMDAgent source code from the Web site and save it in a folder on your PC. The source code is archived in a Zip file, which must be extracted before performing the procedures below.

▼ MMDAgent Web site
http://www.mmdagent.jp/

▼ Procedure
Building and running the source code
This section describes preparations to run the source code with Visual Studio Community 2013.

▼ Procedure
1. Open the solution
   Open the MMDAgent_vs2010.sln file in the extracted folder with Visual Studio Community 2013. When opening the solution, a window to upgrade to Visual Studio 2013 will appear, so select “OK” to upgrade the entire project.
2. Set the startup project
Set the startup project to main

[Notes]
The name of the startup project will be shown in bold.

3. Change the solution configuration
Change the solution configuration to Release.

[Notes]
If executed in Debug configuration, AppData and MMDAgent.mdf will be copied from the Release folder to the Debug folder.
4. **Build**

**Build the solution**

1. Select **Build the solution**.

2. **Build is complete if there are no errors.**
5. Run

Run the exe file generated by the build.

(1) Select Start debug

(2) If MMDAgent starts, you are done
[Notes]
There is a Contents Package in the extracted folder from the Web site. By giving the path of MMDAgent_Example.mdf as a parameter when running MMDAgent, development can be done from an initial configuration that displays a character called “Mei”.

▼ Configuration procedure
Open the project properties for main and under [Configuration properties] -> [Debug], set Command arguments to the path of MMDAgent_Example.mdf.

▼ Screen shot
Developing a new plugin
Adding and configuring a new project
This section describes how to add a new project and prepare to develop a new plugin.

▼ Procedure
1. Add a project
Add a project to the solution. When a new project is added in the solution explorer, the Add Project window appears. Initialize as shown in the screen shots below.

(1) Right-click on the solution
(2) Select Add
(3) Select New project
(4) Select Visual C++
(5) Select Win32 project
(6) Enter the project name (e.g. Plugin_Sample)
(7) Select OK
18
2. Change the project settings

Change the settings of the added project. After the property page window appears, complete settings I to V below.

(1) Right-click the project

(2) Select Properties

[Notes]

To run a Debug configuration, change the solution configuration to Debug and then perform these settings.
I. Character set settings
Change the [Character set] setting in [Configuration properties] - [General] to [Uses multi-byte character set].

▼ Screen shot
II. Set additional include directories
Enter the following values in [Additional include directories] in [Configuration properties]->[C/C++]->[General].

▼ Setting value

```
..\Library_GLee\include;..\Library_Bullet_physics\include;..\Library_MMDFiles\include;..\Library_Julius\include;..\Library_MMDAgent\include;..\Library_GLFW\include;
```

▼ Screen shot
III. Set additional library directories

▼ Setting value
```
..\Library_Bullet_Physics\lib;..\Library_MMDFiles\lib;..\Library_zlib\lib;..\Library_libpng\lib;..\Library_GLee\lib;..\Library_MMDAgent\lib;..\Library_JPEG\lib;..\Library_GLFW\lib;..\Library_FreeType\lib
```

▼ Screen shot
![Screen shot](image)
IV. Added dependency file settings
Set [Additional dependency files] in [Configuration property] -> [Linker] -> [Input] to the following value. It is okay to overwrite the value input initially.

▼ Setting value

```
MMDFiles.lib;libpng.lib;Bullet_Physics.lib;winmm.lib;opengl32.lib;
glu32.lib;zlib.lib;GLEe.lib;MMDAgent.lib;JPEG.lib;GLFW.lib;FreeType.lib;%{AdditionalDependencies}
```

▼ Screen shot

![Screen shot](image_url)
V. Set the output directory
Set [Output directory] in [Configuration properties]->[General] to the following value.

▼ Setting value

\$(SolutionDir)\$(Configuration)¥Plugins¥

▼ Screen shot
3. Set the project dependency relationships

Set the dependency relationships for the added project.

(1) Right-click on Project

(2) Select Build dependencies

(3) Select Project dependencies

(4) Select Library_MMDAgent and Library_MMDFiles

(5) Select OK
Creating, building and running a file
Add a source file and test run it.

▼ Procedure
1. Add the source file
   Add the source file to the project.
2. Enter test code

Enter the following test code into the added source file.

```cpp
#ifdef _WIN32
#define EXPORT extern "C" __declspec(dllexport)
#else
#define EXPORT extern "C"
#endif

#include "MMDAgent.h"

EXPORT void extAppStart(MMDAgent *mmdagent)
{
    // Log output: Plugin_Sample
    mmdagent->sendMessage("Plugin_Sample", "]");
}
```
3. **Build/Run**

   Build and Run in the same way as was done when building the run-time environment, and check that Plugin_Sample was output to the log.

   ▼ Screen shot

   ![Screen shot](image)

   (1) Build/run was successful if Plugin_Sample appears in the log

   [Notes]

   The plugin (.dll) is generated in the Release\Plugins folder, which is at the same level as MMDAgent_vs2010.sln.
Implementable function set
Functions that can be implemented in an MMDAgent plugin are described here. When implementing the functions below, the following EXPORT definitions and inclusion of MMDAgent.h at the beginning of source code is required.

▼ Required code

```c
#ifdef _WIN32
#define EXPORT extern "C" __declspec(dllexport)
#else
#define EXPORT extern "C"
#endif

#include "MMDAgent.h"
```
The MMDAgent class

Functions that can be implemented in a plugin are passed a pointer argument that gives them access to the MMDAgent class. This argument allows use of functions published by the MMDAgent class, such as for issuing internal messages. The syntax and an example of the method for issuing an internal message (sendMessage) are given below. Other published functions are described in Library_MMDAgent\include\MMDAgent.h.

- Syntax

```
void sendMessage(const char *type, const char *format, ...);
```

- Description

Issues an internal message.

- Arguments

  - type
    Type of internal message.

  - format (variable length argument)
    Format specifier. Same as for printf in the C Standard Library.

- Return value

  None

- Example

```
mmagent->sendMessage("Plugin_Sample", "%s", "Arg");
```
extAppStart Function

▼ Description
This function is called once when MMDAgent launches. It is used to initialize the plugin.

▼ Syntax

```
EXPORT void extAppStart(MMDAgent *mmdagent) {}
```

[Arguments]

・mmdagent
  Reference value to access MMDAgent functions.

[Return value]

・void
  None

extAppEnd Function

▼ Description
This function is called once when MMDAgent exits (when the window is closed). It is used to terminate the plugin.

▼ Syntax

```
EXPORT void extAppEnd(MMDAgent *mmdagent) {}
```

[Arguments]

・mmdagent
  Reference value to access MMDAgent functions.

[Return value]

・void
  None
extProcMessage Function

▼ Description
This function is called when an internal message in MMDAgent (EventMessage or CommandMessage) is issued. It is used to include any processing for the issued message.

▼ Syntax

```c
EXPORT void extProcMessage(MMDAgent *mmdagent, const char *type, const char *args) {}
```

[Arguments]

- **mmdagent**
  Reference value to access MMDAgent functions.

- **type**
  The type of the issued internal message.

- **args**
  Contents of the issued internal message.

[Return value]

- **void**
  None
extUpdate Function

▼ Description
This function is called when MMDAgent performs an update. It is used to perform update processing as time progresses within a scene.

▼ Syntax

```
EXPORT void extUpdate(MMDAgent *mmdagent, double deltaFrame) {}
```

[Arguments]

• `mmdagent`
  Reference value to access MMDAgent functions.

• `deltaFrame`
  The frame difference elapsed since the previous update process. Units of 1/30 s.

[Return value]

• `void`
  None
extRender Function

▼ Description
This function is called whenever MMDAgent performs rendering. It is used to perform processing synchronized with vertical refresh.

▼ Syntax

```c
EXPORT void extRender(MMDAgent *mmdagent) {};
```

[Arguments]
・`mmdagent`
  Reference value to access MMDAgent functions.

[Return value]
・`void`
  None
Simple implementation example

The following simple implementation example is a plugin that outputs MMDAgent messages to a file.

▼ Plugin_LogMessage.cpp

```cpp
/* definitions */
#ifdef _WIN32
#define EXPORT extern "C" __declspec(dllexport)
#else
#define EXPORT extern "C"
#endif /* _WIN32 */
#define LOGFILENAME "MessageLog.txt" /* Log file name */
#define PLUGINLOGMESSAGE_NAME "LogMessage" /* Plugin name */

/* Message types related to the log file (Any internal message types can be defined) */
#define MMDAGENT_EVENT_FILEOPEN "LOGMESSAGE_EVENT_FILEOPEN"
#define MMDAGENT_EVENT_FILECLOSE "LOGMESSAGE_EVENT_FILECLOSE"

/* headers */
#include "MMDAgent.h"
#include <fstream>
#include <ctime>

/* variables */
static bool enable;
static std::ofstream ofs;
static time_t t;
static tm *x;

/* extAppStart: load models and start thread */
EXPORT void extAppStart(MMDAgent *mmdagent)
{
    enable = true;
    mmdagent->sendMessage(MMDAGENT_EVENT_PLUGINENABLE, "%s", PLUGINLOGMESSAGE_NAME);
    /* File name used to create the log */
```
const char *fileName = LOGFILENAME;

/* Open the log file (append) */
ofs.open(fileName, std::ios::out | std::ios::app);
if (!ofs) {
    /* Issue a message indicating file open failure */
    mmdagent->sendMessage(MMDAGENT_EVENT_FILEOPEN, "%s can not be opened!", fileName);
}
else {
    /* File opened successfully */
    /* Issue a message that file opened successfully */
    mmdagent->sendMessage(MMDAGENT_EVENT_FILEOPEN, "%s can be opened", fileName);

    /* Get the current time */
t = time(0);
    char buf[32];
    ctime_s(buf, sizeof(buf), &t);

    /* Write the time */
    ofs << buf;
    ofs << "[[Start]]" << std::endl;
}

/*============================================================================*/

EXPORT void extProcMessage(MMDAgent *mmdagent, const char *type, const char *args)
{
    if (enable == true) {
        /* Output to output stream (1 line) */
        /* By removing the comments from the following, the log file will only be written*/
        /* for a particular message type (speech input) */
        // if (MMDAgent_strequal(type, "RECOG_EVENT_STOP"))
        {
            ofs << type << "|" << args << std::endl;
        }
    }
}
The output log file

When the plugin runs correctly, the file MessageLog.txt is output in the current directory (normally the same folder as MMDAgent.exe).
Plugin template

The following is a template containing empty functions that can be used in a plugin.

```cpp
#include "MMDAgent.h"

EXPORT void extAppStart(MMDAgent *mmdagent) {}

EXPORT void extAppEnd(MMDAgent *mmdagent) {}

EXPORT void extProcMessage(MMDAgent *mmdagent, const char *type, const char *args) {}

EXPORT void extUpdate(MMDAgent *mmdagent, double deltaFrame) {}

EXPORT void extRender(MMDAgent *mmdagent) {}
```
3. Building the development and run-time environments (Android)

Overview

This section summarizes procedures for running the published MMDAgent source code on Android. Note that the development machine OS is windows.

This section is written for the environment given below, but the software can be developed and run on other environments (e.g. Windows 8 or greater, other versions of Android Studio, etc.). In such cases, adjust the descriptions in this document as necessary for your environment.

The version of the JDK required will depend on the version of Android Studio being used, so check the system requirements on the “Android Studio and SDK Tools Downloads” page on the Android developer site (https://developer.android.com).

<table>
<thead>
<tr>
<th>Software</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dev. Environment OS</td>
<td>Windows 7 64 bit</td>
</tr>
<tr>
<td>Run-time environment OS</td>
<td>Android 5.0.2 (4.0 or greater recommended)</td>
</tr>
<tr>
<td>Development software</td>
<td>Android Studio 1.4</td>
</tr>
<tr>
<td>JDK Version</td>
<td>Java SE Development Kit 7u80</td>
</tr>
<tr>
<td>Android SDK version</td>
<td>r24.3.4</td>
</tr>
<tr>
<td>Android NDK version</td>
<td>r10e</td>
</tr>
</tbody>
</table>

[Notes]

It will be necessary to enable developer options on the Android terminal used as the run-time environment.
Building the development environment

Downloading Java SE Development Kit 7

Download the Java SE Development Kit 7 installer from the Web site.

▼ Procedure

1. Select this check box
[Notes]

JDK is the Java SE Development Kit, and JRE is the Java Runtime Environment.
Installing the Java SE Development Kit 7

Install the Java SE Development Kit 7 using the downloaded installer. During the installation, the JRE install screen will appear, so follow the instructions to install the JRE as well.

▼ Procedure (JDK install)

* The install path can be changed, but this will affect settings later on, so it is not recommended.
Procedure (JRE install, window appears automatically)

1. Select Next
2. Select Close
Configuring environment variables (JAVA_HOME)
Register the path of the installed JDK in an environment variable.
The environment variable to be registered is as follows.

▼ Environment variable

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable name</td>
<td>JAVA_HOME</td>
</tr>
<tr>
<td>Variable value</td>
<td>C:\Program Files\Java\jdk1.7.0_80</td>
</tr>
</tbody>
</table>

▼ Procedure

* The window above can be displayed with [Control panel] -> [System].
(2) Select Environment Variables...

(3) Select New...

(4) Enter JAVA_HOME for variable name

(5) For Value, enter C:\Program Files\Java\jdk1.7.0_80

(6) Select OK
(7) Check that the variable has been added to system environment variables

(8) Select OK

(9) Select OK
Downloading Android Studio

Download Android Studio from the Web site.

▼ Procedure

Download Android Studio IDE
Android SDK Tools
Android 6.0 Marshmallow フォーム
Google API を使用した Android 6.0 エミュレータのシステムイメージ

(1) Select DOWNLOAD
The version of Android Studio on the Web site is always the latest version, but past versions can be obtained at the following site.

http://tools.android.com/download/studio/canary
Installing Android Studio

Install Android Studio with the downloaded installer.

▼ Procedure

1. Select Next

2. Select Next
* Any path can be used for the installation, but changing it is not recommended because it will affect other settings described below.

(5) Check the install location for the Android SDK. (the grayed-out area will contain the user name)

(6) Select Next

(7) Select Next
[Notes]
The Android SDK is also required for Android development, but if the full version of Android Studio was used, the Android SDK is included and does not need to be installed separately.
Downloading Android NDK

Download the Android NDK from the Web site.

Procedure

1. SELECT the NDK for your development environment

2. Select this checkbox

3. Download the NDK
Installing Android NDK
Run the downloaded installer to install the Android NDK.

▼ Procedure
1. Run the installer
   When the installer is run, compressed folders are extracted automatically, and a folder with the same name as the installer is created in the current folder. This installer is self-extracting, so no user operation is necessary.

2. Move the folder
   Change the name of the extracted folder to ndk and move it to the following folder.

[Folded destination]
C:¥Users¥<user name>¥AppData¥Local¥Android
* Any path can be used for the destination, but changing it is not recommended because it will affect other settings described below.

[Screen shot]
Building the run-time environment

Getting the source code and sample script

Download the MMDAgent source code and Sample Script from the Web site and save it in a suitable location on the PC. The download is a compressed Zip file that must be extracted before performing the following procedures.

▼MMDAgent Web site
http://www.mmdagent.jp/

▼ Procedure

The MMDAgent SourceForge page contains all the releases, instructions for SVN access, and other info.
Creating a new project
Create a new development project.

Procedure

* The above window is displayed automatically when Android Studio launches.
* Package name is used as an ID for the application, so select a name that does not overwrite any existing applications.

* The Package name entered automatically here is used in later settings.

* Compiling and running on earlier than API 14 is not guaranteed.
(9) Select Add No Activity

(10) Select Finish
Changing how projects are displayed
Change the folder display so that it shows the actual project folder structure.

▼ Procedure

1. Select Android
2. Select Project
3. Check that the display has changed
Creating a JNI folder
Create a jni folder to store the MMDAgent source code.

▼ Procedure

1. Right-click app
2. Select New
3. Select Folder
4. Select JNI Folder
5. Select Finish

* The JNI folder is created in the MMDAgent\app\src\main folder.
Importing source code
Copy the downloaded source code to the jni folder.

▼ Procedure (when using the standard explorer)

(1) Select and copy the entire contents of the folder
**Procedure (Android Studio)**

1. Right-click `jni`
2. Select Paste
3. Select OK
4. Check that all was copied to the `jni` folder

[Notes]
This includes files not used with the Android version, but they do not affect development, so we do not delete them.
Creating system and content directories on an Android terminal

For testing, an MMDAgent system directory and content directory must be created on the run-time environment Android terminal.
(When completing the application for distribution, the directory is used for storing data downloaded from the server to the application and other tasks.)

Create the system directory

Create a system directory to store the AppData directory on the Android terminal.

▼ Procedure
1. Create a transfer source folder
   Create a transfer source folder called MMDAgent in a suitable location on the PC.

2. Copy AppData to the transfer source folder.
   Copy the contents of the Release\AppData folder in the downloaded source code to the transfer source folder.

3. Copy the source folder to the run-time environment terminal.
   Copy the transfer source folder created on the PC to the run-time environment terminal. Any path can be specified as the copy destination, but here we use the “/sdcard” folder in internal storage.
   * Depending on the terminal, /sdcard may not necessarily be internal storage.

▼ Example of the transfer source folder
Creating the content directory
Create the content directory for storing the Android terminal sample script.

▼ Procedure
1. Create a transfer source folder
   Create a transfer source folder called MMDAgent_Example in a suitable location on the PC.

2. Copy Sample Script to the transfer source folder.
   Copy the contents of the downloaded Sample Script to the transfer source folder.

3. Copy the source folder to the run-time environment terminal.
   Copy the transfer source folder created on the PC to the run-time environment terminal. Any path can be specified as the copy destination, but here we use the “/sdcard” folder in internal storage.
   * Depending on the terminal, /sdcard may not necessarily be internal storage.

▼ Example of the transfer source folder
Editing files
To build the Android version of MMDAgent, some files must be edited.
Edit the files indicated below, with reference to the lists of Edit items and examples of edited files.

Edit AndroidManifest.xml
▼ File path
MMDAgent\app\src\main\AndroidManifest.xml
* There are multiple files with the same name, so be sure to edit the correct one.

▼ Edit items
1. Change the package name (Line 1)
   Change the Package attribute to the value automatically entered when creating the project.

2. Add permissions to read and write to external storage (Lines 4 and 5)
   Add permissions to read and write to external storage.
   * MMDAgent 1.6 and greater require permissions to write.

3. Add permissions to use the microphone (Line 6)
   Add permissions to use the microphone for speech recognition.

4. Add an activity tag (Lines 15 to 21)
   Enter the names of shared libraries loaded by the NativeActivity in a meta-data tag, and enter the activity main settings and settings to register it in the Android launcher in an intent-filter tag.
File contents after editing

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.mmdagent">

  <!-- add permission -->
  <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
  <uses-permission android:name="android.permission.RECORD_AUDIO" />

  <application
    android:allowBackup="true"
    android:label="@string/app_name"
    android:icon="@mipmap/ic_launcher"
    android:theme="@style/AppTheme">

    <!-- add NativeActivity -->
    <activity android:name="android.app.NativeActivity" android:label="@string/app_name">
      <meta-data android:name="android.app.lib_name" android:value="main" />
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>

  </application>
</manifest>
```

[Notes]

If including Java code when extending the Android version of MMDAgent, the android:hasCode attribute must be added to the application tag.

- **android:hasCode attribute values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>true (default)</td>
<td>Includes Java code.</td>
</tr>
<tr>
<td>false</td>
<td>Does not include Java code.</td>
</tr>
</tbody>
</table>
Edit local.properties

File path
MMDAgent\local.properties

Edit items
1. Add the Android NDK path (Line 11)
   Add the Android NDK install folder path.

File contents after editing

```bash
## This file is automatically generated by Android Studio.
#
# Do not modify this file -- YOUR CHANGES WILL BE ERASED!
#
# This file should *NOT* be checked into Version Control Systems,
# as it contains information specific to your local configuration.
#
# Location of the SDK. This is only used by Gradle.
# For customization when using a Version Control System, please read the
# header note.
sdk.dir=C:\Users\<user name>\AppData\Local\Android\sdk
ndk.dir=C:\Users\<user name>\AppData\Local\Android\ndk
```

* Modify <user name> to match your environment.
* Use “¥¥” as the folder separator.
Edit build.gradle

▼ File path
MMDAgent\app\build.gradle
* There are multiple files with the same name, so be sure to edit the correct one.

▼ Edit items
1. Change the package name (Line 9)
   Change the applicationId to the value automatically input when creating the project.

2. Disable Android Studio build (Line 16)
   To use the ndk-build command, disable Android Studio build.

3. Add a task to generate config.h (Lines 31 to 68)
   Add a task to generate the settings file for Julius (the speech recognition module).

4. Add tasks to perform the compile (Lines 70 to 79)
   Add tasks to run the ndk-build command.
   * Modify ndkDir on line 72 to match your environment.

5. Add task dependency relationships (Lines 81 to 84, 103 to 104)
   Configure the dependency relationships between tasks in this file.

6. Add a task to delete config.h (Lines 86 to 91)
   Add a task to delete the configuration file for Julius (the speech recognition module).

7. Add a task to delete ndk-build related files (Lines 93 to 101)
   Add a task to delete files generated by the ndk-build command.
   * Modify ndkDir in line 95 to match your environment.
```groovy
apply plugin: 'com.android.application'

import org.apache.tools.ant.taskdefs.condition.Os

android {
  compileSdkVersion 21
  buildToolsVersion "21.1.2"

  defaultConfig {
    applicationId "com.example.mmdagent"
    minSdkVersion 15
    targetSdkVersion 21
    versionCode 1
    versionName "1.0"
  }

  sourceSets.main.jni.srcDirs = [] // avoid using NdkCompile task

  buildTypes {
    release {
      minifyEnabled false
      proguardFiles(getDefaultProguardFile('proguard-android.txt'),
                   'proguard-rules.pro');
    }
  }

  dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:21.0.3'
  }

  task makeConfigFile << {
    def configFile1 = file("src/main/jni/Library_Julius/include/julius/config.h");
    configFile1.createNewFile()
    configFile1.write('#define JULIUS_PRODUCTNAME ""' + System.getProperty("line.separator"))
  }
```

Nagoya Institute of Technology
```
35   configFile1.append('#define JULIUS_VERSION "4.3"' + System.getProperty("line.separator"))
36   configFile1.append('#define JULIUS_SETUP "Fast"' + System.getProperty("line.separator"))
37   configFile1.append('#define JULIUS_HOSTINFO ""' + System.getProperty("line.separator"))
38   configFile1.append('#define RETSIGTYPE void' + System.getProperty("line.separator"))
39   configFile1.append('#define STDC_HEADERS 1' + System.getProperty("line.separator"))
40   configFile1.append('#define UNIGRAM_FACTORING 1' + System.getProperty("line.separator"))
41   configFile1.append('#define LOWMEM2 1' + System.getProperty("line.separator"))
42   configFile1.append('#define PASS1_IWCD 1' + System.getProperty("line.separator"))
43   configFile1.append('#define SCAN_BEAM 1' + System.getProperty("line.separator"))
44   configFile1.append('#define GPRUNE_DEFAULT_BEAM 1' + System.getProperty("line.separator"))
45   configFile1.append('#define CONFIDENCE_MEASURE 1' + System.getProperty("line.separator"))
46   configFile1.append('#define LM_FIX_DOUBLE_SCORING 1' + System.getProperty("line.separator"))
47   configFile1.append('#define GRAPHOUT_DYNAMIC 1' + System.getProperty("line.separator"))
48   configFile1.append('#define GRAPHOUT_SEARCH 1' + System.getProperty("line.separator"))
49   configFile1.append('#define HAVE_STRCASECMP 1' + System.getProperty("line.separator"))
50
51   def configFile2 = file("src/main/jni/Library_Julius/include/sent/config.h");
52   configFile2.createNewFile()
53   configFile2.write('#define LIBSENT_VERSION "4.3"' + System.getProperty("line.separator"))
54   configFile2.append('#define AUDIO_API_NAME ""' + System.getProperty("line.separator"))
55   configFile2.append('#define AUDIO_API_DESC ""' + System.getProperty("line.separator"))
56   configFile2.append('#define AUDIO_FORMAT_DESC ""' + System.getProperty("line.separator"))
57   configFile2.append('#define GZIP_READING_DESC ""' + System.getProperty("line.separator"))
58   configFile2.append('#define STDC_HEADERS 1' + System.getProperty("line.separator"))
59   configFile2.append('#define USE_MIC 1' + System.getProperty("line.separator"))
60   configFile2.append('#define USE_ADDLOG_ARRAY 1' + System.getProperty("line.separator"))
61   configFile2.append('#define HAVE_SOCKLEN_T 1' + System.getProperty("line.separator"))
62   configFile2.append('#define HAVE_UNISTD_H 1' + System.getProperty("line.separator"))
63   configFile2.append('#define HAVE_ZLIB 1' + System.getProperty("line.separator"))
64   configFile2.append('#define HAVE_STRCASECMP 1' + System.getProperty("line.separator"))
65   configFile2.append('#define USE_VM_JNI 1' + System.getProperty("line.separator"))
66   configFile2.append('#define MFCC_SINCOS Table 1' + System.getProperty("line.separator"))
67   }
68
69```
task buildNative(type: Exec) {
    def ndkDir = project.plugins.findPlugin('com.android.application').getNdkFolder()
    def ndkDir = "C:\Users\<用户名>\AppData\Local\Android\ndk"
    def jOption = '-j'+Runtime.getRuntime().availableProcessors()
    if(Os.isFamily(Os.FAMILY_WINDOWS)){
        commandLine("$ndkDir/ndk-build.cmd", jOption, '-C', file('src/main').absolutePath,
            'NDK_APP_LIBS_OUT=jniLibs');
    }else{
        commandLine("$ndkDir/ndk-build", jOption, '-C', file('src/main').absolutePath,
            'NDK_APP_LIBS_OUT=jnilibs');
    }
}

buildNative.dependsOn 'makeConfigFile'
tasks.withType(JavaCompile) {
    compileTask -> compileTask.dependsOn 'buildNative'
}

task cleanConfigFile << {
    def configFile1 = file('src/main/jni/Library_Julius/include/julius/config.h');
    configFile1.delete();
    def configFile2 = file('src/main/jni/Library_Julius/include/sent/config.h');
    configFile2.delete();
}

task cleanNative(type: Exec) {
    def ndkDir = project.plugins.findPlugin('com.android.application').getNdkFolder()
    def ndkDir = "C:\Users\<用户名>\AppData\Local\Android\ndk"
    if(Os.isFamily(Os.FAMILY_WINDOWS)){
        commandLine("$ndkDir/ndk-build.cmd", 'clean', '-C', file('src/main').absolutePath,
            'NDK_APP_LIBS_OUT=jnilibs');
    }else{
        commandLine("$ndkDir/ndk-build", 'clean', '-C', file('src/main').absolutePath,
            'NDK_APP_LIBS_OUT=jnilibs');
    }
}
<table>
<thead>
<tr>
<th></th>
<th>cleanNative.dependsOn 'cleanNative'</th>
</tr>
</thead>
<tbody>
<tr>
<td>103</td>
<td>clean.dependsOn 'cleanNative'</td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td></td>
</tr>
</tbody>
</table>
Edit Android.mk

▼ File path
MMDAgent\app\src\main\jni\Library_MMDAgent\Android.mk

* There are multiple files with the same name, so be sure to edit the correct one.

▼ Edit items
1. Change the path for DMMDAGENT_OVERWRITEEXEFILE (Line 38)
   Change it to that of the fst file in the Content directory. Note that this will be referenced as an exe file, so enter the filename extension as exe.
   
   ¥"/sdcard/MMDAgent_Example/MMDAgent_Example.exe¥"

   * If following the instructions in this manual exactly, enter the path above.

2. Change the path for DMMDAGENT_OVERWRITECONFIGFILE (Line 39)
   Change it to that of the mdf file in the Content directory.

   ¥"/sdcard/MMDAgent_Example/MMDAgent_Example.mdf¥"

   * If following the instructions in this manual exactly, enter the path above.

3. Change the path for DMMDAGENT_OVERWRITESYSDATADIR (Line 40)
   Change it to that of the AppData folder in the System directory.

   ¥"/sdcard/MMDAgent/AppData¥"

   * If following the instructions in this manual exactly, enter the path above.

4. Change the path for DMMDAGENT_OVERWRITEPLUGINDIR (Line 41)
   Change it to match the Package name of the created project.

   ¥"/data/data/<packageName>/lib¥"

   * Enter the value automatically input when creating the project in place of <packageName> above.
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE     := MMDAgent
LOCAL_SRC_FILES  := src/lib/BoneController.cpp
                  src/lib/LipSync.cpp
                  src/lib/LogText.cpp
                  src/lib/Message.cpp
                  src/lib/MMDAgent.cpp
                  src/lib/MMDAgent_utils.cpp
                  src/lib/MotionStocker.cpp
                  src/lib/Option.cpp
                  src/lib/PMDObject.cpp
                  src/lib/Plugin.cpp
                  src/lib/Render.cpp
                  src/lib/ScreenWindow.cpp
                  src/lib/Stage.cpp
                  src/lib/FreeTypeGL.cpp
                  src/lib/TileTexture.cpp
                  src/lib/Timer.cpp
LOCAL_C_INCLUDES := $(LOCAL_PATH)/include
                  $(LOCAL_PATH)/../Library_JPEG/include
                  $(LOCAL_PATH)/../Library_Bullet_Physics/include
                  $(LOCAL_PATH)/../Library_GLee/include
                  $(LOCAL_PATH)/../Library_libpng/include
                  $(LOCAL_PATH)/../Library_zlib/include
                  $(LOCAL_PATH)/../Library_MMDFiles/include
                  $(LOCAL_PATH)/../Library_GLFW/include
                  $(LOCAL_PATH)/../Library_FreeType/include
                  $(LOCAL_PATH)/../Library_UTF8-CPP/include
LOCAL_CFLAGS     += -DMMDAGENT_DONTRENDERDEBUG
                  -DMMDAGENT_DONTUSESHADOWMAP
                  -DMMDAGENT_DONTPICKMODEL
                  -DMMDAGENT_DONTUSEMOUSE
-DMMDAGENT_DONTUSEWINDOW
-DMMDAGENT

-DMMDAGENT_OVERWRITEEXEFILE="/sdcard/MMDAgent_Example/MMDAgent_Example.exe"
-DMMDAGENT_OVERWRITECONFIGFILE="/sdcard/MMDAgent_Example/MMDAgent_Example.mdf"
-DMMDAGENT_OVERWRITESYSDATADIR="/sdcard/MMDAgent/AppData"
-DMMDAGENT_OVERWRITEPLUGINDIR="/data/data/com.example.mmdagent/lib"

include $(BUILD_STATIC_LIBRARY)
Building and running source code

Build the source code and run the app on an Android terminal.

Procedure

1. Select app
2. Select
3. Select Choose a running device
4. Check that the device is selected
5. Select OK

* If no errors occur during the build, the window above will automatically appear.
If the MMD Agent is displayed on the Android device, the build was successful.