Series 60 Developer Platform 2.0: Guidelines For Testing J2ME™ Applications

Version 1.0
March 8, 2004

SERIES 60 PLATFORM

Legal Notice

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. Furthermore, information provided in this document is preliminary, and may be changed substantially prior to final release. 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

Change History

March 8, 2004	1.0	Initial document release.

1. Introduction to Guidelines For Testing J2ME™ Applications

Testing is an important part of application development. This document lists fundamental issues that must be taken into account when testing J2ME MIDP applications (MIDlets) designed for Series 60 Developer Platform 2.0. However, many of the issues presented in this document are useful for all MIDlets in general. Because of the style of this document, it can also be used as a guideline when designing and developing applications for Series 60 Developer Platform 2.0.

It is important to test both the functionality and the usability of an application to ensure that the application is easy and efficient to use, and keeps end user satisfied.

The document goes through fundamental, functionality, and usability requirements of MIDlets. It does not explain software testing techniques and does not give any test cases. Please refer, for example, to literature on software testing that is available in the market if you need more information about software testing in general.

You should also note that there might be some special issues in Series 60 Developer Platform 2.0 devices. For this reason, please read the possible Known Issues or other documents carefully and test the special issues correspondingly.

For more information on efficient programming and usability guidelines, refer to, for example, www.forum.nokia.com/series60, www.forum.nokia.com/testing, and www.forum.nokia.com/java.

2. Fundamental Requirements

2.1 Non-technical Issues

Application testing is more than technical testing. It is important that all general non-technical issues are in condition as well. For example, the Intellectual Property Rights (IPR) must not be violated and the form and the content of the application must be generally accepted, that is, it is not considered offensive.

Check that:

- Intellectual Property Rights (IPR) are ok.
- The content of the application is acceptable.

2.2 Application Performance and Reliability

Devices that are compliant with Series 60 Developer Platform 2.0 are capable of running quite heavy applications. However, developers must keep in mind that the speed of the application must not compromise the use and purpose of the application. This issue must be considered already in the early phases of application design. Depending on the type of the application, developers must also keep in mind that end users may use other applications concurrently. Therefore excessive consumption of the device's running power should be avoided.

Make sure that the application does not cause any harm to its user, the system's applications, or the data stored in the system. J2ME MIDP is a secure platform and creating virus applications for it is hardly possible. However, applications that work harmfully for the user can accidentally be created. For example, an application works harmfully if it unnecessarily consumes a lot of the device's processing power or fills up the device's flash memory. Purposeless network communication must be avoided as well because it may cause the user unnecessary expenses. Therefore, optimize the application by finding and correcting possible bottlenecks in the code, saving only necessary data to the device's flash memory and keeping the communication outside the device reasonable.

Check that:

- The application speed does not compromise the use and purpose of the application.
- The application does not consume the device's running power excessively.
- The application does not cause any harm to the user, other applications or data.
- There are no bottlenecks in code.
- Only necessary data is saved into the flash memory.
- Communication to and from the device is kept within reasonable limits.

2.3 Application Security

A mobile phone is carried everywhere by its owner. Because of this there is always a risk of losing the phone. Therefore, do not store any sensible data into mobile device without

encryption. For example, the application must not save the credit card number to the phone's memory. However, if the nature of the application requires saving sensible data, the data must be encrypted and hidden behind a password. In addition, the user must be informed clearly that sensible data is stored into the phone's memory.

The application must not echo the input of sensitive data, for example, pins and passwords. However, the chosen character may appear briefly in order to confirm which character has been entered; however, the character must then be masked. This can be done, for example, by using the TextField class. Set the value of the input constraints parameter to TextField.PASSWORD as the following example shows:

```
TextField textField = new TextField("Password", "", 10,
TextField.PASSWORD);
```

Possible communication of the device must be secure if sensitive information is transmitted between two end points or when a user may be charged for the result or consequences of a transaction. In such situations use secure HTTP (HTTPS) or secure socket connection. Both are supported in Series 60 Developer Platform 2.0.

Encryption is also needed if sensible data is transmitted via Bluetooth or Short Message Service (SMS). The used Bluetooth stack provides security mechanisms and the Bluetooth implementation enables and disables security mechanisms. Encryption of SMS messages must be implemented by the developer.

Check that:

- Sensible data is encrypted before it is stored into the device.
- Accessing sensible data requires a password.
- The user is informed when sensible data is stored into the device's memory.
- HTTPS or secure socket is used for secure communication.
- SMS messaging and the Bluetooth connection are protected if needed.

2.4 Installation

For good user experience it is important that the application installation works smoothly and easily. Be sure that at least all mandatory attributes and their values in the Java Application Descriptor (JAD) file are typed correctly and that they are in the right format. The attributes and values of the Java Archive (JAR) package's manifest file must be correct as well. The best way to ensure that is to use a packaging tool, for example, Nokia Developer's Suite for J2ME™ or Sun Wireless Toolkit. With these tools it is easy to set and update attributes and values into both the JAD and manifest files.

Because of the requirements of the MIDP 1.0 and MIDP 2.0 specifications the application descriptor must contain at least the following attributes:

- MIDlet-Name.
- MIDlet-Version.
- MIDlet-Vendor.
- MIDlet-Jar-URL.
- MIDlet-Jar-Size.

Check also that the details reported in the JAD file are consistent with the information reported within the application (for example, the information shown in the application's About session). The best way to ensure this is to request the `getAppProperty()`

method of the MIDlet class when this kind of information is needed. The `getAppProperty()` method retrieves the requested information from the JAD file or the JAR package's manifest file. For example, `getAppProperty("MIDlet-Version")` returns the version of the MIDlet.

Check that:

- All mandatory attributes and their values in the JAD file and the JAR package are correct.
- The application descriptor contains the attributes specified in the MIDP 1.0 and MIDP 2.0 specifications.
- Data reported in the JAD file is consistent with the information reported within the application.

3. Functionality Testing

Be sure that all functions work as they should. Test also all possible extraordinary events and use cases during the application run.

3.1 Environmental Changes

A mobile phone is a device that is used in different environments and situations. Because of this, MIDlets must work without errors if the environment changes rapidly. However, if errors occur, they must be handled as well.

Basically, it is extremely important that all exceptions are carefully handled in the code and the MIDlet run is guaranteed regardless of the exception. Ensure that the user is well informed in case of extraordinary situations. Also note that any error messages in the application must be clearly understandable. Error messages must clearly explain the nature of the problem, and, where appropriate, indicate what action needs to be taken.

Check that:

- MIDlets work correctly even though the environment may change.
- If errors occur, they are handled as well.
- The MIDlet runs even though exceptions are handled at the same time.
- The user is informed in extraordinary situations.

3.2 Networking

Interruption of network connection during MIDlet run must be considered and treated appropriately. Check that your MIDlet handles possible network errors and informs the user about them. The following code example shows how to get a notification of a possible error in the network connection and how to inform the user about it.

```
public void connect()
{

  try {
    createConnection(url);
  }
  catch (IOException ioe) {
//Handle possible error here. Inform user (for example, by showing
//Alert message) and/or try to establish the connection again.

  }

}


void createConnection(String url) throws IOException
{
    safeToExit = false;
    try {
      connection = (StreamConnection) Connector.open(url);
      stream = connection.openInputStream();
      //Handle the stream content
      }
    }
    finally {
      if (stream != null) {
```

```
        stream.close();
      }
      if (connection != null) {
        connection.close();
      }
      safeToExit = true;
    }
}
```

The application must also handle situations where connection is not allowed and close the connection after the session is over.

The system usually closes network connections when the application is about to close. However, it is recommended that the network connection is either disconnected by the user or programmatically before the application closes.

Check that:

- Network exceptions are handled and the user is notified about them.
- The network connection is closed when it is not needed anymore and/or the application is closing.

3.3 Wireless Messaging

Wireless Messaging API enables sending Short Message Service (SMS) messages. Verify that the MIDlet successfully sends the SMS message. Be sure that the application gives the user an accurate and appropriate error message if the mobile device is unable to send the SMS message because of external factors (for example, network connectivity).

Check that the message is formatted appropriately if it is sent to the mobile device's mailbox. If the message is sent to an application, verify also that the SMS message is received and processed correctly by the receiving application.

Check that:

- The MIDlet sends SMS messages correctly.
- The user gets a clear error message if something goes wrong.
- The message has an appropriate format and that it is received and processed correctly by the application.

3.4 Bluetooth Connectivity

Bluetooth device and service discoveries may take a while to complete. Therefore it is advisable to optimize these procedures. Also the user interface should be logical for good user experience.

In device discovery, use Limited Inquiry Access Code (LIAC) instead of General Inquiry Access Code (GIAC) whenever possible. With LIAC inquiry, the application finds only the devices that are in Limited-Discoverable mode thus speeding up the procedure. Using LIAC is especially feasible when discovering counter part phone(s) that are waiting for your connection, for example, to start a game.

You can also optimize the discovery procedures by stopping the discovery after the first match(es). In service discovery, for example, you can stop the search after you have found the first device(s) running the desired service of the application.

In idle mode, Bluetooth does not significantly increase the power consumption. However, when you have an active Bluetooth connection, it is worth considering to disconnect the link whenever no data is transferred for a long time to decrease power consumption. When the connection is reopened, use the `retrieveDevices()` method of the DiscoveryAgent to return an array of Bluetooth devices that have either been found by the local device during previous inquiry requests or have been specified as a pre-known device depending on the argument of the method.

A Bluetooth connection may face some interruptions as well, for example, if a device disappears for some reason from the Bluetooth session. Please note that the application is responsible for possible recovery actions, for example, for reconnecting the link, for continuing to work after one device has left the session, and for notifying the users about the situation.

The system usually closes connections when the application is about to close. However, it is recommended that the network connection is either disconnected by the user or programmatically before the application closes.

Check that:

- Device and service discoveries are efficient.
- The connection is turned off if it is not needed for a long time.
- Other devices continue the Bluetooth session even though one device would disappear from the session.
- The user is informed in case of interruptions.

3.5 Record Management System (RMS)

As stated before, save only necessary data into the device's flash memory. Optimize the data, if possible, and use packing algorithms if a lot of data must be saved. However, compare the amount of extra code for packing algorithms with the benefit of the size of packed data.

Remember to inform the user if overwriting a significant entry in the database is irreversible.

It is important to handle exceptions and be prepared for environmental changes when saving data to the flash memory of the device through Record Management System (RMS). Series 60 devices do not have a strict limit for MIDlet's RMS space, and basically MIDlets can save as much data to the flash memory as there is free memory available. However, remember that another application (Symbian or Java) may have consumed all the flash memory – in that case the system throws a `RecordStoreFullException`.

It is also possible that an extraordinary environment change takes place while saving or reading data to/from the RMS. For example, if the device's battery gets empty while saving data, it is possible that the data is not saved into the flash memory or the saved data is corrupted. Make sure that the application works without errors even if the saved data includes old information or is broken.

Note that if an application is updated, the RMS data of the older version may remain in the flash memory of the device. Check that the existing data is compatible with the newer version or that it is deleted programmatically.

Check that:

- The device's flash memory is not filled with unnecessary data.

- The user is informed if important data is about to be overwritten in the database.
- The application is prepared for exceptions.

3.6 Pausing a MIDlet

Sometimes it might be useful that the MIDlet pauses itself automatically. For example, all possible system notifications are shown over the MIDlet, thus hiding the MIDlet in the background.

MIDP specifications define that a system can set MIDlets to the paused state if the system needs a lot of the device's resources. In the paused state, the MIDlet frees all its resources. The currently available Series 60 Developer Platform 2.0 devices have sufficient running power and memory, and they do not need to enter the Paused state. This means that developers must create their own pausing mechanisms.

In other words, developers must create a method that pauses an active session when one of the following events occurs during the MIDlet interaction.

- A system notification is shown on the screen (for example, an incoming call, battery full or low).
- The user presses the red dial key, the power key, or the application key of the device.
- The main screen of the application is hidden by a system menu or another application is set to run on the foreground.

In practice, the MIDlet must always be paused when it is hidden. This is especially important in game applications, because the player will probably lose the game if it is not immediately paused when the game is hidden.

MIDlets that have been created for the Series 60 Developer Platform can be paused with the `isShown()` method of the Displayable class or `hideNotify()` method of the Canvas or CustomItem classes.

All User Interface (UI) components are inherited from the Displayable class. Thus the `isShown()` method can be used to test whether a UI component is visible or not. Repetitive requesting of the `isShown()` method can be used to get information if the UI component is hidden for some reason. The `isShown()` method can be used for high-level UI components like Form or List, but in case of low-level UI components that are inherited from Canvas or CustomItem classes it is better to use their proprietary `hideNotify()` and `showNotify()` methods.

The `hideNotify()` method is called right after the Canvas object has left the display. Create an auto-pause mechanism inside the `hideNotify()` method to stop threads, cancel timers, save important values, and so on. See the following code example:

```
protected void hideNotify()
{
   remainingTime = endTime - System.currentTimeMillis();
   myThread.stop();
   autoPaused = true;
   repaint();
   // Include a pause test in paint() method to check if paused
   // paint a pause message on screen if autoPaused true
}
```

```
protected void paint(Graphics g)
  {
   // paint game screen here
   if (autoPaused == true) {
      // paint pause message
   }
  }
```

It is possible to create an auto-continue mechanism with the `showNotify()` method. The `showNotify()` method is called just before the Canvas is getting back on the display. In this method it is useful to, for example, restore important values, timers or restart threads, so the application execution can continue as normal.

```
protected void showNotify()
{
   myThread = new Thread(this);
   // Include a pause test in keyPressed() method for continue
   // set new endTime and set autoPaused false, then repaint
}

protected void keyPressed(int keyCode)
  {
   if (autoPaused == true) {
      autoPaused = false;
      endTime = System.currentTimeMillis() + remainingTime;
      myThread.start();
      repaint();
   }
   // Rest of key detection routine here
  }
```

Check that:

- The application can be paused.
- MIDlet pauses itself automatically if it is hidden.

4. Usability Testing

4.1 Usability Basics

This chapter lists only the basic requirements for usability testing. For more thorough instructions on planning good usability, refer to the document *Developer Platform 1.0 for Series 60: Usability Guidelines for J2ME™ Games v1.0.*

Good usability builds the basis for a good application. If the application's usability is poor, some users might stop using the application. To ensure user satisfaction you should arrange separate usability tests and start testing the application already in an early phase of application development. Fixing possible usability problems is easier if they are found in an early phase.

For positive user experience it is important that all main functionalities of the application can be accessed easily through the main menu. Ensure that each functionality works properly as described in the application documentation and that there are no hidden functionalities.

Check also that the terminology is correct and that there are no grammatical mistakes. Do not use difficult terminology or abbreviations.

Ensure that the application gives enough feedback to the user. Typically users do not wait for more than a couple of seconds if the display does not change and seems jammed after they have selected a function.

Thus any selection of a different function in the application should be performed within 5 seconds. There must be a visual indication within 1 second that the function will be performed. The visual indication can be, for example, prompting for user input, using splash screens or progress bars, or displaying text such as "Please wait...", "Sending message". These indications can be enhanced with sounds.

Check that:

- All main functions of the application can easily be accessed through the main menu.
- Each functionality works as described in the documentation.
- Terminology and grammar are correct and no abbreviations are used.
- The application gives enough feedback to the user.

4.2 User Interface

4.2.1 Menus

Check that menu structures are grouped logically and not too deep. A wide menu hierarchy is preferred over a deep one. The user must not get lost in the menus. Ensure that the back button is always at hand. Closing the application must be easy as well.

The order of the items in the menus must be well defined. It is advisable to arrange items by the frequency of use, that is, the most frequently used item should be on the top of the menu.

All items in the menus must be shown as selectable items and the labels of the items must be descriptive. The user must easily see which screen items can be selected and/or changed, especially if low–level UI components are used as menus. Selectivity is not a problem if high–level UI menu components are used because they are similar to the system menus.

If the application requires completing multi-phased forms, the current screen number as well as the total number of the screens must be indicated to the user in, for example, the following format: Page 3/5. The last screen in the form must give the user some informative feedback, for example, a Thank you screen after pressing the Submit button.

Add a Help/Instructions option to the main menu of the application. However, be sure that the user can perform the main tasks without reading the Help/Instructions section. Instructions must be useful and complete and include at least purpose of the application, rules, and use of the keys.

The application should contain a screen that gives information about the developer and other generic information about the application. This could, for example, be an About screen in the main menu.

Check that:

- Sequences of actions are grouped logically.
- Menus are grouped logically and menu structures are not too deep.
- The back button is always available.
- The order of the menu is logical.
- Menu items are marked clearly.
- The application has a Help/Instructions topic.
- Information about the developer and the application is given.

4.2.2 Commands

Check that softkey labels reflect user interface style of the device. Keep in mind that in Series 60 devices the system automatically locates all commands that have a positive meaning (for example, Open, Yes, Next, continue, Start) on the left softkey whereas commands with a negative meaning (for example, Back, Close, No, Cancel, Stop) are located either on the right softkey or on the left softkey because only one command can be located under the right softkey. The possible other negative commands are located under the left softkey.

To which softkey and in which order the commands are presented depends on:

- Command type.
- Command priority.
- The order in which the commands where added to the UI component.

Refer to the *Series 60 Developer Platform 2.0: Specification v1.0* for more information about command mappings and command labels.

The following code sample shows how to create positive and negative commands.

```
//Positive commands
Command cmdOpen = new Command("Open", Command.SCREEN, 1);
Command cmdOK = new Command("OK", Command.OK, 1);

//Negative commands
commandCancel = new Command("Cancel", Command.CANCEL, 1);
commandBack = new Command("Back", Command.BACK, 1);
```

In Series 60 devices all UI components, except for Nokia UI API's FullCanvas class, include an Exit or Close command by default, developers do not have to add exiting commands to those UI components.

- Check that softkey labels reflect the style of the user interface.

4.2.3 Sounds

Ensure that each sound (if sounds are used) in the application is distinctive and has a well-defined meaning. For example, a happy sound must be used for a positive action and a sad sound for a negative action.

Be sure that the current status of the possible sound settings does not affect the playability of the application (for example, whether or not the sound is activated).

All in all, you should be careful when adding sounds because too much sound may make the use of the application annoying. Therefore, you should add sounds only to places and situations where they give extra information or experience for the user. Be sure that sounds can be set off easily.

Check that:

- Each sound has a distinctive meaning.
- Sound settings do not affect the playability of the application.
- Sounds are not overused in the application.

4.2.4 Backlights and vibra

Nokia's UI API of the MIDP implementation for Series 60 devices does not support backlights and vibra functionality. However, those features are supported via the MIDP 2.0 APIs.

The vibra functionality can be used to give extra feedback to the user. Please note that it is a wearing functionality, so it should be used only in special circumstances and more rarely than sounds.

Backlights should always be on if the application is actively used without user interaction (there have been no key presses to keep the backlights automatically on). However, backlights consume quite a lot of battery power, so there should be a setting that enables to turn the backlights off.

Check that:

- Vibra is not used too often.
- The user can turn backlights off.

4.2.5 Settings

It is advisable to have user settings for the application, for example, settings for sounds. Regardless of the settings it is important that all settings and selections

can be canceled or restored. Be also sure that the current status of each setting has a clear meaning and that the user can easily understand what changes if a setting is changed. Note that settings should be saved when the application is closed.

Check that:

- The user can cancel and/or restore settings and selections.
- The current status of each setting is clear.
- Settings are saved.

4.2.6 Localization

The date format must be handled according to the target country. Verify that date, time, time zone, week start, numeric separators, and currency are formatted according to the conventions of the target country of the implemented language and supported throughout the application.

Data entry fields must accept and properly display national alphabets, for example, ä, ü, and ß. If possible, test that all data entry fields accept and properly display all special character input.

Check that:

- The language conventions of the target country are followed.
- National alphabets are accepted and displayed.

4.2.7 Games

The following guidelines concern especially game MIDlets. It is important that the general view of the game is clear and that the game gives enough feedback and immediately responds to the player.

The rule or function of each element in the game must be clear (for example, how the player and the enemies are represented, and how they interact).

The game must be paused automatically as described earlier and it is also important that the player can pause the game by him/herself. When the game is paused it must be easily continued.

If the application is closed in the middle of the game, the game status should be saved so that the game can be continued later on.

Check that:

- The general view and the rules and functions of the game are clear.
- The game gives enough feedback to the player.
- The user can pause and continue the game and the status is saved.

5. References

Developer Platform 1.0 for Series 60: Usability Guidelines for J2ME™ Games v1.0. Available through http://www.forum.nokia.com/

Efficient MIDP Programming. Available through http://www.forum.nokia.com/

MIDP 1.0: Mobile Information Device Profile for Java 2 Micro Edition, version 1.0, JSR 37 Expert Group. Available through http://java.sun.com/products/midp/

MIDP 2.0: Mobile Information Device Profile for Java 2 Micro Edition, version 2.0, JSR 118 Expert Group. Available through http://java.sun.com/products/midp/

Series 60 Developer Platform 2.0: Specification v1.0. Available through http://www.forum.nokia.com/