

# SNAP Mobile: Hello World Tutorial for Java™ ME Clients

Version 1.2; June 20, 2007

SNAP Mobile

**NOKIA**

Copyright © 2006, 2007 Nokia Corporation. All rights reserved.

Nokia and Forum Nokia are trademarks or registered trademarks of Nokia Corporation. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. Other product and company names mentioned herein may be trademarks or trade names of their respective owners.

#### **Disclaimer**

The information in this document is provided “as is,” with no warranties whatsoever, including any warranty of merchantability, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. This document is provided for informational purposes only.

Nokia Corporation disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information presented in this document. Nokia Corporation does not warrant or represent that such use will not infringe such rights.

Nokia Corporation retains the right to make changes to this specification at any time, without notice.

#### **License**

A license is hereby granted to download and print a copy of this specification for personal use only. No other license to any other intellectual property rights is granted herein.

## Contents

<b>1</b>	<b>Introduction.....</b>	<b>5</b>
1.1	Audience.....	5
1.2	Documents and resources.....	5
1.3	Using this document.....	5
<b>2</b>	<b>SNAP Mobile Hello World sample application.....</b>	<b>6</b>
2.1	What does the Hello World sample include? .....	6
2.2	Running the Hello World sample .....	6
2.3	Creating the Hello World sample .....	8
<b>3</b>	<b>Addendum.....</b>	<b>11</b>
<b>4</b>	<b>Evaluate this resource.....</b>	<b>13</b>

## Change history

March 20, 2006	Version 1.0	Initial document release at <a href="http://www.forum.nokia.com">www.forum.nokia.com</a>
June 21, 2006	Version 1.1	Document updated due to replacement of SNAP Mobile Community Service Emulator with the SNAP Mobile Emulation Environment, which the SNAP Mobile Hello World sample application uses for a simulated connection to the SNAP Mobile Community Services.
June 20, 2007	Version 1.2	Updates to code samples based on changes to the <code>ServerComm</code> constructor in the SNAP Mobile Client API 1.4.1. Information about sample test applications added to the Addendum.

# 1 Introduction

This document, which is part of the SNAP Mobile Client SDK, supports developers who are using the SNAP Mobile Client API to create connected, Java™ ME (formerly Java™ 2 Platform, Micro Edition, or J2ME™) multiplayer games for mobile devices.

## 1.1 Audience

This document is meant for Java™ ME game developers who want to get started using the SNAP Mobile Client API. The API provides developers a way to add networked community features, such as multiplayer game-playing functionality, authentication, friends (or contacts) lists, messaging, player presence, ranking and event reporting, and other features to game applications. The document assumes that you are familiar with standard online game production and development principles.

## 1.2 Documents and resources

The following resources support the SNAP Mobile technology:

- **SNAP Mobile Client SDK:** Set of resources supporting SNAP Mobile game development, available at <http://www.forum.nokia.com/games/snapmobile> and through the Sun Java Wireless Toolkit. The SDK includes the SNAP Mobile Client API, Javadocs, game development requirements (including standard game requirements and registration and login requirements), a community service emulator, sample applications with source code and documentation, a game developer's guide, and other resources. (See also <http://snapmobile.nokia.com> for other important information and technical documentation on SNAP Mobile.)
- **Sun Java Wireless Toolkit:** The most current version of the toolkit includes a SNAP Mobile sample application with source code and Javadocs. The toolkit is available at <http://java.sun.com/products/sjwtoolkit/>.

## 1.3 Using this document

This document is divided into the following chapters:

- Chapter 2, "SNAP Mobile Hello World sample application," provides instructions for building and running the SNAP Mobile Hello World sample application.
- Chapter 3, "Addendum," points to other common uses of the SNAP Mobile Client API.

## 2 SNAP Mobile Hello World sample application

This chapter provides an introduction to the SNAP Mobile Client API using the SNAP Mobile Hello World sample application (Hello World sample). The Hello World sample is a simple application that provides code for logging in to the SNAP Mobile Community Services (community services) and retrieving a self-addressed “Hello World” message from them. This chapter explains how to run the Hello World sample while connected to the SNAP Mobile Emulation Environment.

### 2.1 What does the Hello World sample include?

The SNAP Mobile Client SDK provides the Hello World sample in its `/samples/SampleApps` directory. The Hello World sample includes the following files:

- `helloworld.jad`, `HelloWorld.jar`

The JAD and JAR files for the Hello World sample.

- `HelloWorld.class`, `Console.class`

The compiled source files for the Hello World sample.

- `HelloWorld.java`, `Console.java`

The Java source files for the Hello World sample. `HelloWorld.java` provides the logic for the Hello World sample’s core functionality: Logging in to the community services, passing a self-addressed “Hello World” message to the community services, waiting for its event listener to receive the message, and logging out. `Console.java` provides the sample’s display logic by implementing the public interface for all messages that the Hello World sample generates.

### 2.2 Running the Hello World sample

This section provides a procedure for running the Hello World sample on the Sun Java Wireless Toolkit while using the SNAP Mobile Emulation Environment to simulate networked communication to the community services. It also introduces you to SNAP Mobile attributes in the JAD file for the Hello World sample application.

To run the Hello World sample:

1. Launch the SNAP Mobile Emulation Environment (`sm-emu.jar` in the `/tools/ServerEmulator` directory) and check for the Hello World GCID configuration:

```
49152
```

**Note:** The JAR file is executable, so you can double-click it to launch the emulation environment. See “Launching the emulation environment” in [SNAP Mobile: Emulation Environment User’s Guide](#) for additional assistance with this step.

2. Start the emulation environment.
3. Use a text editor to open the `helloworld.jad` file.  
`helloworld.jad` is in the `/samples/sampleApps` directory.
4. Check for the following attribute configurations in the `helloworld.jad` file:

```
SNAP-Mobile-Host: localhost
SNAP-Mobile-OperatorID: 012345
SNAP-Mobile-SKU: 1
```

```
username: test1
password: test
```

See [SNAP Mobile: Game Developer's Guide](#) for more information on required SNAP Mobile attributes.

**Note:** For convenience during the initial game development process, sample applications often retrieve the values of user name and password attributes from the JAD file. However, in no case shall these attributes appear in JAD files that game developers submit for SNAP Mobile game certification. Instead, developers must allow users to create their own user names and passwords through standard in-game registration and login API calls that are specified in [SNAP Mobile: Registration and Login Guidelines](#). In addition, developers must specify the game class ID in the game code.

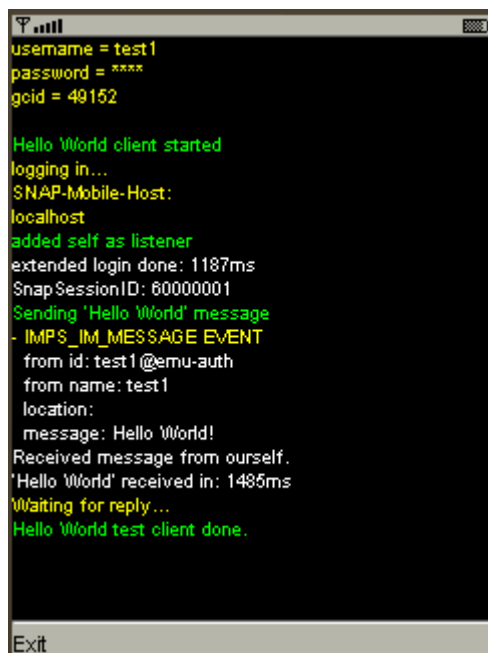
5. Double-click `helloworld.jad` to open it in your client emulator (for example, the Sun Java Wireless Toolkit client emulator).

As a guide to completing this step, you can refer to “Launching the emulation environment” in the [SNAP Mobile: Emulation Environment User's Guide](#).

**Note:** You can download the latest version of the Sun Java Wireless Toolkit from <http://java.sun.com/products/sjwtoolkit/>.

6. Run the Hello World sample in the client emulator.

If this step is successful, the emulator should look like Figure 1.



```

username = test1
password = ****
goid = 49152

Hello World client started
logging in...
SNAP-Mobile-Host:
localhost
added self as listener
extended login done: 1187ms
SnapSessionID: 60000001
Sending 'Hello World' message
- IMPS_IM_MESSAGE EVENT
  from id: test1@emu-auth
  from name: test1
  location:
  message: Hello World!
Received message from ourself.
'Hello World' received in: 1485ms
Waiting for reply...
Hello World test client done.

Exit

```

Figure 1: Hello World sample running in a client emulator

## 2.3 Creating the Hello World sample

This section walks you through the key steps used to create the Hello World sample. The Hello World sample models best coding practices for a simple SNAP Mobile application that logs in to the community services, sends and then listens for the “Hello World” message to itself, and logs out.

The Hello World sample implements the `SNAPEventListener` class to receive all incoming event and error messages, in particular to receive the self-addressed “Hello World” message. The sample also initiates a thread by implementing the standard Java ME `Runnable` interface.

**Note:** Implementing the SNAP event listener is a universal SNAP Mobile game (or application) development practice. SNAP Mobile applications periodically poll for asynchronous events, including new messages from friends, notifications about the status of game rooms and lobbies and the users who are in them, updates to the online state (or presence) of friends (Online, Offline, Busy), and so on. The SNAP Mobile gateway notifies the event listener of such events on a regular basis (whenever the listener polls for events from the community services). The SNAP Mobile Emulation Environment simulates the behavior of the SNAP Mobile gateway and the community services.

The steps that follow highlight important uses of the SNAP Mobile Client API in bold font. The code samples come from `HelloWorld.java` and serve to illustrate the main functionality of the Hello World sample. The code samples rely on the Java source files for the Hello World sample and do not work without them.

1. Initialize the Hello World sample and add the sample to itself as a SNAP event listener:

```
// Initialize ServerComm and add self as SnapEventListener
comm = new ServerComm();
comm.addSnapEventListener(this);
console.println("added self as listener", Console.GREEN);
```

The `ServerComm` class facilitates communication between a SNAP Mobile game client and the community services.

The Hello World sample adds itself as a SNAP event listener so that it can receive all messages and errors from the community services. Use `addSnapEventListener` to begin asynchronous event retrieval. The SNAP Mobile gateway notifies the event listener of community service events on a regular basis (after getting all events). See the chapter “Polling for Asynchronous Events” in [SNAP Mobile: Game Developer’s Guide](#) for more detailed information.

The Hello World sample prints the success notification in green (see also, `Console.java`).

2. Log in to the community services:

```
// Log in using extendedLogin()
then = System.currentTimeMillis();
ItemList il = comm.extendedLogin(username, password, gameClassID, null);
loggedIn = true;
done = false;
delta = (int)(System.currentTimeMillis() - then);
console.println("extended login done: " + delta + "ms", Console.WHITE);
console.println("SnapSessionID: " + il.getItem("snapSessionID"),
Console.WHITE);
```

The Hello World sample uses `System.currentTimeMillis()` to calculate the login period in milliseconds.

The Hello World sample uses the `extendedLogin` method to perform a unified login to all the community services: SNAP Game, IMPS, and Web services. In addition to logging in to the community services, the `extendedLogin` request retrieves the buddy list (`getBuddyList`) and sets presence data (`setPresence`), all in a single request.

The Hello World sample prints the login period and the SNAP game session ID in white (see also, `Console.java`).

**Note:** The IMPS services use presence attributes to notify friends (members of the client's friends list) of changes in the client's status; for example, whether the client is available for text messaging, unavailable, and so on. The implicit call to `getBuddyList` at login is necessary for the client to receive the online status of friends in the user's friends list. For example, `friend1` and `friend3` may be online, while `friend2` may not be online when the user logs in. The status of these friends may change while the user is online. For example, `friend1` might log out, and `friend2` might log in. IMPS service event 307 (`IMPS_IM_PRESENCE`) returns the online presence of the user's friends list and notifies the client of changes to a friend's online presence. For more information, see [SNAP Mobile: Game Developer's Guide](#).

3. Send yourself a "Hello World" message:

```
// Send "Hello World" buddy message to yourself.
console.println("Sending 'Hello World' message", Console.GREEN);
start = System.currentTimeMillis();
comm.sendBuddyMessage(username, "Hello World!");
```

The Hello World sample prints an `info` message to the client application or emulator indicating that the sample is sending the "Hello World" message to the community services.

Then the Hello World sample uses the `sendBuddyMessage` method to send a self-addressed "Hello World" message.

4. Wait to receive the "Hello World" message:

```
// Wait until message is received.
console.println("Waiting for reply... ", Console.YELLOW);
try {Thread.sleep(3000);} catch (InterruptedException e) {}
} catch (Exception e) {
    console.println("Error (middle)!: " + e.getMessage(), Console.RED);
}
```

Prior to logging in, the Hello World sample added an event listener to receive the "Hello World" message from the community services. The Hello World sample uses the `processEvents` method to receive the incoming message from the community services. (See `HelloWorld.java` for the implementation of the `processEvents` method. See the Javadocs for a specification of it.)

The Hello World sample then prints the "Hello World" message with the calculated round trip time (in milliseconds).

5. Catch exceptions:

```
catch (Exception e) {
    console.println("Error (end)!: " + e.getMessage(), Console.RED);
}
```

At this point, the Hello World sample catches all exceptions within its `try` statement. The Hello World sample prints error strings in red font (see also, `Console.java`). See [SNAP Mobile: Game Developer's Guide](#) for guidelines on handling errors.

6. Remove the Hello World sample from the event listener:

```
// First remove self from SnapEventListener callbacks.

comm.removeSnapEventListener(this);
```

**Note:** All SNAP Mobile games and applications must call `removeSnapEventListener` before logging out. Failure to stop the listener can lead to a session time-out error.

## 7. Log out:

```
// Then log out.
```

```
comm.unifiedLogout();
```

The Hello World sample uses the `unifiedLogout` method to log out of the community services and notifies the client that the instance of the Hello World sample is done.

### 3 Addendum

The SNAP Mobile SDK includes additional test samples that demonstrate important game functionality, including game data packet transmission within game rooms. Game data can represent moves in a game and other events that occur during a gameplay session.

SNAP Mobile games also allow users to send messages to their friends. Users locate their friends in their game's Friends lists. To determine whether it is okay for a user to send messages to a friend, it is necessary for games to process the extended presence updates (`IMPS_IM_PRESENCE` Event) that the client receives. These updates allow you to determine whether to display the online state of friends as Available, Busy, or Offline in the user's Friends list. To get the specific rules that apply to the display of online states in a game, see [SNAP Mobile: Game Developer's Guide](#).

Feature	Resource files			
	JAD files	JAR files	Java files	Class files
Sending game packets to a client	gamepacket.jad gamepacket2.jad	gamepacket.jar gamepacket2.jar	GamePacket.java Console.java	GamePacket.class Console.class
Messaging friends	buddychat.jad buddychat2.jad	buddychat.jar buddychat2.jar	BuddyChat.java Console.java	BuddyChat.class Console.class
Receiving extended presence updates	presence.jad presence2.jad	presence.jar presence2.jar	Presence.java Console.java	Presence.class Console.class

Table 1: SNAP Mobile test samples

**Note:** Refer to the Java files to understand the coding practices that the samples follow. Refer to the [SNAP Mobile: Game Developer's Guide](#) and the Javadocs to understand underlying concepts.

To launch the additional samples:

1. Set the GCID of the sample in the emulation environment.

GCID	Game Name
49152	Hello World
49721	Sample Game: Maze Racer
32270	GamePacket
16111	BuddyChat
17513	Presence

Figure 2: GCID settings in the emulation environment

2. Start the emulation environment.
3. Launch the JAD files: `*.jad` and `*2.jad` in your client emulator.

For example, launch `gamepacket.jad` and `gamepacket2.jad` to transfer game packets between two users (`test1` and `test2`) automatically. Figure 3 provides an example that runs these samples.

```

Screenshot 1 (test1):
username = test1
password = ****
gcid = 32270

GamePacket client started
logging in...
SNAP-Mobile-Host:
localhost
added self as listener
extended login done: 1234ms
SnapSessionID: 60000024
SnapUserID: 536870914
Start game...
Game started: 1157 ms.
Wait for follower to join game room
Sending packet #0
Sending packet #1
Sending packet #2
- GAME PACKET -
  from id: 536870915
  message: Packet: 0
Sending packet #3

Screenshot 2 (test2):
username = test2
password = ****
gcid = 32270

GamePacket client started
logging in...
SNAP-Mobile-Host:
localhost
added self as listener
extended login done: 1547ms
SnapSessionID: 60000025
SnapUserID: 536870915
Wait for leader to create game room
Start game...
Game started: 1126 ms.
Sending packet #0
Sending packet #1
- GAME PACKET -
  from id: 536870914
  message: Packet: 1
Sending packet #2

Screenshot 3 (test1):
message: Packet: 5
Sending packet #7
- GAME PACKET -
  from id: 536870915
  message: Packet: 6
Sending packet #8
Sending packet #9
- GAME PACKET -
  from id: 536870915
  message: Packet: 7
- GAME PACKET -
  from id: 536870915
  message: Packet: 8
Waiting for reply...
- GAME PACKET -
  from id: 536870915
  message: Packet: 9
All messages received: 4656ms
Waiting for reply...
Waiting for players to stop... (10 seconds)
First Packet: #0
Last Packet: #9
Missing: none.
GamePacket test client done.

Screenshot 4 (test2):
- GAME PACKET -
  from id: 536870914
  message: Packet: 6
Sending packet #6
- GAME PACKET -
  from id: 536870914
  message: Packet: 7
Sending packet #7
Sending packet #8
- GAME PACKET -
  from id: 536870914
  message: Packet: 8
- GAME PACKET -
  from id: 536870914
  message: Packet: 9
Sending packet #9
Waiting for reply...
All messages received: 4172ms
Waiting for reply...
Waiting for players to stop... (10 seconds)
First Packet: #1
Last Packet: #9
Missing: none.
GamePacket test client done.

```

Figure 3: Game packet transmission between clients (test1 and test2)

Before sending a game packet, each client sets up an event listener and then logs in. After logging in, one client (the follower) waits for the other client (the leader) to create a game room. The game begins, and the clients send ten game packets to one another. Then the clients check for missing packets and report the results.

## 4 Evaluate this resource

Please spare a moment to help us improve documentation quality and recognize the resources you find most valuable, by [rating this resource](#).