How to implement purchase and restoration of multiple items with In-App Purchase and Java ME

This article describes how to perform purchase and restoration of 2 or more items with In-App Purchase for Series 40 by piping the purchase and restoration tasks in a thread. The In-App Purchase API does not support simultaneous multi-purchases or multi-restorations on Series 40. This is because the In-App server blocks with the first incoming request. No further requests can be processed by the server until the current one is completed. The client needs to implement the necessary logic, in order to queue the multiple restorations or purchases locally and forward them to the sever only after the currently active task is completed.

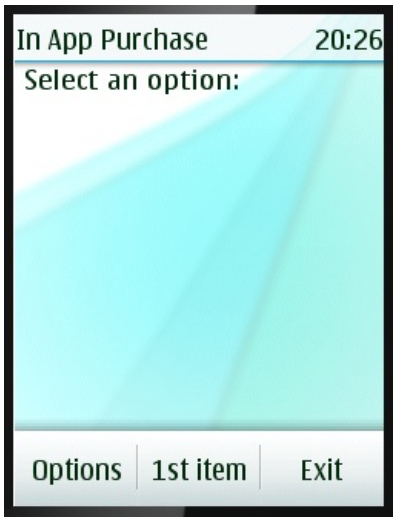
Introduction

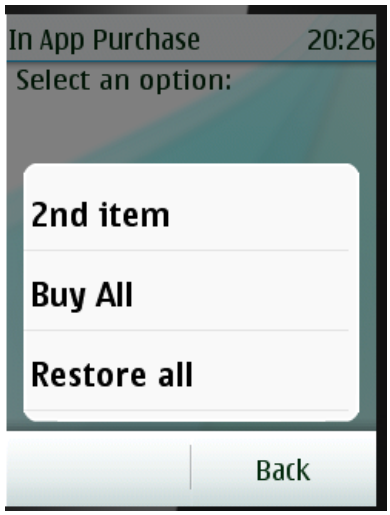
In this example we demonstrate how to purchase and restore simultaneously two DRM-protected items. Please note that restoration can only be performed for Nokia DRM-protected content. Restoration of DRM-protected content currently cannot be tested before the application is published and the content is DRM-protected by Nokia Store. We therefore use the In-App Purchase Simulation of Nokia's SDK 1.1 for Java, for running and testing the MIDlet. The application consists of following commands:

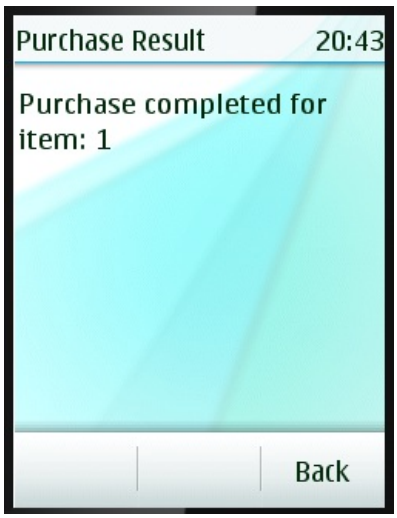
- Display information about the first item
- Display information about the second item
- Perform a purchase for both the first and the second item
- Restore both items

Note: In order for this example to properly work, one needs to create an IAP_VARIANTID.txt file inside the resource directory of the working project, having six zeros 000000 as content

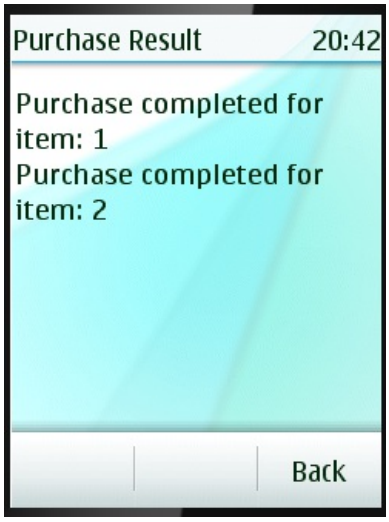
The information about each item's price, description and title is shown on the same screen as the commands. The result from the purchase and restoration is shown on a new screen and there is a back command to return to the main screen. In the screen shots below, a multiple purchase is attempted and the results for each purchase are asynchronously returned one after the other by the server:







(C) Copyright Nokia 2014. All rights reserved.

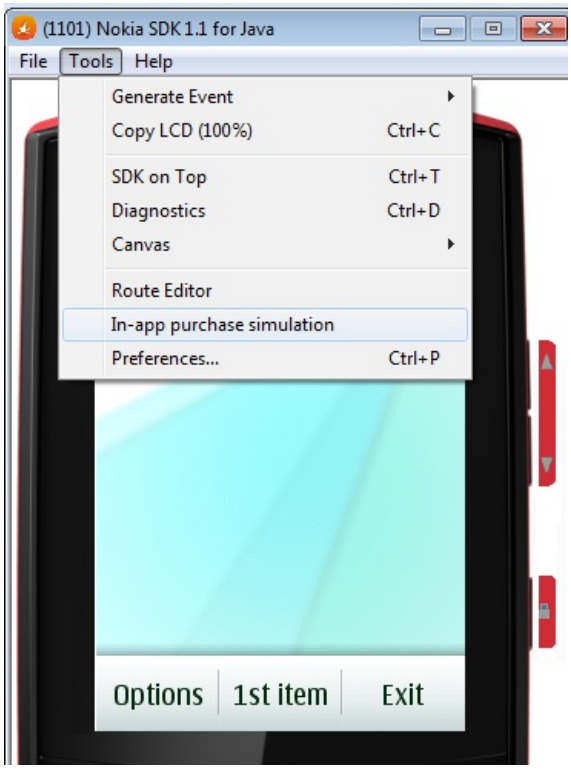


In-App Purchase Simulation Settings

Before one can test the code for this MIDlet, the SDK's In-App Purchase Simulation settings need to be properly configured. Two In-App items should be added to the simulation, each with a description, a price, a title and a Product ID. The Product ID is a unique 6 digit identifier for the purchasable item. It is assigned automatically by Nokia when the item has passed the Store's Quality Assurance process. Here, we use two arbitrary Product IDs. Please note, that these should be replaced with the actual ones, before publishing the application.

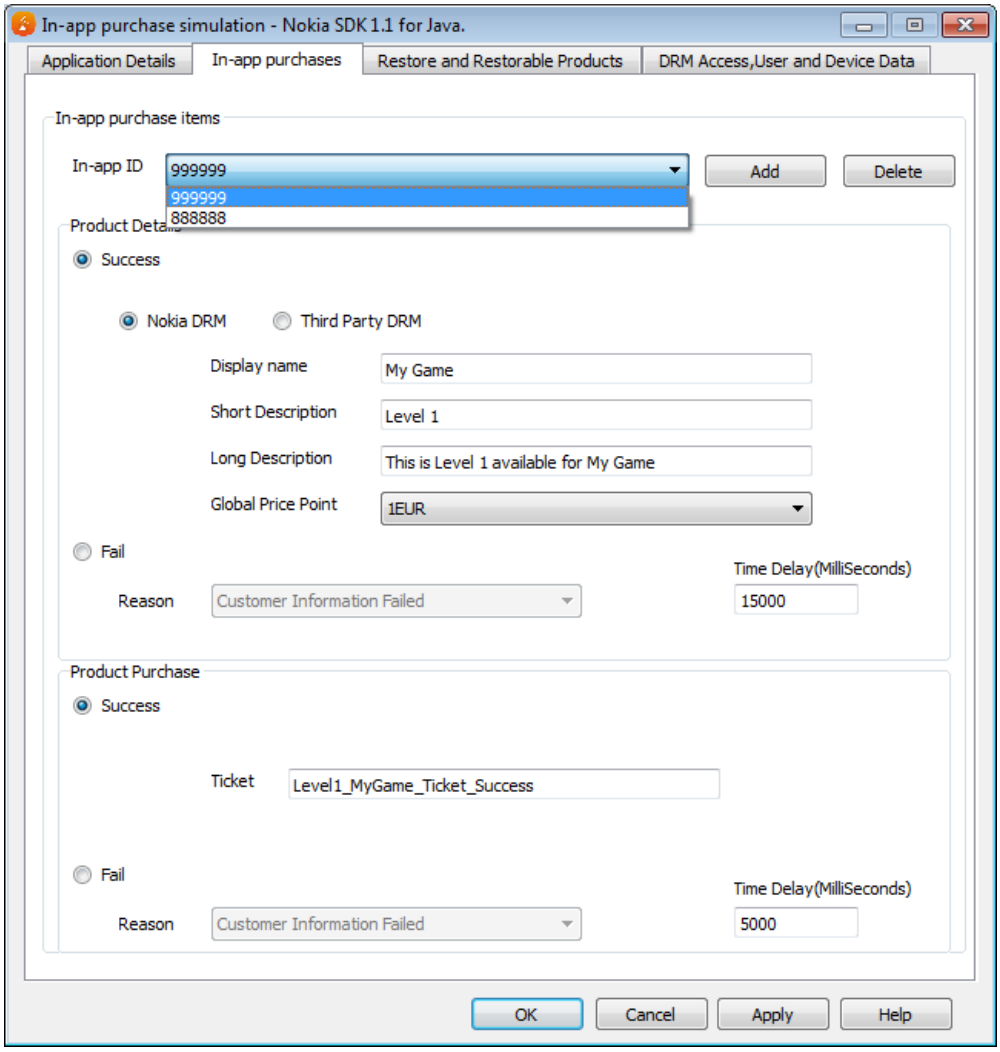
The images below demonstrate

- how to launch the In-App Purchase Simulation from the SDK



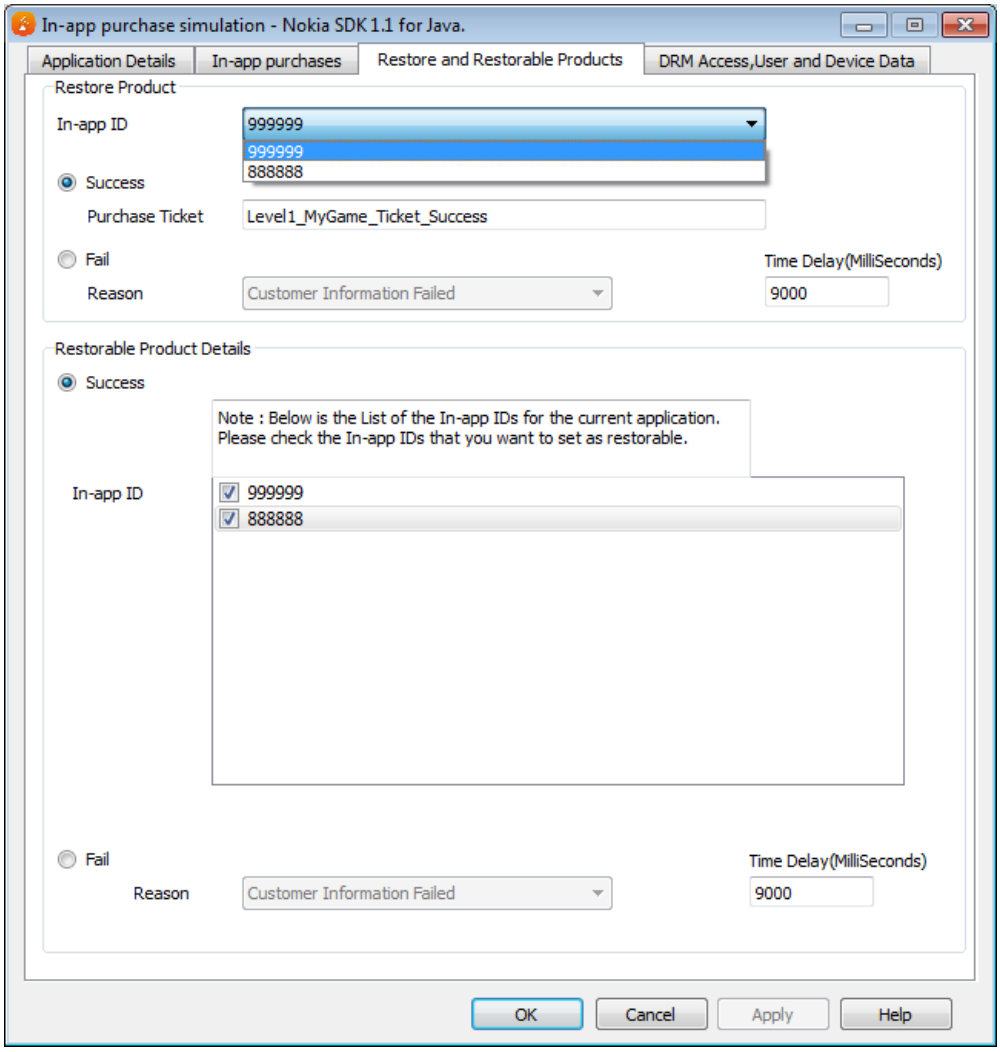
- how to add Product IDs along with their metadata for purchase operations

(c) Copyright Nokia 2014. All rights reserved.



- and how to enable DRM-protected restoration simulation for each Product ID

(c) Copyright Nokia 2014. All rights reserved.



Notice that a custom delay can be manually entered by the user, to simulate the server's delay in returning a response.

The logic behind piping the requests

The Product IDs are stored in an array. This example uses only two items, but the same logic can be applied to three or more items. All the multiple purchases and restorations occur inside a thread. The thread reads the array with the Product IDs, performs either a purchase or restoration for each ID and pauses after every iteration. It regularly wakes up to see if the current purchase or restoration is completed in order to continue with the next.

```
                                                //this loop iterates through the items that
need to be restored or purchased
  for(int i=0; i<ids.length ; i++)
  {
   //this is the item under restoration or purchase currently
   activeId ++;
   int status;

   if(!multipleRestoration) {
    status = manager.purchaseProduct(ids[i],
IAPClientPaymentManager.FORCED_AUTOMATIC_RESTORATION);
   }
   else {
    //sends the restoration request
    status = manager.restoreProduct(ids[i],
IAPClientPaymentManager.ONLY_IN_SILENT_AUTHENTICATION);
   }
   if (status != IAPClientPaymentManager.SUCCESS) {
```


(c) Copyright Nokia 2014. All rights reserved.

```
    System.out.println("do not expect a call back");
   }
   //doesn't allow the restoration or purchase to proceed to the next item in line,
unless the current restoration receives a call back
   proceed = false;

   while( !proceed) {
    try {
     //sleeps for 2 seconds then wakes up and checks if it is time for the next item to
be restored
     Thread.sleep(2000);
    } catch (InterruptedException e) {
     // handle Thread interruption
    }
   }
  }
```

In the above snippet, the purchaseProduct call will cause the MIDlet to connect to the SDK's In-App Purchase Simulation, process the request and return back the result via the purchaseCompleted call back method:

```
 //purchase call back
 public void purchaseCompleted(int status, String ticket) {
  if(status == IAPClientPaymentManager.SUCCESS) {
   resultForm.append("Purchase completed for item: " + (activeId + 1) + "\n");
  }
  else {
   resultForm.append("Purchase failed for item: " + (activeId + 1) + "\nThe status is: "
+ status + "\n");
  }
  // this unblocks the sleeping thread and performs purchase on the next item (if any)
  proceed = true;
 }
```

At the end of this call back method a boolean flag is turned to true in order to allow the sleeping thread proceed with the next purchase.

Similarly, the restoreProduct call, will query the server and return a response via the restorationCompleted method:

```
 //restoration call back
 public void restorationCompleted(int status, String ticket) {

  if( status ==IAPClientPaymentListener.OK) {
   resultForm.append("Restoration completed for item: " + (activeId + 1) + "\n");
  }
  else {
   resultForm.append("Restoration failed for item: " + (activeId + 1) + "\nThe status
is: " + status + "\n");
  }
  //this unblocks the sleeping thread and performs restoration on the next item (if any)
  proceed = true;
 }
```

The MIDlet's code

(c) Copyright Nokia 2014. All rights reserved.

```
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.StringItem;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;
import com.nokia.mid.payment.IAPClientPaymentException;
import com.nokia.mid.payment.IAPClientPaymentListener;
import com.nokia.mid.payment.IAPClientPaymentManager;
import com.nokia.mid.payment.IAPClientProductData;
import com.nokia.mid.payment.IAPClientUserAndDeviceData;


public final class InAppMultiItemsMIDlet
 extends MIDlet
 implements IAPClientPaymentListener, CommandListener, Runnable {

 private Form mainForm; //the main form
 private Form resultForm; //this form is used for the result for purchasing and
restoring
 private Display display;
 private Command exitCmd = new Command("Exit", Command.EXIT, 0);
 private Command showFirstID = new Command ("1st item", Command.OK, 1); //displays
information for the 1st item
 private Command showSecondID = new Command ("2nd item", Command.OK, 2); //displays
information for the 2nd item
 private Command buyAll = new Command("Buy All", Command.OK, 5); //performs
multipurchase
 private Command restoreAll = new Command ("Restore all", Command.OK, 6); //performs
multirestoation
 private Command backCmd = new Command ("Back", Command.BACK, 0); //this command returns
back to main form from the result form
 private StringItem info;

 IAPClientPaymentManager manager;
 Thread thread;  //the multiple restoration thread
 boolean proceed = false;  //blocks the multiple restoration thread until the current
restoration is completed
 String[] ids;  //The list of Product IDs to be restored
 int activeId = -1; //keeps track of the current item under restoration or purchase
 boolean multipleRestoration = false; //this flag changes a multi purchase to multi
restoration

 protected void startApp() throws MIDletStateChangeException {
  display = Display.getDisplay(this);
  mainForm = new Form("In App Purchase");
  display.setCurrent(mainForm);
  //all the commands
  mainForm.addCommand(exitCmd);
  mainForm.addCommand(showFirstID);
  mainForm.addCommand(showSecondID);
  mainForm.addCommand(buyAll);
  mainForm.addCommand(restoreAll);
```


(c) Copyright Nokia 2014. All rights reserved.

```
  info = new StringItem("Select an option",null);
  mainForm.append(info);
  mainForm.setCommandListener(this);

  //some random values. These need to be changed with the appropriate Product IDs.
  ids = new String[2];
  ids[0] = "999999" ; //First Item
  ids[1] = "888888" ; //Second Item

  try {
   manager = IAPClientPaymentManager.getIAPClientPaymentManager();
   IAPClientPaymentManager.setIAPClientPaymentListener(this);
  } catch (IAPClientPaymentException e) {
   info.setLabel("Error");
   info.setText("IAPClientPaymentException:" + e.getMessage() + "\n");
  }
 }

 public void productDataReceived(int status, IAPClientProductData pd) {
  //Title, price and short description is shown
     if (status == IAPClientPaymentListener.OK) {
      String fullDescription = "";

      String title = pd.getTitle();
         String price = pd.getPrice();
         String sdesc = pd.getShortDescription();

         fullDescription = "Title:" + title + "\n" + "Price:" + price + "\n" + "Short
Description:" + sdesc + "\n";
         info.setLabel("Description");
         info.setText(fullDescription);
     }
     else {
      info.setLabel("Error");
      info.setText("Product data retrieval failed with code:" +status);
     }
 }

 //each operation needs the product ID of the item in question
 public void commandAction(Command c, Displayable d) {
  if(c == exitCmd) {
   notifyDestroyed();
  }

  if(c == showFirstID) {
   showDescription("999999");
  }

  if(c == showSecondID) {
   showDescription("888888");
  }

  if(c == buyAll) {
   multipleRestoration = false;
   doPurchaseRestoreAll(ids);
  }
```


(C) Copyright Nokia 2014. All rights reserved.

```
  if(c == restoreAll) {
   multipleRestoration = true;
   doPurchaseRestoreAll(ids);
  }

  if(c == backCmd) {
   display.setCurrent(mainForm);
  }
 }

 public void showDescription (String id) {
  int status = manager.getProductData(id);
  if (status != IAPClientPaymentManager.SUCCESS) {
       System.out.println("Do not expect a call back\n");
  }
 }

 public void doPurchase(String id) {
  resultForm = new Form("Purchase Result");
  resultForm.addCommand(backCmd);
  resultForm.setCommandListener(this);
  display.setCurrent(resultForm);
  manager.purchaseProduct(id, IAPClientPaymentManager.FORCED_AUTOMATIC_RESTORATION);
 }

 public void doPurchaseRestoreAll(String[] ids) {
  //the thread that handles the multiple restoration and purchase
  thread = new Thread(this);
  thread.start();
 }

 //purchase call back
 public void purchaseCompleted(int status, String ticket) {
  if(status == IAPClientPaymentManager.SUCCESS) {
   resultForm.append("Purchase completed for item: " + (activeId + 1) + "\n");
  }
  else {
   resultForm.append("Purchase failed for item: " + (activeId + 1) + "\nThe status is: "
+ status + "\n");
  }
  // this unblocks the sleeping thread and performs purchase on the next item (if any)
  proceed = true;
 }

 //restoration call back
 public void restorationCompleted(int status, String ticket) {

  if( status ==IAPClientPaymentListener.OK) {
   resultForm.append("Restoration completed for item: " + (activeId + 1) + "\n");
  }
  else {
   resultForm.append("Restoration failed for item: " + (activeId + 1) + "\nThe status
is: " + status + "\n");
  }
  //this unblocks the sleeping thread and performs restoration on the next item (if any)
  proceed = true;
```


(c) Copyright Nokia 2014. All rights reserved.

```
 }

 public void run() {
  //Displays the restoration result page
  if(!multipleRestoration) {
   resultForm = new Form("Purchase Result");
  }
  else {
   resultForm = new Form("Restoration Result");
  }
  resultForm.addCommand(backCmd);
  resultForm.setCommandListener(this);
  display.setCurrent(resultForm);

  //this loop iterates through the items that need to be restored or purchased
  for(int i=0; i<ids.length ; i++)
  {
   //this is the item under restoration or purchase currently
   activeId ++;
   int status;

   if(!multipleRestoration) {
    status = manager.purchaseProduct(ids[i],
IAPClientPaymentManager.FORCED_AUTOMATIC_RESTORATION);
   }
   else {
    //sends the restoration request
    status = manager.restoreProduct(ids[i],
IAPClientPaymentManager.ONLY_IN_SILENT_AUTHENTICATION);
   }
   if (status != IAPClientPaymentManager.SUCCESS) {
    System.out.println("do not expect a call back");
   }
   //doesn't allow the restoration or purchase to proceed to the next item in line,
unless the current restoration receives a call back
   proceed = false;

   while( !proceed) {
    try {
     //sleeps for 2 seconds then wakes up and checks if it is time for the next item to
be restored
     Thread.sleep(2000);
    } catch (InterruptedException e) {
     // handle Thread interruption
    }
   }
  }
  activeId = -1;
 }

 //unimplemented methods
 protected void destroyApp(boolean arg0) throws MIDletStateChangeException {

 }

 protected void pauseApp() {
```


(C) Copyright Nokia 2014. All rights reserved.

```
  }

  public void userAndDeviceDataReceived(int arg0,IAPClientUserAndDeviceData arg1) {

  }

  public void productDataListReceived(int arg0, IAPClientProductData[] arg1) {

  }

  public void restorableProductsReceived(int status, IAPClientProductData[] list) {

  }
 }
```

Resources

The source code of this MIDlet is available for download from here: File:InAppMultiItemsMIDletSource.zip

The binary files of this MIDlet are available for download from here: File:InAppMultiItemsMIDletBinaries.zip

See also

- The Developer's guide - In App Purchase

(C) Copyright Nokia 2014. All rights reserved.