Java Card technology, debugging, upload

Javacard overview

Sun Microsystems published the Java Card Platform Specification and the Java Card Development Kit, which includes a reference implementation based on the specification. The aim is to provide the basis for cross-platform and cross-vendor applet interoperability. The current version of the JavaCard specification is 3.0, but available cards usually supports only 2.1 or 2.2 [JC222]. JavaCard applet is Java-like application that is uploaded to a smart card and is executed by the Java Virtual Machine on the smart card.

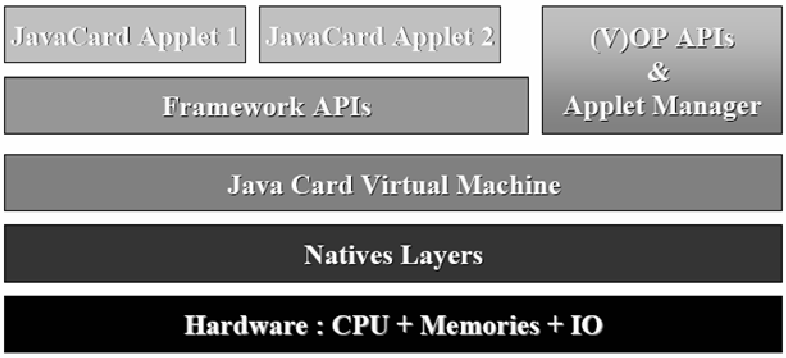

Figure 1 - JavaCard architecture overview

Java Card applications are compiled using common Java compilers. Due to limited memory resources and computing power, Java Cards do not support:

- dynamic class loading,
- security manager,
- threads and synchronization,
- garbage collection and object cloning,
- finalization,
- large primitive data types (float, double, long and char) and
- most of std. classes (most of the java.lang, Object and Throwable in limited form).

JavaCard security

The main security features of Java Card include:

- All the benefits of the Java language: data encapsulation, safe memory management, packages, etc.
- Applet isolation based on the Java Card firewall: applets cannot directly communicate with each other. Special interface for sharing objects Shareable must be implemented to allow cross applets interaction.
- Atomic operations using transaction mode: JCSystem.beginTransaction(), commitTransaction(), abortTransaction().
- Transient data, which guarantees that sensitive session data is wiped out: JCSystem.makeTransientByteArray().
- A rich cryptography API for encryption, digital signatures and message digests.
- Secure communication with the card reader, if the card is Open Platform compliant (secure messaging, security domains).

Creating a JavaCard applet (GemXpressoRADIII, JBuilder7)

The aim of this text is to create and compile a simple JavaCard applet. Detailed information can be found in JavaCard API documentation. Described techniques use GemXpresso RADIII and its plug-in into Borland JBuilder7.

Required development tools

- Development environment (IDE) for Java language (e.g. Borland JBuilder7)
- GemXpresso RADIII. Contains plug-in for JBilder7 and management environment JCardManager for loading applets onto smart card, sending APDU commands etc.
- JavaCard Development Kit 2.1.2

General information

Whole process consists of four logical steps:

1. Creation of the applet as a descendant of the class javacard.framework.Applet in arbitrarily development environment for Java language with respect to restrictions of the JavaCard platform (restricted primitive types, only basic classes from javacard.* package). Compilation using general Java compiler into *.class.
2. Conversion of *.class into *.jar using JavaCard converter. The code can be checked using JavaCard Off-Card Verifier for correctness (not necessary for testing purposes).
3. Loading file *.jar onto the smart card, installation of the included applet and registration in operating system of smart card. Done using appropriate interface like OpenPlatform.
4. Repeated usage of the installed applet. During the first step, applet with the given AID (unique identifier) is selected as the active one using SELECT command. All subsequent APDU commands are directly passed to the active applet.

Source code, compilation

Using wizard *File->New…->Gemplus->JavaCard applet*, a class with a selected name (*„Applet name“*) is created. Set element *type* to *„JavaCard Open Platform Applet“*, insert unique identification of package (*„Package AID“*) and unique identification of applet (*„Applet AID“*). In the next step, select the type of the card for which the conversion will be performed.

The result is a skeletal code of the applet that can be compiled and uploaded onto smart card immediately. The skeletal code contains basic functionality required for installation and registration of applet in the operating system of smart card.

Applet contains the following methods:

- *protected applet_name(byte[] buffer, short offset, byte length)* (constructor). It is called from the *install* method only once during installation of applet. It is preferable to initialize all needed structures, allocate all needed memory and objects that will be required by applet in the body of this method (memory can be already occupied later)
- *public static void install(byte[] bArray, short bOffset, byte bLength).* This method is called only once during the installation of the applet. Perform only calling of applet constructor.
- *public boolean select().* This method is called every time, when somebody tries to select this applet as the active one. Selection of an applet can be suppressed in this method by returning *false* value, for example because a required condition is not satisfied. Needed structures can be initialized or security conditions reset here. This method is called every time before applet can be used.
- *public void deselect().* This method is called when the terminal sends a command to set the applet as inactive (incoming commands will not be sent to this applet any

more until new select command is sent). It can be used for the cleanup of the code. WARNING: There is no guarantee that this method will be called! The method is called only during a correct deactivation of the applet, but not when smart card is suddenly removed from reader. That is why it is highly recommended to perform cleanup and initialization in the *select()* method.

- *public void process(APDU apdu)* . This method serves as the entrance gate for all APDU commands (except system-reserved) received by the card after SELECT command. Is properly to define own class of instructions (CLA), test the header of incoming APDU to this class value and return exception ISO7816.SW_CLA_NOT_SUPPORTED if it does not match. Branching based on the value of the INS parameter is performed using the switch statement and the APDU command (APDU object) is passed to the appropriate method. Actions inside *process()* method are under control of the applet programmer.

```
public class Foo extends javacard.framework.Applet
   protected Foo(byte[] buffer, short offset, byte length)  {
     // allocation of required memory and objects, initialization of structures
     m_byteArray = JCSystem.makeTransientByteArray((short) 128,  System.CLEAR_ON_DESELECT);
     // registration of instance
   }
   public static void install(byte[] bArray, short bOffset, byte bLength) throws ISOException  {
      new Foo (bArray, bOffset, (byte)bLength );
   }
   public boolean select()  {
      // <PUT YOUR SELECTION ACTION HERE>
      // return status of selection, if false, applet is not selected and cannot be used
      return true;
   }
    public void deselect()  {
      // <PUT YOUR DESELECTION ACTION HERE>
      return;
   }
   public void process(APDU apdu) throws ISOException  {
     // get the APDU buffer
     byte[] apduBuffer = apdu.getBuffer();

     // ignore the applet select command dispached to the process
     if (selectingApplet()) return;

      if (apduBuffer[ISO7816.OFFSET_CLA] == INS_CLA_FOO) {
        // APDU instruction parser
        switch ( apduBuffer[ISO7816.OFFSET_INS] ) {
           case INS_CARD_SET_KEY: SetKey(apdu); break;
           case INS_CARD_COPYINPUT: CopyInput(apdu); break;
           default: ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED); break;
         }
      }
      else ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
  }
```

Other methods can be defined as needed and are called from the *process()* method. If there is a need to work with incoming/outgoing data (usual), the following structure of the method is needed:

```
void Test(APDU apdu) {
        // reading of APDU header that contains CLA, INS, P1, P2, LC
        byte[]    apdubuf = apdu.getBuffer();

        // reading of incoming data (relevant only when LC > 0)
        short     dataLen = apdu.setIncomingAndReceive();

         // manipulating header (e.g. reading of CLA and P1)
         byte cla = apdubuf[ISO7816.OFFSET_CLA];
         byte p1 = apdubuf[ISO7816.OFFSET_P1];

         // sending the content of m_testArray with length ARRAY_LENGTH to SC output
         Util.arrayCopyNonAtomic(apdubuf, ISO7816.OFFSET_CDATA, m_testArray, (short) 0,
         ARRAY_LENGTH);
         apdu.setOutgoingAndSend(ISO7816.OFFSET_CDATA, ARRAY_LENGTH);
   }
```

The applet is compiled using Project->Make project and converted into a smart card compatible format. The conversion starts using the Convert command from the plug-in menu added by the GemXpresso RADIII into environment (viz. **Figure 2**). This menu can be also used for modification of settings of project, adding new types of card for conversion etc.

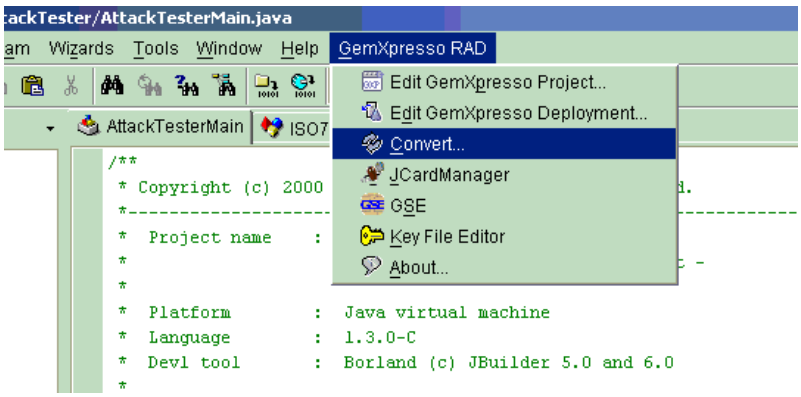

Figure 2: Conversion of compiled code for smart card.

Applet uploading, installation

In this step we assume, that we already have compiled and converted the applet (files *.jar or *.sap (for smart card simulator)).

1. Run JCardManager environment
2. Insert the smart card or run the smart card simulator (Tools->Gse Gui)
3. Switch to tab OP 2.0.1

4. Authenticate against the smart card („Authenticate“ command). Smart cards used for testing purposes have the default file containing authentication information (diversification keys - e.g. for *GXP_E64PK* the file C*:\Program Files\Gemplus\RADIII\resources\targets\GXP_E64PK_MUNI.properties*)
5. If older version of applet with same AID is already present on the card, it must be removed using “Delete” command. Instances of the applet must also be removed. Remove instances of the applet FIRST and the package as SECOND. The package with active instance cannot be removed.
6. Upload the new package with your applet using “Upload file into a card“ command. Set the path to package (*.jar or *.sap) and unique identification of the package „Package AID“.
7. Install applet. „Package AID“ (see step 6.), „Applet AID“ and „Instance AID“ must be filled. Last two can be same. During installation, *install()* method of applet of your applet is invoked.

That steps will finish the installation of the applet and the applet is ready for use. The applet will resist on the smart card until it is explicitly removed. If the smart card simulator is used without saving the internal state (“clean” card every time), installation must be performed every time. Steps from 4 to 7 can be automated using GemXpresso Deployment (Deployment->New…).

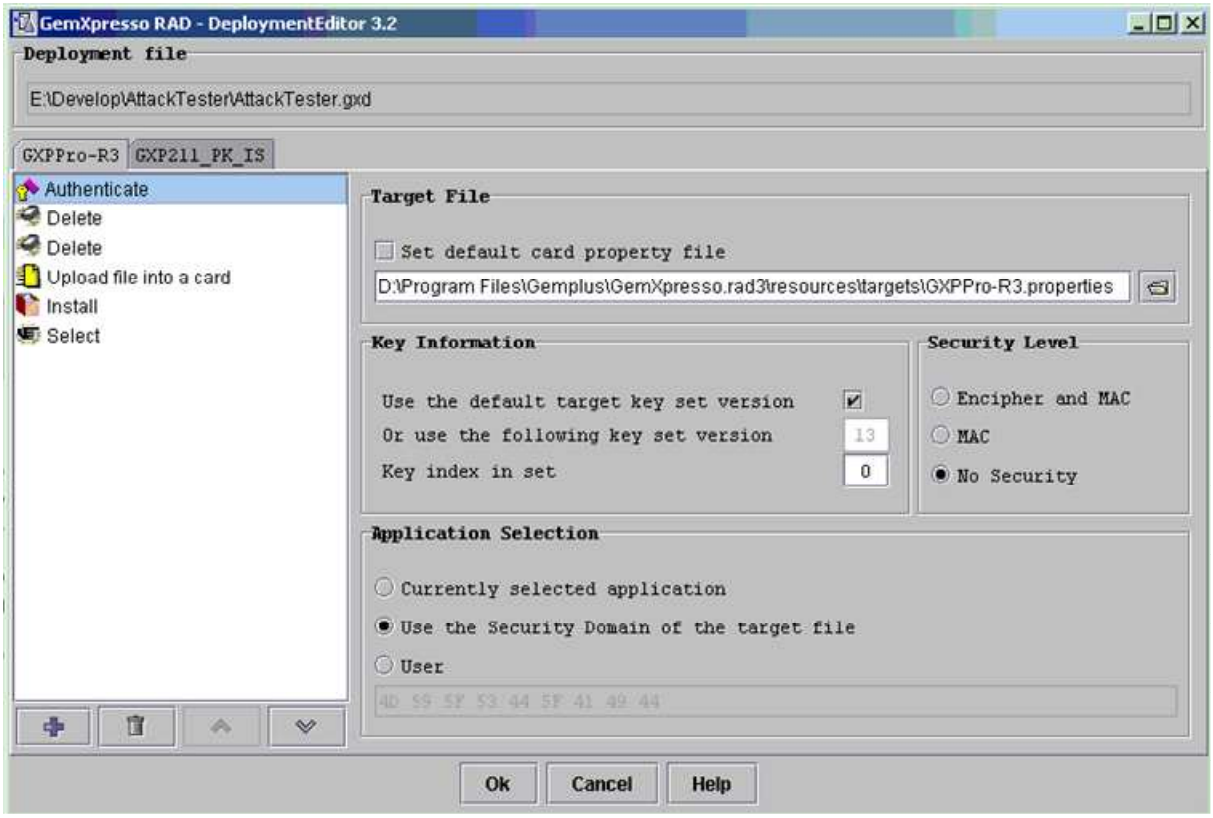

Figure A-3: GemXpresso Deployment.

Communication with applet

Installed applet with AID (e.g.. ‘41 74 74 61 63 6B 54 65 30 30 30 31‘) is assumed in this part. Switch to tab OP 2.0.1.

1. Send „Select“ command with AID of applet (41 74 74 61 63 6B 54 65 30 30 30 31). Return status <- 90 00 in log windows should be displayed as a confirmation of the successful selection of the applet as being active.
2. Send your commands using „Send APDU“ with parameters filled according to your needs. Card responses can be found in the log window.

Debugging using JBuilderu7 and Gemplus Simulator

The debugging process of a real smart card is difficult as there's no possibility to perform debug steps using the source code, no trace output can be displayed etc. GemXpresso RADIII environment contains a possibility to run smart card simulator for a particular type of a real smart card with debug features enabled. The debugging can be performed directly from Borland JBuilder7:

1. In project settings Project->Project Properties…->Run set item „Main class“ to 'com.gemplus.javacard.gse.Simulator' and „Application parameters“ to '-port 5000 -card GXPPro-R3' (GXPPro-R3 smart card will be simulated). For different type of card change string after *-card* statement and ALSO change type of linked library of simulator in Project->Project properties->Required libraries->GSE…
2. Compile and convert applet as usual.
3. Set breakpoints to the required positions in the code and run Run->Debug project. A simulator of the selected card is launched (GXPPro-R3 in our case). The type of the launched card is displayed in the log window of JBuilder7.
4. Upload and install the applet onto the simulator using the JCardManager and send *Select* command.
5. Send APDU that cause invocation of the method with our breakpoint.

RADII settings, paths, …

JBuilder7 paths

The paths to required libraries can be set in menu Project->Project Properties->Required Libraries. Two general groups of packages must be set, JavaCard package (for compilation of JavaCard applet) and Gemplus card simulator (for real-time debugging). To change packages in Required Libraries tab highlight packages group (e.g. GemXpresso – Crypto) and click on Edit. Then select proper package group in left side of window and add required packages in the right side as shown on figure 1.

1. JavaCard package (usually named as GemXpresso – Crypto):

- **cryptix-jce-api.jar** - usually positioned in $RADIII_installDir$\lib\ cryptix-jce-api.jar
- **cryptix-gemxpresso.jar** - usually positioned in $RADIII_installDir$\lib\cryptix-gemxpresso

2. Gemplus simulator (GSE). Depends on type of card, which you like to simulate. All simulators packages are usually positioned in $RADIII_installDir$\lib\gse\. Link only such single package, that corresponds to card you like to simulate. .

- e.g. if you like to simulate GXPPro-R3 card, then link package **gse_gxppro-r3.jar**
- e.g. if you like to simulate GXPLite-Generic card, then link package **gse_gxplite-generic.jar**

Note, that **GXPLite-Generic simulator do NOT support SHA-1 and MD5** hash functions. If you like to debug applet with these functions, use simulator of GXPPro-R3 card.

JBuilder7 debugging

See figure 5 for overview.

1. Place breakpoint withing JBuilder IDE (red bullet on side of source code line).
2. Run debugg session using Run->Debug project – simulator is automatically started for you. Type of simulated card is determined by gse_XXX.jar package linked in step 2. In JBuilder logging window, you should see something like (GXPPRO-R3 card is simulated here):

   Processing keys diversification for default key set...
   GSE is starting the simulation of the "GXPPRO-R3" card
   Simulator is listening on port 5000...
   Server is running...
   Socket (3653) locked by host: 127.0.0.1
   < Card Reset >

3. Switch to JCardManager, upload and install applet to Terminal:"Simulator", Card:"simulated_card_type" (GXPPro-R3 in our case). When line of code with breakpoint is executed, debugger is triggered.
4. **Do not forget to close simulated card before running another debugging session** using big red square button in simulator window (in JBuilder IDE).

RADIII deployment

For setup working applet on real or simulated smar card, you have to accomplish following steps. Steps 3-7 can be automatized using RADIII Deployment.

1. Create, compile and convert JavaCard applet (described in PV181: Training 5 – Java Card technology)
2. Run JCardManager, insert card or run simulator, select proper reader Terminal and card type.
3. Authenticate against card (AUTHENTICATE command). Target file points to file that contains diversified keys that are used during authentication. These files are usually positioned in $RADIII_installDir$\ resources\targets\.
   a. If you are using simulator, then select proper file (e.g. GXPPro-R3.properties for GXPPro-R3) or just check "Set default card property file".
   b. If you are using real card, than proper file should be GXPPro-R3 E32PK.properties for GXP E32PK Pro-R3 r.3.2 (ATR= 3B 7D 94 00 00 80 31 80 65 B0 83 01 02 90 83 00 90 00) or **GXP_E64PK_MUNI.properties** for GXP E64 PK (ATR= 3B 7E 94 00 00 80 25 A0 00 00 00 28 56 80 10 21 00 01 08). "Set default card property file" must be unchecked!
4. Delete older version of applet, if installed (applet with same AID). At fisrt, delete applet instance, then applet package.
5. Upload new applet to card. Check "Automatic selection fo the extension…" that automatically select *.sap version of applet for simulator or *.jar version of applet for real card. Select or write manually path to applet files. These files are typically located in $Applet_directory$\oncard\card_name\....\javacard\applet_name\applet_name.sap. E.g. E:\ Develop\LabakApplet\oncard\GXPLite-Generic\Labak\javacard\Labak
6. Install applet. Set Package, Applet and Instance AID as set in JBuilder7 javacard project. You can acces and modify these values using JBuilder7->GexXpresso RAD->Edit GemXpresso Project. Package AID and Applet AID must be different (e.g. applet AID = package AID concatenated with '01'), applet AID and instance AID can be same.

7. Select applet by its applet AID.

Clever way how to automatize whole process is to use RADIII deployment that works as batch for commands described in steps 3-7 and can be repeately executed. **Each card type has its own tab in Deployment!** Based on card type selected in JCardManager main window commands set is executed. See figure 3, 4 for overview.

These sets can be different, however **most often they differ only in**:

- Authenticate command – proper *.properties file must be selected. Check “Set default card property file” if using simulated card.
- Upload file into a card command – proper directory for given card type must be selected (subdirectory of $Applet_directory$\oncard\ directory.

New deployment can be created by Deployment ->New from JCardManager. Add all available card types.

Running deployment

If you have selected proper terminal, card type and deployment is loaded using Deployment-> Open of Reopen command, than press **Deployment->Deploy.** Steps 3 – 7 are then sequentially executed and results is displayed in logging window. **Every command should end with “<- 90 00” status** (except Delete command, if no applet was previously installed). **If Select fail, then find last command, that ends sucesfully (90 00) and check settings for subsequent command.** Repeat this process, untill Select is OK. If you are using debbugger connected to simulated card, you can actually debug the Install command (constructor of applet is executed).

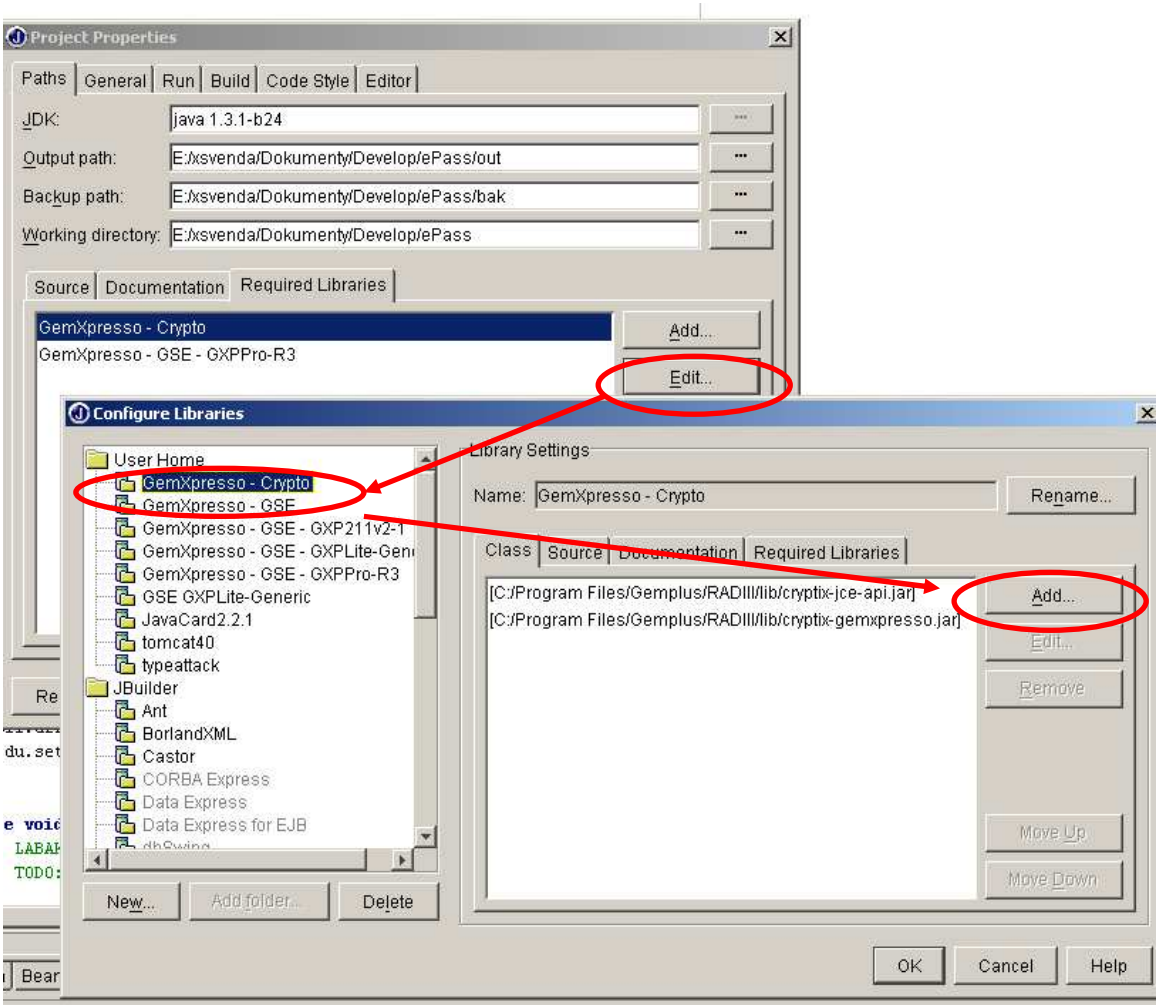

Figure 4. JBuilder libraries

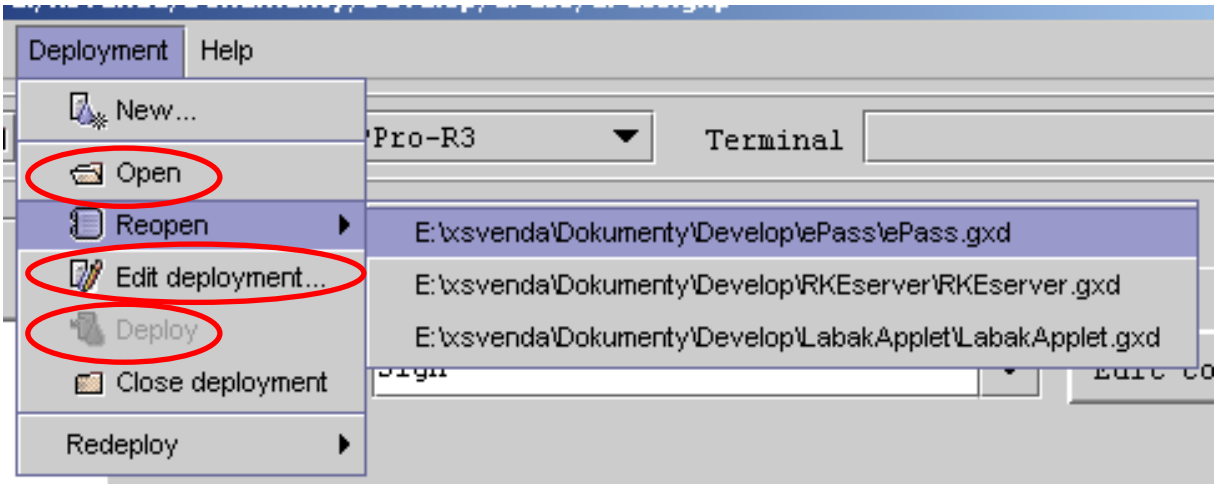

Figure 5 Open deployment

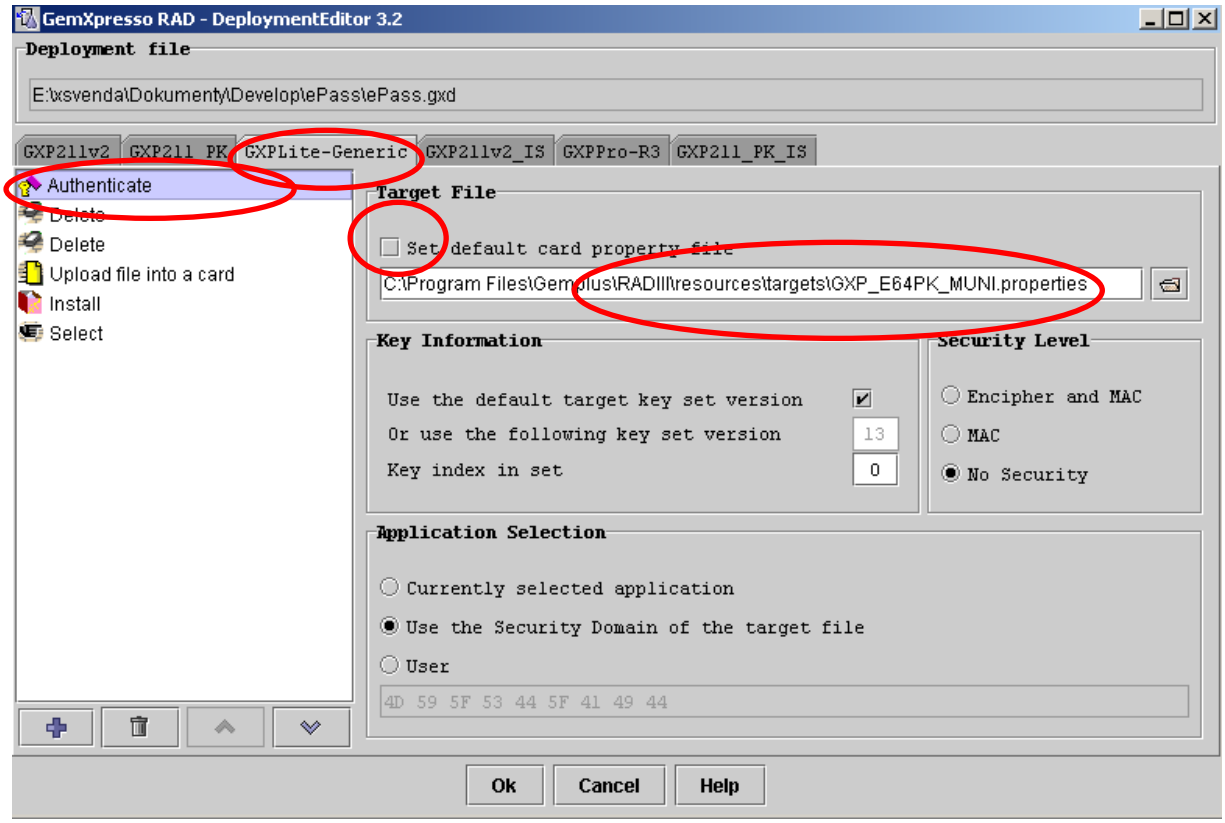

Figure 6 Deployment – Authenticate against real card GXP E64 PK

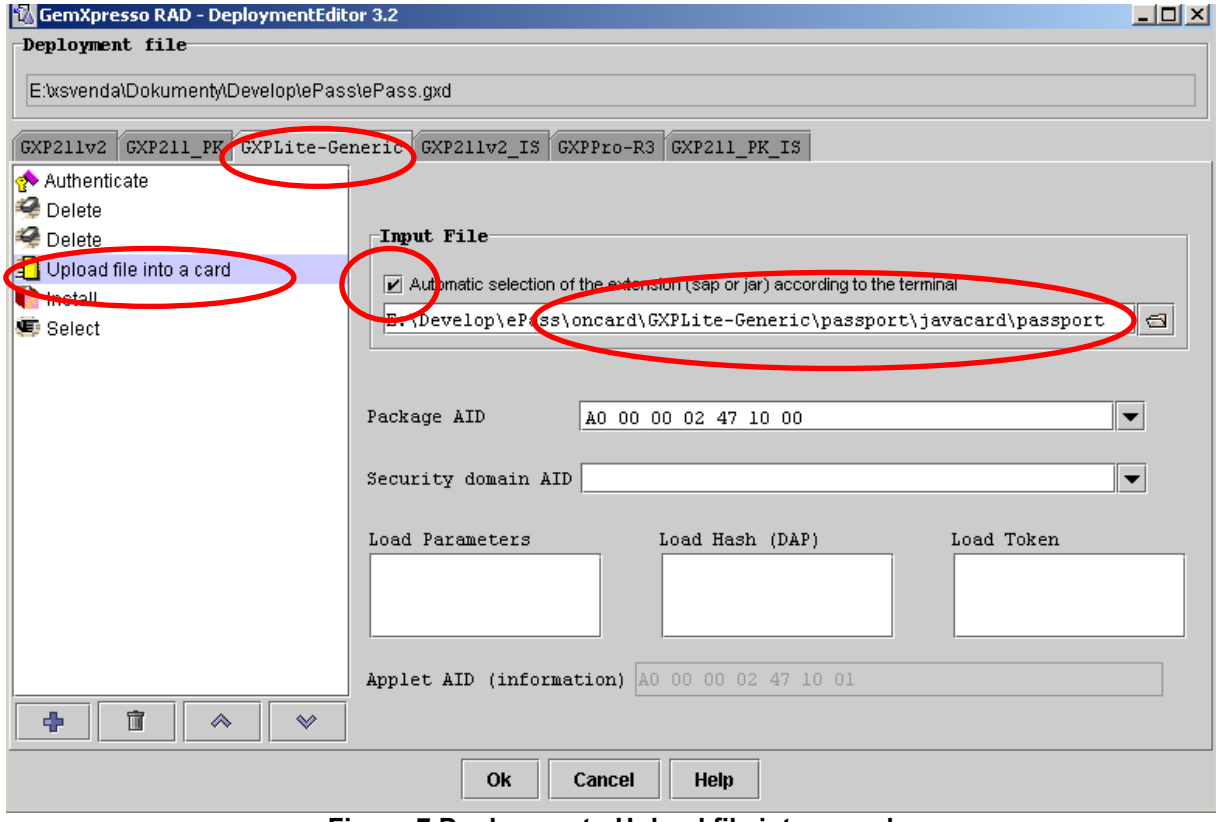

Figure 7 Deployment - Upload file into a card

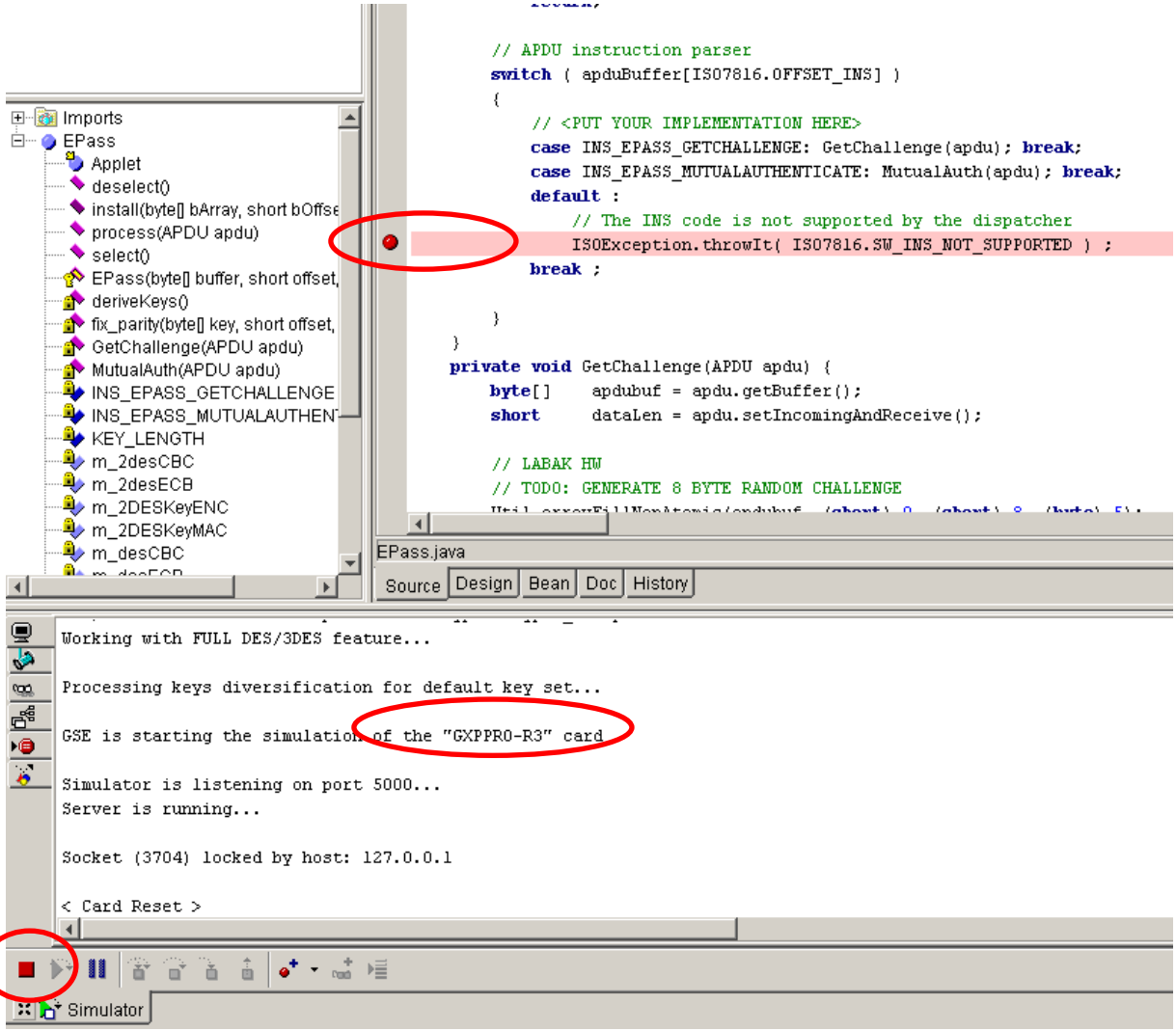

Figure 8 Debug session

List of installed applets (Card Explorer)

Authenticate command must be executed prior to enumeration of installed applets otherwise no applets will be shown. You can use this view to track down installed applets and to obtain AID necessary for applets removal. Remove Applet first and Package second – package with installed instances (applets) cannot be removed.

Important: Do not delete applets or packages starting with A0 xx xx – these applets are system ones and necessary for correct card run!

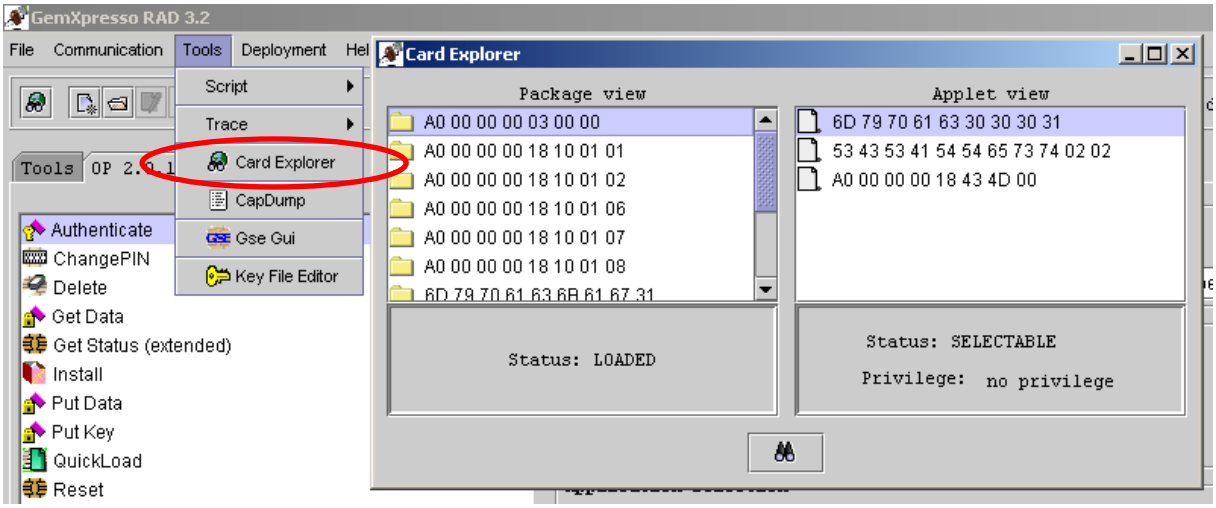

Figure 9 View installed applets and packages

Homework

Create a JavaCard applet capable to sign (reasonably small) file on PC hard-disk by on-card signing only (no key is transferred to PC, hashing of file is performed on smart card). Use ALG_RSA_SHA_PKCS1 signature algorithm. Note, that file size can be bigger than data that can be send in one APDU (~260B), so multiple APDUs may be required. Process should consist:

1. Read first block from file to be signed (~260B).
2. Send block to card.
3. Update signature engine on card.
4. Read next block from file and continue with step 2.
5. When all blocks are read, than produce signature, returns it as output data and display on the screen.

Use P1 of APDU header in step 2. to signalize last block. Use Signature.update() to process incoming data in APDU except the last block. Use Signature.sign() to encrypt last block.

References

[JC222] JavaCard 2.2.2 http://java.sun.com/products/javacard/specs.html