JADE TUTORIAL

JADE PROGRAMMING FOR ANDROID

USAGE RESTRICTED ACCORDING TO LICENSE AGREEMENT.

last update: 14 June 2012. JADE 4.2.0

Authors: Giovanni Caire (Telecom Italia S.p.A.)
Giovanni Iavarone (Telecom Italia S.p.A.)
Michele Izzo (Telecom Italia S.p.A.)
Kevin Heffner (PEGASUS SIMULATION)

TABLE OF CONTENTS

1 INTRODUCTION

This document describes how to create JADE-based applications running on the Android Operating System. The reader is assumed to be already familiar with both JADE, Android programming and the Eclipse development environment that is typically used to create Android applications. If this is not the case we do recommend the following documents before going ahead.

- JADE Programming tutorial (http://jade.tilab.com/doc/tutorials/JADEProgramming-Tutorial-for-beginners.pdf)
- JADE Administration guide (http://jade.tilab.com/doc/tutorials/JADEAdmin/index.html)
- Android Tutorial (http://developer.android.com/resources/tutorials/hello-world.html)

Though applications intended to run in an Android environment are fully written in Java, the application model is quite different with respect to that of normal Java applications. In order to meet Android requirements and specificities an ad-hoc version of JADE (that will be referred to as JadeAndroid) has been created. JadeAndroid is directly available for download as a jar archive in the download area of the JADE web site. People interested in modifying or recompiling it should download the JADE sources plus the LEAP Add-on and follow the instructions included in the LEAP User Guide.

In order to show the main steps that are typically necessary to create a JADE-based Android application, this tutorial makes use of a minimal Chat platform whose complete sources are available for download at http://jade.tilab.com/papers-examples.htm.

The tutorial is structured as follows:

Section 2 presents the Chat platform and shows how to install and run it;

Section 3 shows how to create and properly configure the Eclipse project where the Android Chat Client application can be built and customized;

Section 4 goes into the internals of the Android Chat Client application (other non-Android components of the Chat platform are out of the scope of this document) and highlights how to address typical issues such as activating an agent and making it interact with Android Activities.

2 INSTALLING AND RUNNING THE CHAT PLATFORM

As depicted in Figure 1 the Chat platform described in this tutorial is composed of three modules:

Android Chat Client – This is an Android application that allows participating to the chat from an Android device. It includes an Android GUI and an agent managing the interactions with other components.
Standard Chat Client – This is a normal Java application that allows participating to the chat from a PC or any device supporting a standard JVM. It includes a SWING GUI and an agent managing the interactions with other components.
Chat Server – This is the platform Main Container with an agent (called ChatManagerAgent) on top that keeps track of all agents currently connected to the chat.

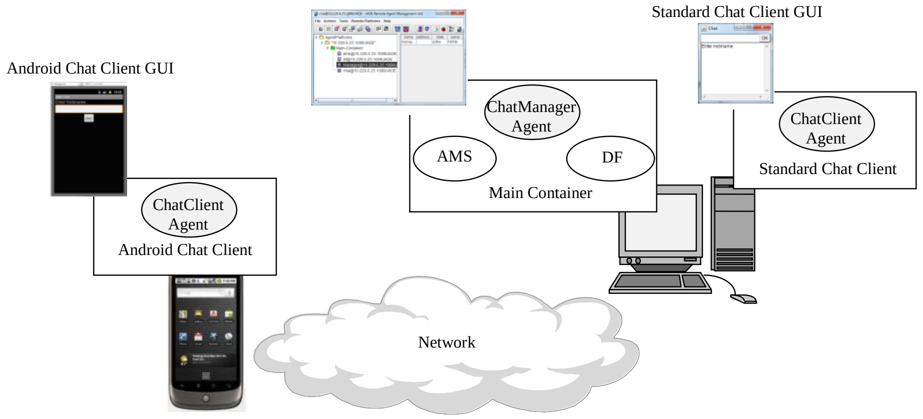


Figure 1. Chat Client-Server Platform components

2.1 Downloading the Chat platform software

The Chat platform can be downloaded from the JADE web site at http://jade.tilab.com/papers-examples.htm. The distribution consists of two archives whose structure is described below.

The ***chatStandard.zip*** file contains the following items:

```
standard
  |--build.xml                     ANT build file;
  |--bin
  |     |--startChatParticipant.bat  DOS client startup file;
  |     |--startChatParticipant.sh   Unix client startup file;
  |     |--startPlatform.bat         DOS Main Container + ChatManager startup file;
  |     |--startPlatform.sh          Unix Main Container + ChatManager startup file;
  |
  |--lib
  |     |--jade.jar                  JADE library;
  |     |--chatOntology.jar          Chat Ontology classes;
  |     |--chatStandard.jar          Standard Chat client and ChatManager classes;
  |
  |--src
       |--... (Standard Chat client and ChatManager sources)
```

The *chatOntology.jar* contains precompiled classes implementing concepts, predicates and actions that must be known by agents in the system. These classes are packaged in a separate jar file since thy must be available in all modules.
The *chatStandard.jar* file contains precompiled classes (but ontology ones) implementing the Standard Chat Client and Chat Manager Agent modules;

The ***chatAndroid.zip*** file contains the following items:

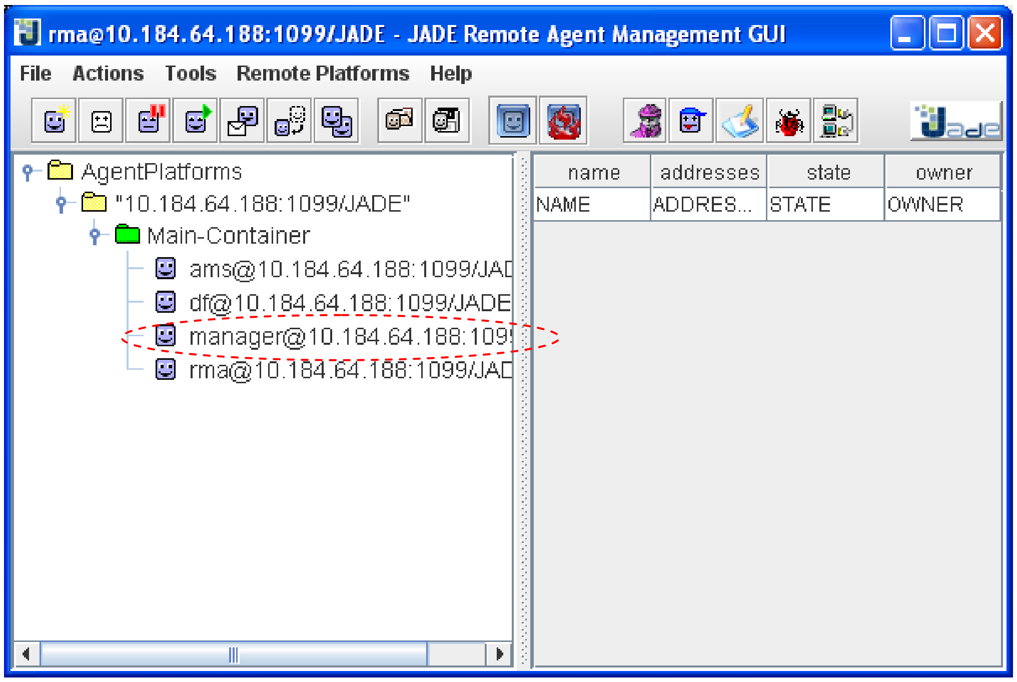
```
android
  |--build.xml                  ANT build file;
  |--build.properties           Build properties;
  |--default.properties         Android managed build properties;
  |--local.properties           Android managed configuration properties;
  |--proguard.cfg               Android managed configuration;
  |--bin
  |    |--chatClient.apk        Android Chat client application;
  |
  |--libs
  |    |--JadeAndroid.jar       JadeAndroid library;
  |    |--chatOntology.jar      Chat Ontology classes (same as in chatStandard.zip);
  |
  |--res
  |    |--... (android gui resources)
  |
  |--src
       |--... (android client sources)
```

The ***chatClient.apk*** file is the Android Chat client application ready to be installed in an Android device.

2.2 Starting the Chat Server

To launch the Chat Server (i.e. the Main Container with the Chat Manager agent on top) execute the following steps:

1. From a shell/DOS-prompt move to the folder where you decompressed the ***chatStandard.zip*** archive and then down to `standard/bin`;
2. Run the ***startPlatform.bat*** batch file for Windows or ***startPlatform.sh*** for Linux. The RMA (Jade Remote Management Agent) GUI should appear as depicted in Figure 2.


rma@10.184.64.188:1099/JADE - JADE Remote Agent Management GUI
File
Actions
Tools
Remote Platforms
Help

AgentPlatforms
"10.184.64.188:1099/JADE"
Main-Container
ams@10.184.64.188:1099/JAD
df@10.184.64.188:1099/JADE
manager@10.184.64.188:109
rma@10.184.64.188:1099/JAD

name	addresses	state	owner
NAME	ADDRES...	STATE	OWNER

Figure 2. The Main Container with the Chat Manager Agent

2.3 Starting Standard Chat clients

To populate the chat you can run one or more client agents as follows:

1. From a shell/DOS-prompt move to the folder where you decompressed the ***chatStandard.zip*** archive and then down to `standard/bin`;
2. Run the ***startChatParticipant.bat*** batch file for Windows or ***startChatParticipant.sh*** for Linux. The Standard Chat Client GUI should appear as depicted in Figure 3.
3. Insert a nickname and click OK.

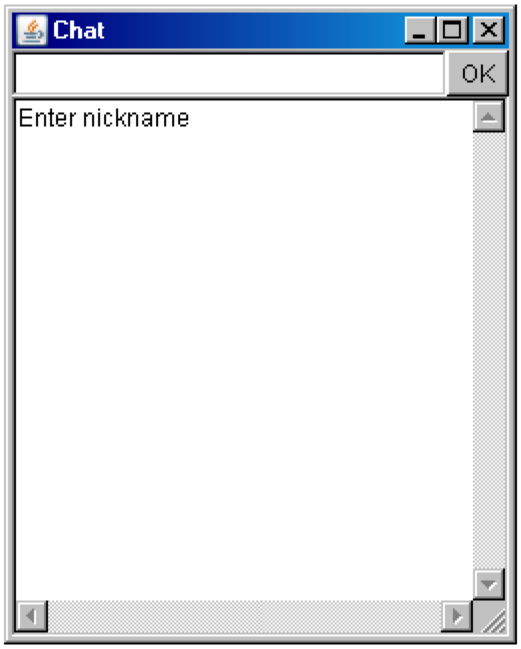


Figure 3. The Standard Chat Client GUI

2.4 Installing and Starting the Android Chat Client

To start the Chat Client on the Android emulator it is necessary to setup the Android SDK, create an AVD with **at least an API level 10 (Android 2.3.3)** and run the emulator.
Alternatively, if you have a real android device, be sure the appropriate drivers where installed and connect it to your computer via data cable (usually USB).
NOTE – If you use a real device, remember that the Android Chat Client will have to connect to the Main Container. Therefore be sure your device is configured so that it is possible to open a TCP connection with the PC where the Main Container is running over either a LAN or the Internet.
How performing the above operations is out of the scope of this tutorial. Refer to the Android documentation for more details.

At this point install the Chat Client on your Android emulator/device as follows:

1. From a shell/DOS-prompt move to the folder where you decompressed the ***chatAndroid.zip*** archive and then down to `android/bin`;
2. Run the android SDK command;
   ```
   adb install chatClient.apk
   ```

A new Application should appear on your device called ‘Jade Chat’ as shown in Figure 4.

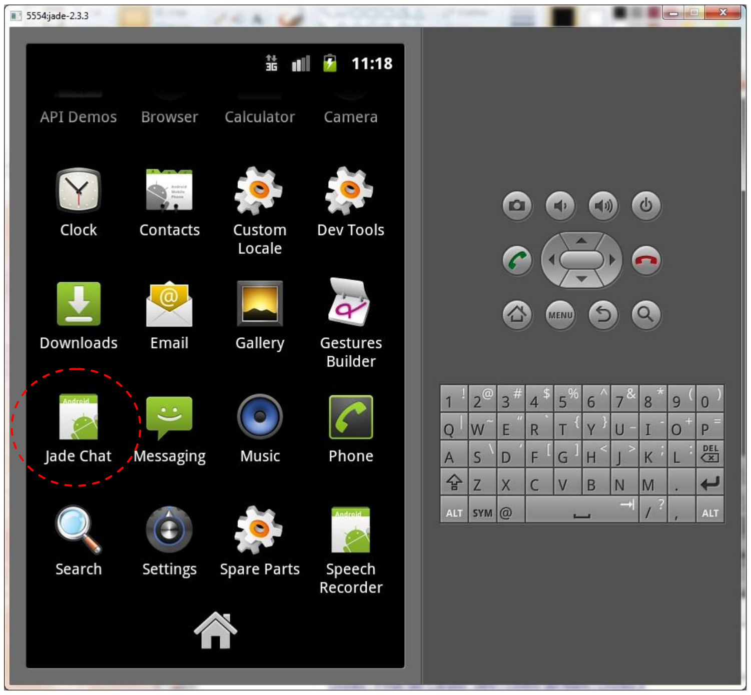


Figure 4. The applications list on the android emulator

Click on the Jade Chat icon to start the Chat Client. The screen depicted in Figure 5 should appear.

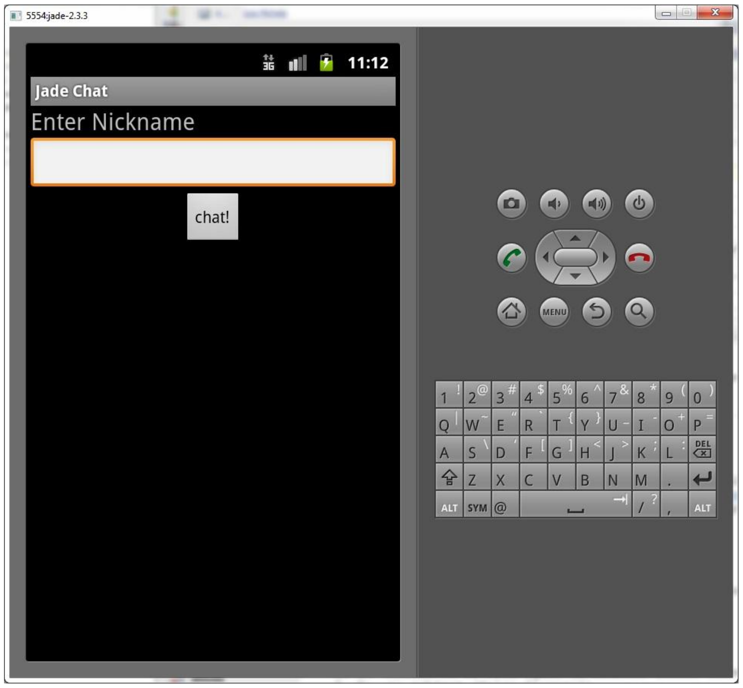
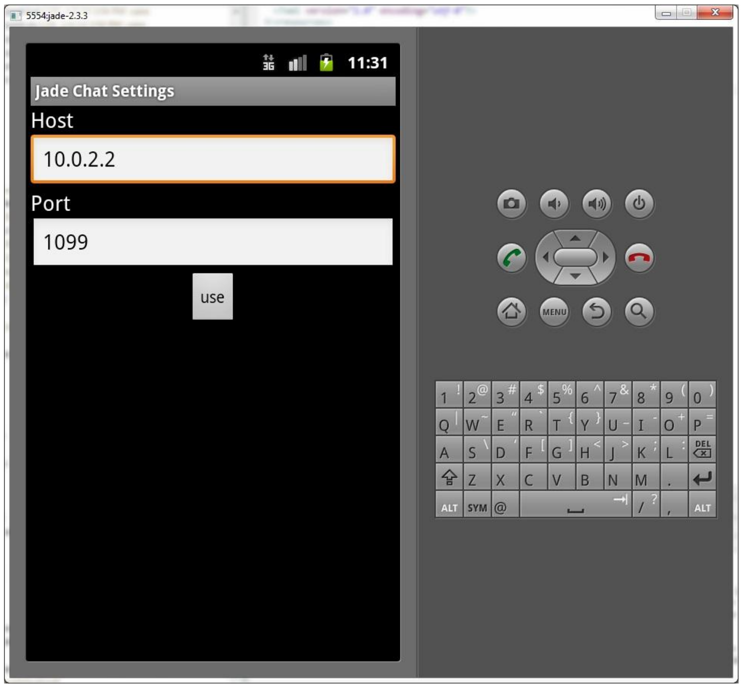


Figure 5. The Android Chat Client Login form

If you are using a real device, before entering a nickname, you typically need to setup your application with Main Container parameters (host and port). In order to do that click on the 'Menu' button and select 'settings'. Then enter the Main Container host and port asdepicted in Figure 6.



Host

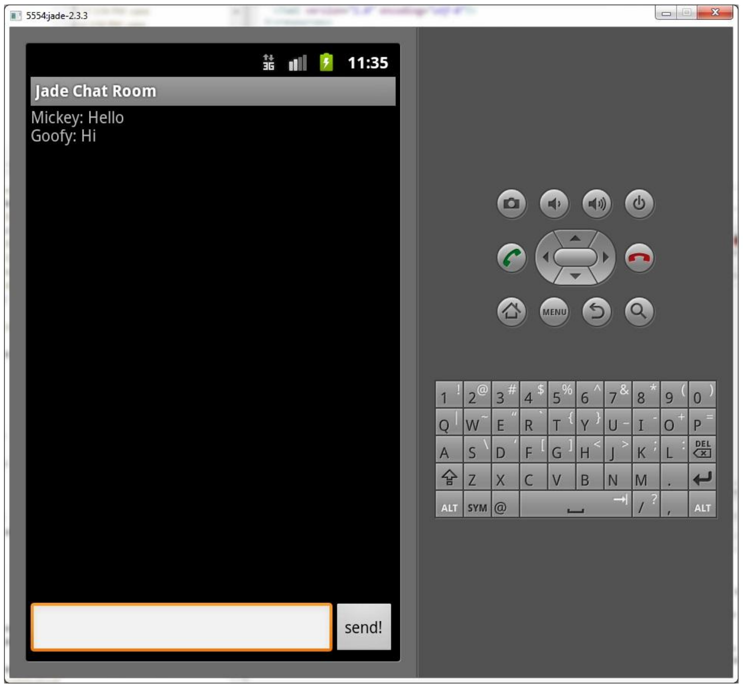
10.0.2.2

Port

1099

Figure 6. The Configuration form

Having inserted your nick name, you can start to chat.



Mickey: Hello
Goofy: Hi

Figure 7. The chat form

In order to see the list of users currently participating to the chat, click on the ‘Menu’ button and select the ‘participants’ menu item.

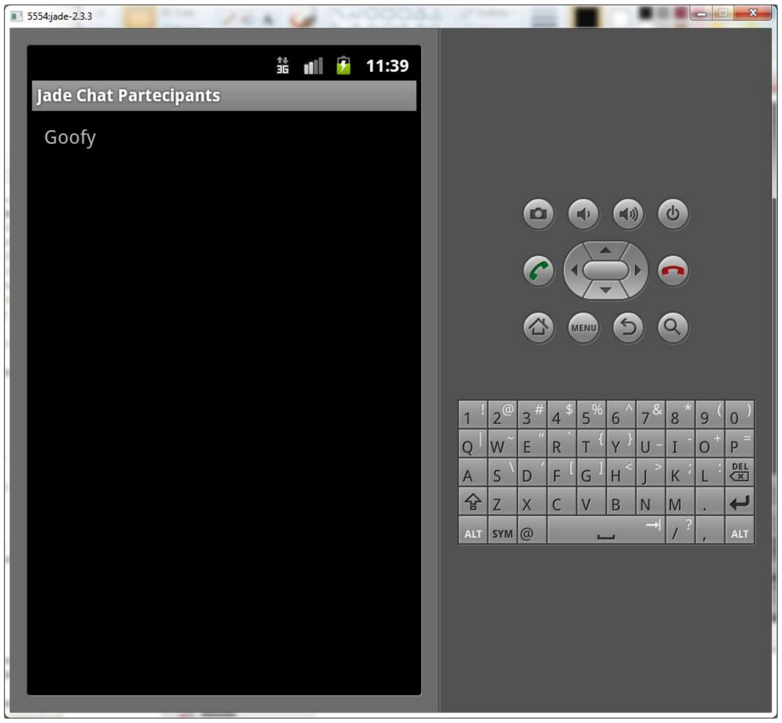


Figure 8. The participants list

3 BUILDING THE ANDROID CHAT CLIENT WITH ECLIPSE

In this section we describe how to setup the Eclipse project to compile and modify the code of the Android Chat client application. As usual we assume that the Android Developer Toolkit (ADT) plugin has already been installed into the Eclipse IDE and a target Android Virtual Device (AVD) with at least platform 2.3.3 (API 10) has been defined.

NOTE 1 - In previous section you already installed the Android Chat Client application in your Android Emulator/device. Before going on in this section it is suggested to uninstall it. This is because, since in this section the application will be recompiled in your local environment, very likely the signature of the resulting APK file will be different from that of the ***chatClient.apk*** included in the distribution package. If this is the case, the installation will fail. In order to uninstall the Android Chat Client application, perform the following steps:

1. From a DOS-shell prompt, move to the folder where you decompressed the ***chatAndroid.zip*** archive and then go to `android/bin`;
2. Run the android SDK command;
   ```
   adb uninstall chat.client.gui
   ```

NOTE 2 - Some screenshots may be slightly different depending on the versions of Eclipse and ADT. Those included in this document have been generated using Eclipse Indigo Service Release 1 and the Android Development Toolkit 14.0.0.

3.1 Creating the Android Chat client Project

In Eclipse, select **"File** > **New** > **Other... ";** The resulting dialog should have a folder labelled "Android" which should contain "Android Project" as depicted in Figure 9.

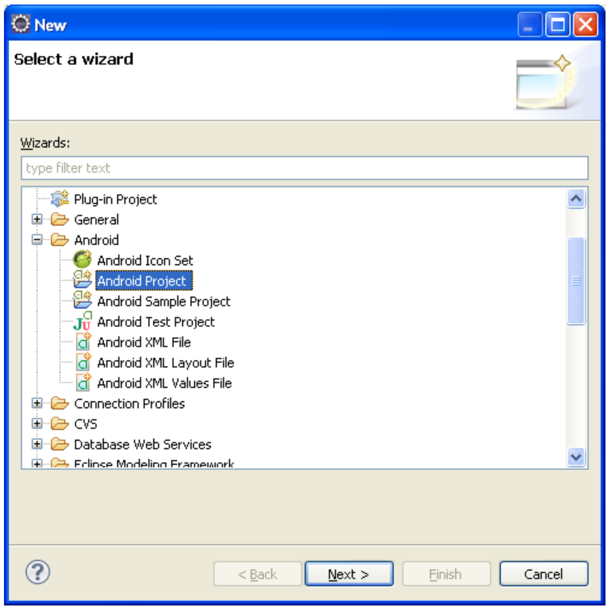
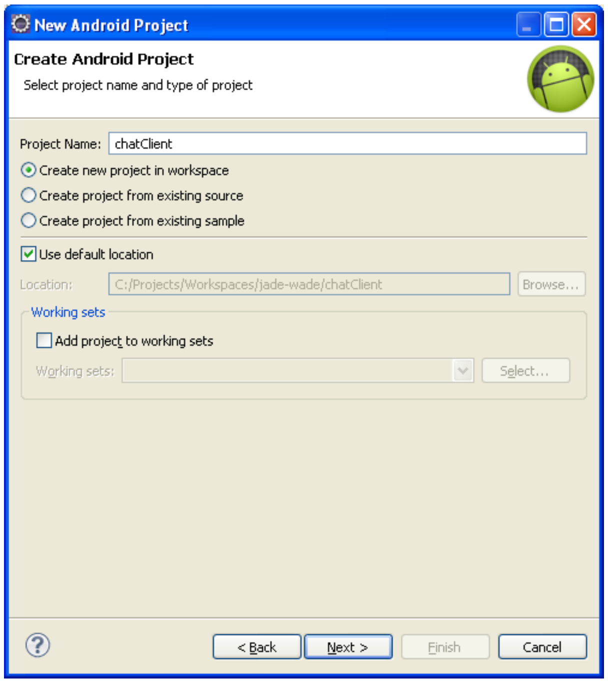


Figure 9. Creating an Android Project in Eclipse

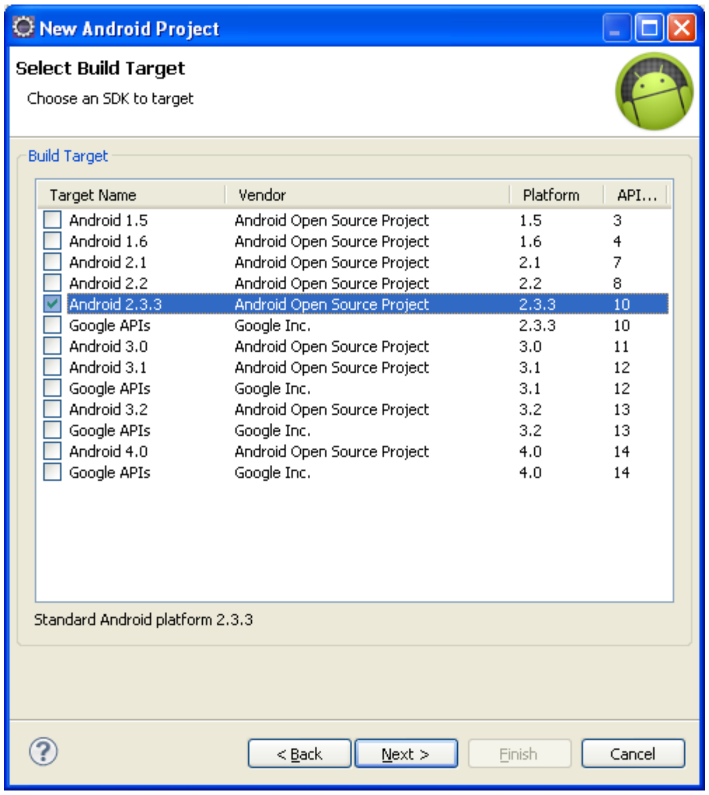
Select "**Android Project**" and click "**Next".**



Project Name: chatClient
(●) Create new project in workspace
( ) Create project from existing source
( ) Create project from existing sample
[✓] Use default location
Location: C:/Projects/Workspaces/jade-wade/chatClient
Working sets
[ ] Add project to working sets
Working sets:

Figure 10. Project general information setting step

Type "**chatClient**" as Project Name, and flag (if not yet done) the "Use default location" checkbox to create the project in your default workspace. Then click "**Next**".

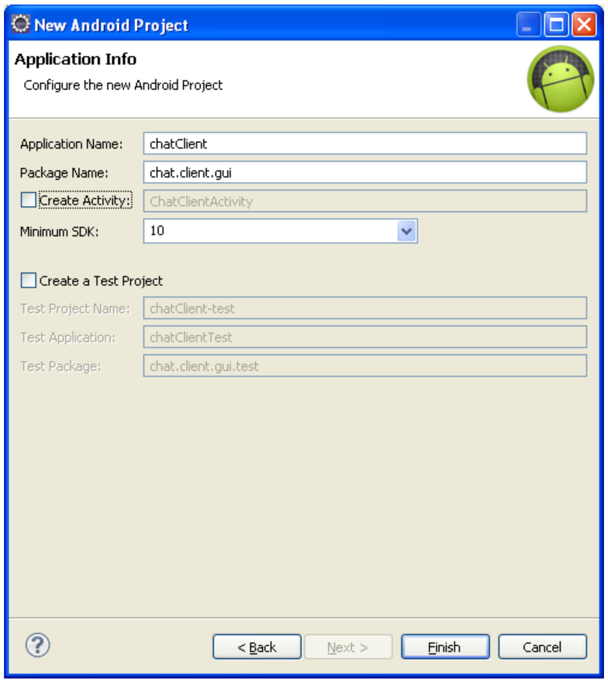


Target Name	Vendor	Platform	API...
Android 1.5	Android Open Source Project	1.5	3
Android 1.6	Android Open Source Project	1.6	4
Android 2.1	Android Open Source Project	2.1	7
Android 2.2	Android Open Source Project	2.2	8
✓ Android 2.3.3	Android Open Source Project	2.3.3	10
Google APIs	Google Inc.	2.3.3	10
Android 3.0	Android Open Source Project	3.0	11
Android 3.1	Android Open Source Project	3.1	12
Google APIs	Google Inc.	3.1	12
Android 3.2	Android Open Source Project	3.2	13
Google APIs	Google Inc.	3.2	13
Android 4.0	Android Open Source Project	4.0	14
Google APIs	Google Inc.	4.0	14

Standard Android platform 2.3.3

Figure 11. SDK selection step

Choose the "**Android 2.3.3**" platform SDK and click "**Next**".



Application Name:	chatClient
Package Name:	chat.client.gui
☐ Create Activity:	ChatClientActivity
Minimum SDK:	10
☐ Create a Test Project	
Test Project Name:	chatClient-test
Test Application:	chatClientTest
Test Package:	chat.client.gui.test

Figure 12. Application infos

Enter “**chatClient**” (again) as Application name, “**chat.client.gui**” as Package Name and choose 10 (android 2.3.3) as Minimum SDK. Unflag the “Create Activity” checkbox (since we already have all the sources available, we don’t want Eclipse to create new empty classes for us) and click “**Finish**”.

3.2 Copying Android Chat client sources in the project

In previous step we created an empty Android Chat Client project. Now we have to fill it with the sources and other resources included in the demo distribution package. To do that copy the content of the `android` folder in the ***chatAndroid.zip*** archive, to the folder `chatClient` created by Eclipse in your eclipse workspace. Overwrite all existing files.

In Eclipse refresh (press F5) your project three and verify that you get a structure similar to that depicted in Figure .

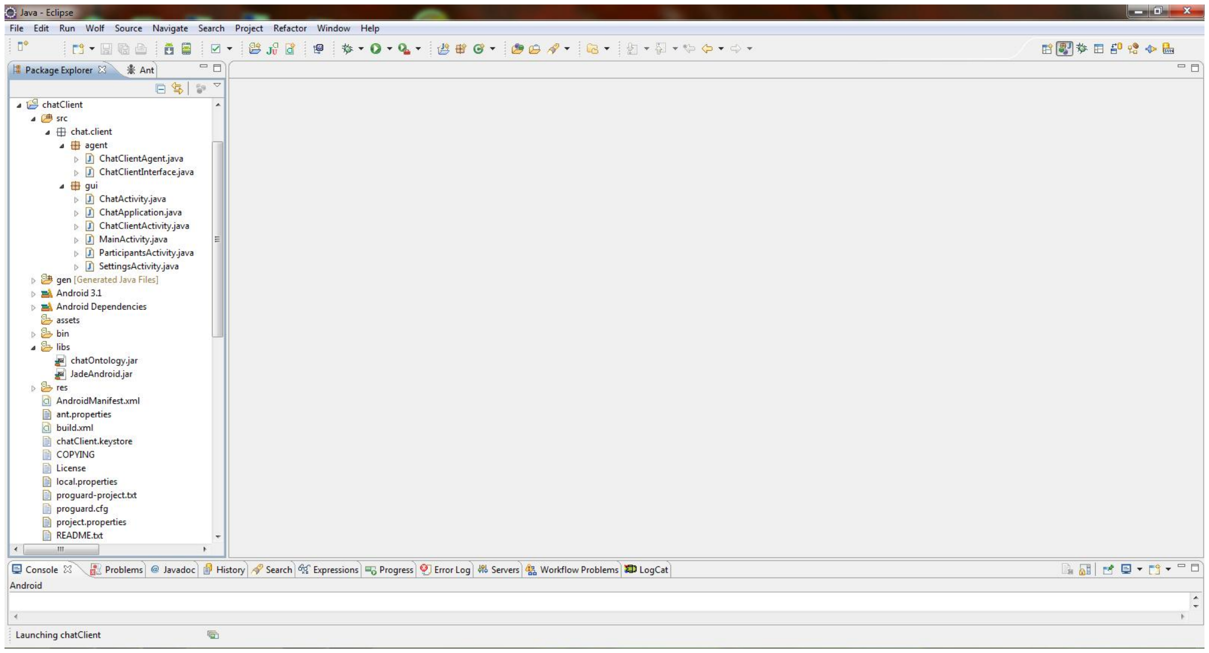


Figure 13. Android Chat Client project structure

3.3 Running the project

At this point we are ready to compile (if the Build Automatically option is selected Eclipse already did that in background) and run the Android Chat Client. First of all be sure the Chat Server is up and running on your PC. If this is not the case start it as described in section 2.2. Then, from the Eclipse Package Explorer, right-click on the chatClient project root folder and choose “**Run as**” →“**Android Application**”

4 HOW THE CHAT APPLICATION WORKS

In this section the internal workings of the Android Chat Client application are described. First the application structure briefly is presented. Then we focus on JADE-Android specific aspects and in particular on:
- ***how to start the JADE runtime and to create agents on it***;
- ***how to implement interactions between user interface classes and agents in both directions***.

On the other hand, how the Chat Client application uses the Android API and interacts with the framework is outside the scope of this tutorial.

4.1 Application structure

4.1.1 Resources

The user interface consists of four screens: the login screen, the configuration screen, the effective chat screen and an information screen that shows the list of the people currently participating to the chat. Those screens were defined in android using the following resources:

- login layout (see Figure 5);
- configuration layout (see Figure 6);
- chat layout (see Figure 7);
- participants list layout (see Figure 8);
- setting menu activated from the login layout;
- chat options menu activated from the chat layout; and
- string definition file;

Those files are located in the `res` folder of the android project according to the structure below:

```
chatClient/
 |- src/
 |    | …
 |- res/
      |- layout/
      |    |- main.xml
      |    |- chat.xml
      |    |- participant.xml
      |    |- participants.xml
      |    |- settings.xml
      |- menu/
      |    |- main_menu.xml
      |    |- chat_menu.xml
      |- values/
           |- strings.xml
```

The login screen defined in ***main.xml*** layout consists of an **EditText** component used to enter a nickname and a **Button** used to log into the chat application. To complete the screen functionalities we need to define a menu, named ***main_menu.xml*** with additional options to access the configuration screen and to exit the application.

The chat layout defined in ***chat.xml*** is composed of a **ScrollView** component with a **TextView** inside that shows the messages exchanged in the chat. In the bottom part of the layout two controls are located: an **EditText** and a **Button** to allow the user to write and send messages. From the chat screen it is possible to show the list of participants or clear the messages received by means of the actions defined in the ***chat_menu.xml***.

The participants list, located in the ***participants.xml*** layout, uses the native android layout used to show the contacts list and consists of a **ListView** inside a **LinearLayout**. Each line of the **ListView** is customized to show only the nickname of the participant. The line was customized in the ***participant.xml*** layout.

The configuration screen consists of two **TextView** used to insert the IP address and port of the Main Container. This is defined in the ***settings.xml*** layout.

All the **id**s related strings are defined in the file ***string.xml***.

4.1.2 Classes

The application code is organized in two packages called `chat.client.gui` and `chat.client.agent` according to the following structure.

```
chatClient
 |--src
 |   |--chat
 |      |--gui
 |      |    |--MainActivity.java
 |      |    |--ChatActivity.java
 |      |    |--ChatApplication.java
 |      |    |--ParticipantsActivity.java
 |      |    |--SettingsActivity.java
 |      |
 |      |--agent
 |           |--ChatClientAgent.java
 |           |--ChatClientInterface.java
 |
 |-- AndroidManifest.xml
```

The `chat.client.gui` package contains Android related classes such as the Application class and Activity classes. The `chat.client.agent` package contains the class of the Chat Client Agent and the interface that it provides to the GUI components.

As usual for Android applications the ***AndroidManifest.xml*** defines the application functionalities within the Android OS.

4.2 Starting JADE and creating agents

Consistent with the Android architecture, the JADE runtime is wrapped by an Android service. More specifically, the jadeAndroid.jar library includes two service classes `jade.android.RuntimeService` and `jade.android.MicroRuntimeService` that wrap a full container and a split container, respectively. In this tutorial we use a split container that, in general, is the suggested approach when working with mobile devices (see the LEAP User Guide for more details). The MicroRuntimeService is declared in the Android manifest as below.

```
<application ... android:name="ChatApplication">
      <service android:name="jade.android.MicroRuntimeService" />
      …
</application>
```

4.2.1 Binding to the service

The first operation to activate the JADE runtime from an Android Activity is to bind to the MicroRuntimeService. As a result a `jade.android.MicroRuntimeServiceBinder` object is retrieved such that it will be possible to perform all JADE management operations. The code below (taken from the `startChat()` method of the `chat.client.gui.MainActivity` class) shows how this is done.

```
serviceConnection = new ServiceConnection() {
      public void onServiceConnected(ComponentName className, IBinder service) {
            // Bind successful
            microRuntimeServiceBinder = (MicroRuntimeServiceBinder) service;
            ...
      };

      public void onServiceDisconnected(ComponentName className) {
            // Bind unsuccessful
            microRuntimeServiceBinder = null;
      }
};

bindService(new Intent(getApplicationContext(), MicroRuntimeService.class),
          serviceConnection,
          Context.BIND_AUTO_CREATE);
```

4.2.2 Starting a JADE split Container

Having retrieved the `MicroRuntimeServiceBinder` object it is now possible to start a JADE Split Container as shown in the code snipped below (taken from the `startContainer()` method of the `chat.client.gui.MainActivity` class).

```
Properties pp = new Properties();
pp.setProperty(Profile.MAIN_HOST, host);
pp.setProperty(Profile.MAIN_PORT, port);
pp.setProperty(Profile.JVM, Profile.ANDROID);
…
microRuntimeServiceBinder.startAgentContainer(pp,
      new RuntimeCallback<Void>() {
            @Override
            public void onSuccess(Void thisIsNull) {
                  // Split container startup successful
```

```
                ...
            }

            @Override
            public void onFailure(Throwable throwable) {
                // Split container startup error
            }
        } );
```

As usual the host and port where the Main Container is running (as well as other configuration options) must be specified in a Properties object.

According to the Android philosophy all operations are asynchronous and the result is made available by means of a `jade.android.RuntimeCallback` object.

4.2.3 Starting an Agent

Once the JADE runtime is up and running it is possible to start the Chat Client Agent on it as shown in the code snippet below (taken from the `startAgent()` method of the `chat.client.gui.MainActivity` class)

```
microRuntimeServiceBinder.startAgent(nickname,
        ChatClientAgent.class.getName(),
        new Object[] { getApplicationContext() },
        new RuntimeCallback<Void>() {
            @Override
            public void onSuccess(Void thisIsNull) {
                // Agent successfully started
                ...
            }

            @Override
            public void onFailure(Throwable throwable) {
                // Agent startup error
                ...
            }
        } );
```

Note that the ApplicationContext is passed to the agent as argument. This will allow the agent to access Android API when needed (see for instance section 4.4).

4.3 GUI components to Agent interactions

The cleanest way to implement interactions between GUI components (mainly Android Activities) and the agent is to exploit the Object-to-Agent (O2A for short) interface mechanism introduced in version 4.1.1 of JADE. This allows an agent to expose one or more interfaces that can be retrieved by external components. External components can trigger agent tasks by invoking the methods of these interfaces and thus retrieve agent information and so on according to the requirements of the application.

In the Chat Client application the Chat Client Agent exposes the `ChatClientInterface` as shown in the code snippet below:

```
public interface ChatClientInterface {
	public void handleSpoken(String s);
	public String[] getParticipantNames();
}
```

The `handleSpoken()` method is used in the `ChatActivity` to make the agent forward a chatted message to all chat participants.

The `getParticipantNames()` is used in the `ParticipantsActivity` to show the list of people currently participating to the chat.

The following line, taken from the agent `setup()` method, shows how the Chat Client Agent exposes the O2A interface described above.

```
registerO2AInterface(ChatClientInterface.class, this);
```

Similarly the following line, taken from the onCreate() method of the ChatActivity, shows how GUI components can retrieve the O2A interface exposed by the Chat Client Agent.

```
chatClientIf = MicroRuntime.getAgent(nickname).getO2AInterface(ChatClientInterface.class);
```

When the user types a message and clicks on the ***Send*** button of the ChatActivity the following code is triggered.

```
String message = messageField.getText().toString();
if (message != null && !message.equals("")) {
	chatClientIf.handleSpoken(message);
	messageField.setText("");
}
```

4.4 Agent to GUI components interactions

When working with JADE, it is often the case that an agent must proactively show some information in the GUI. In the Chat Client application this is the case, for instance, when a message chatted by another participant is received by the agent and must be shown in the `ChatActivity`. The suggested way to implement this kind of interaction is to exploit the mechanism that Android provides to allow different components to interact. This is based on broadcasting so called Intents that can be received by interested components. The code snippet below shows how the `ChatClientAgent` creates and broadcasts an Intent to notify the GUI that a message has been received.

```
Intent broadcast = new Intent();
broadcast.setAction("jade.demo.chat.REFRESH_CHAT");
broadcast.putExtra("sentence", speaker + ": " + sentence + "\n");
logger.info("Sending broadcast " + broadcast.getAction());
context.sendBroadcast(broadcast);
```

Where `context` is the field where the agent stored the ApplicationContext received as startup argument (see section 4.2.3).

The code snippets below show how the ChatActivity registers a receiver to intercept Intents carrying received sentences and handles them, respectively:

```
myReceiver = new MyReceiver();
IntentFilter refreshChatFilter = new IntentFilter();
refreshChatFilter.addAction("jade.demo.chat.REFRESH_CHAT");
registerReceiver(myReceiver, refreshChatFilter);
…

private class MyReceiver extends BroadcastReceiver {

	@Override
	public void onReceive(Context context, Intent intent) {
		String action = intent.getAction();
		if (action.equalsIgnoreCase("jade.demo.chat.REFRESH_CHAT")) {
			TextView chatField = (TextView) findViewById(R.id.chatTextView);
			chatField.append(intent.getExtras().getString("sentence"));
			scrollDown();
		}
	}
}
```

The tutorial ends here. Feel free to use the demo software and modify it as you like. Have fun!