AN4375
Application note

Android application associated to the EVAL-RX95HF board – Firmware upgrade use case example

Introduction

This application note describes how the Android application associated to the EVAL-RX95HF board is structured. This application note intends to explain how to integrate the Firmware upgrade function from a cellular phone to the EVAL-RX95HF board in order to help the developer to speed up the development and software maturation steps of his own Android application.

The Android application associated to the EVAL-RX95HF board is a simple use case demonstration of a half-duplex Near field communication (NFC) between an NFC-enabled cellular phone and the dual memory interface receiver, ST-RX95HF. When a phone is tapped on the EVAL-RX95HF board, a Near field communication is detected. If the EVAL-RX95HF board is configured in tag emulator mode, the application is automatically launched. The user can then upload the new firmware on the board, and request the EVAL-RX95HF to switch to this new firmware if the binary update is successful. In case of an NFC communication loss during the upload, the application has the capability to resume the upload from the latest chunk successfully sent to the evaluation board.

Figure 1. Typical application block diagram

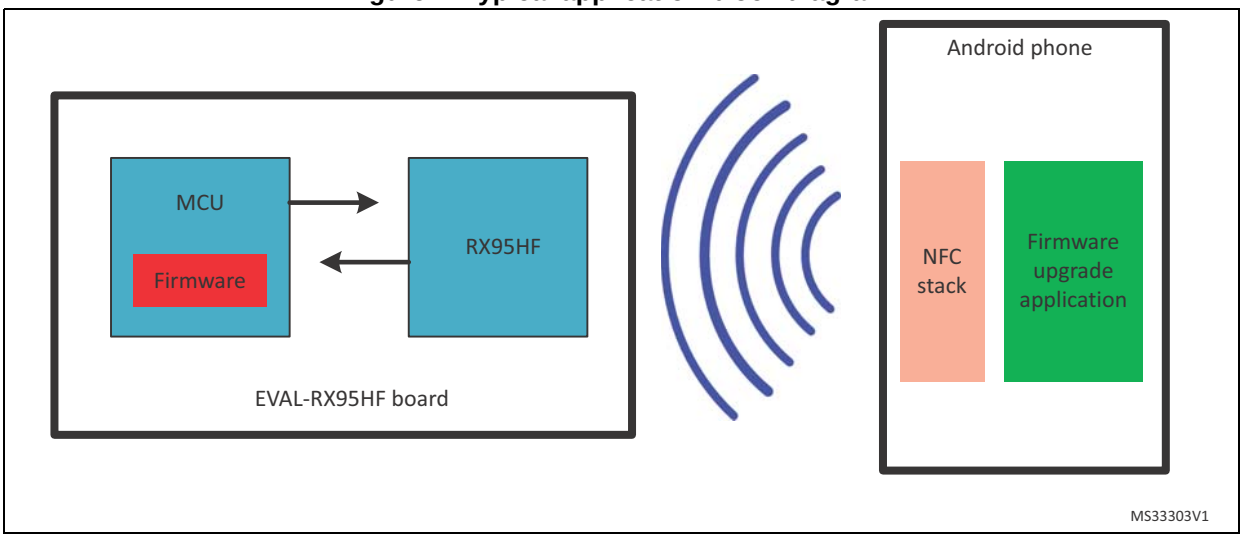


Table 1. Applicable tools and software

Type	Part numbers
Android application package source code	STSW-RX95HF003
Android application binary	STSW-RX95HF002
EVAL-RX95HF board firmware	STSW-RX95HF001
EVAL-RX95HF board	EVAL-RX95HF

Contents

List of tables

List of figures

Reference documents

RX95HF datasheet	Datasheet available on *www.st.com* under *RX95HF*
ISO/IEC 18092-4	
M24SRXX datasheets	Datasheets available on *www.st.com* under *M24SR series*
RX95HFDemo FWU application source code	STSW-RX95HF001 from *www.st.com*
RX95HFDemo FWU application generated javadoc	(available with source code package)
Android reference web site	*http://developer.android.com/reference/packages.html*

Glossary

APK	Android application package file. APK is the file format used to distribute and install application software. Package stores the android application binaries, resources and data.
GUI	Graphic user interface
Javadoc	Javadoc is a facility provided by Java to auto-generate documents from java source code. As Java is an Android code language, this tool is used to automatically document the code (easiness to browse source code with standard internet browser)
NFC	Near field communication

1 Overview

The goal of the application note is to help developers to implement their own NFC Android application solution. it is not intended to explain how to create, build, debug or install Android applications on Android phones. Please browse through Android courses on the web, such as those available at *http://developer.android.com/index.html*.

The source code of the Eclipse project and the associated generated javadoc package are available on ST web site under *STSW-RX95HF003*.

The application is built around two simple screens, showing the two Android activities which compose the application.

The first one is the welcome screen. It is displayed when the application is launched over a user request by selecting the RX95HF demo's application widget provided from the Android application panel. At this stage, the application waits for the Android system NFC intent, which is triggered when the EVAL-RX95HF board is tapped.

Figure 2. Welcome screen

The second screen is displayed once the expected NFC Android system event is triggered on the EVAL-RX95HF board detection which must be in tag emulator mode. This screen opens from the application Welcome screen or from the Android system idle state. Once the application has been installed, it is registered to be automatically launched at the evaluation board detection. The screen displayed offers the capability to select the binary file and to start or resume the Firmware upload process.

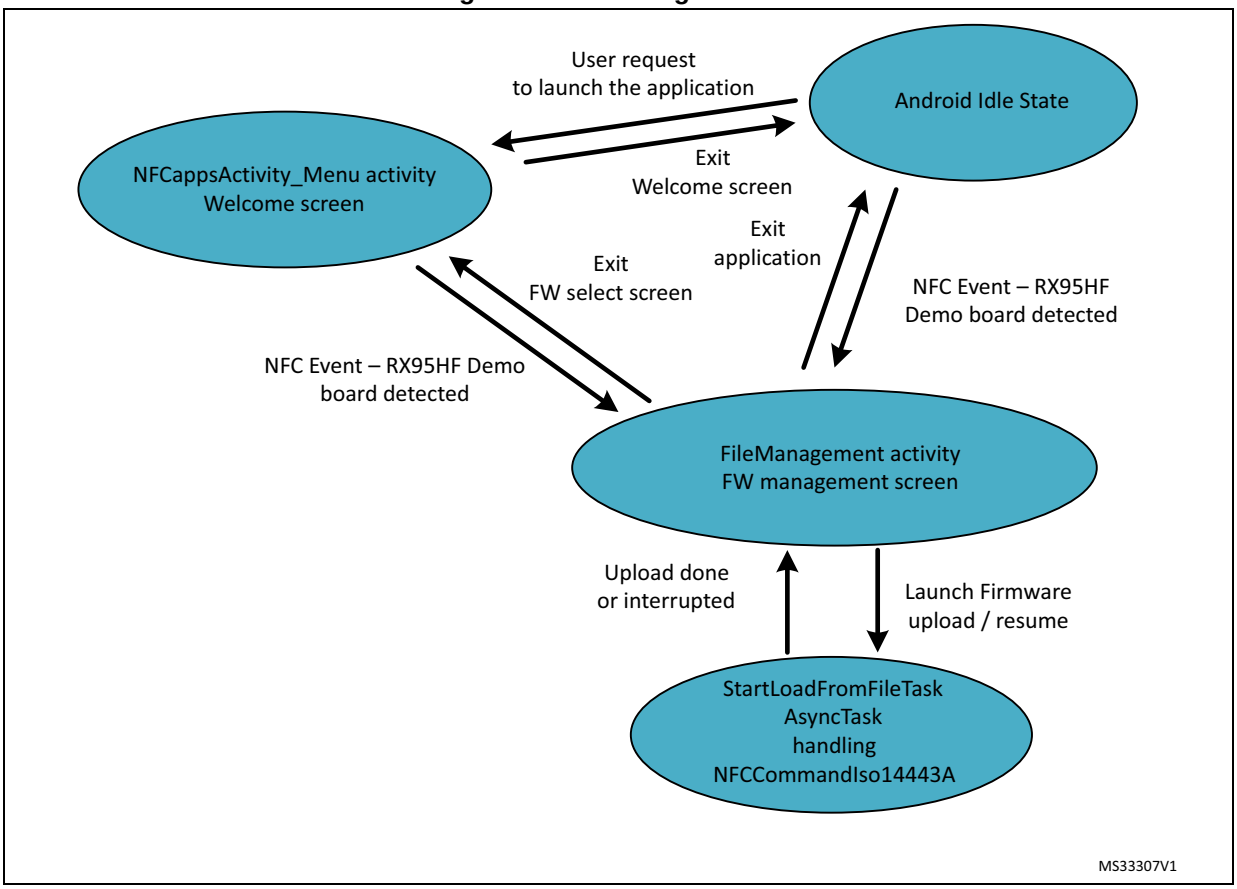
Figure 3. Firmware selection and upload screen

Two activities compose the application: NFCappsActivity_Menu and FileManagement.

1. NFCappsActivity_Menu handles the Welcome screen, the field widget animation and the Android NFC interface initialization.
2. FileManagement activity, triggered on NFC event, controls the upload process. The FileManagement activity implements a light NFC capable object dedicated to the digital protocol communication with the Android native NFC stack.

Here is a logical view of the threads involved during the application life:

Figure 4. Thread logical view



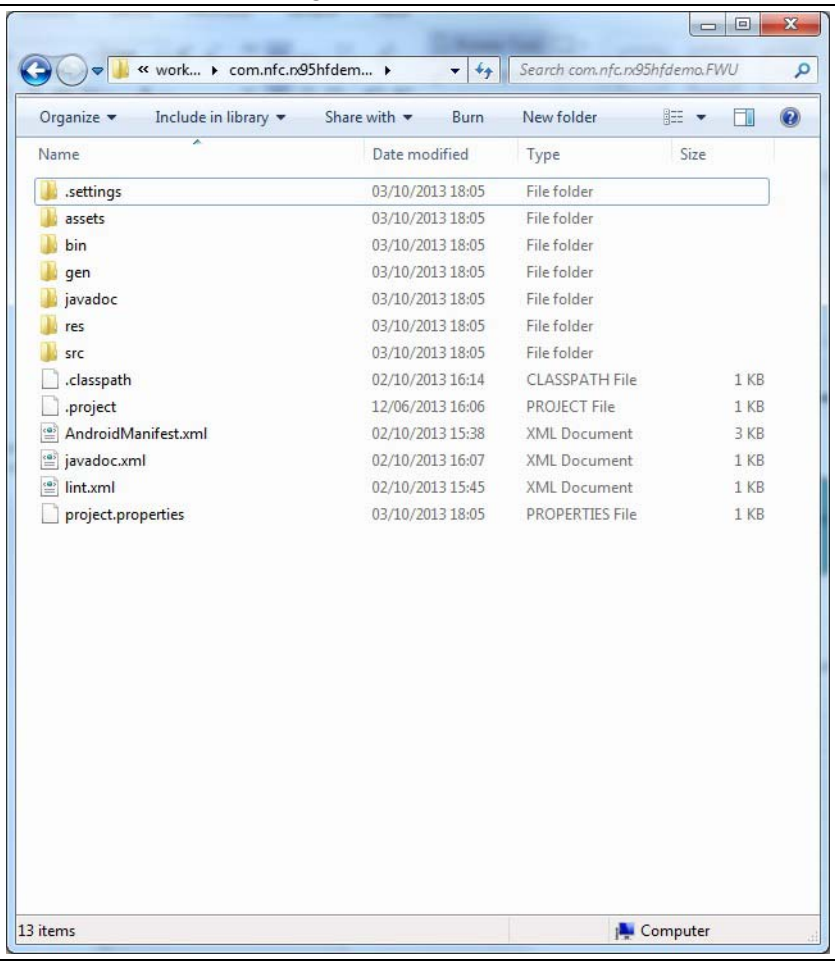
2 Software architecture

2.1 Overview

Like all other Android applications, the firmware upgrade application associated to the EVAL-RX95HF board follows the android application architecture defined by Google:

- AndroidManifest.xml.
  Every application must have an AndroidManifest.xml file (with precisely that name) in its root directory. The manifest presents essential information about the application to the Android system, information the system must have before it can run any of the application's code.
  AndroidManifest.xml declares both NFCappsActivity_Menu and FileManagement activities. NFC filtered intents are defined to be caught when NFCappsActivity_Menu activity is in the active state.
- The ***assets*** folder stores the RX95HF firmware binary example available from *www.st.com* under the root part number *STSW_RX95HF001*). The Firmware is copied, if not yet available in the android application's ***appData*** folder, to ensure the user gets a Firmware to upload when he starts the application for the first time.
- The ***gen*** folder stores the auto-generated files during the application building steps, such as the resource ID file (R.java).
- The ***javadoc*** folder stores the documentation extracted from the source code. *index.html* file under *javadoc* folder is the documentation root which provides a way to browse through the documentation and the java object structures.
- The ***Res/*** directory has subdirectories containing all the resources, such as the image resources, the layout resources, the string resource file and so on. These resources define the default design and content for the Android application. They provide the xml layouts of NFCappsActivity_Menu and FileManagement screens with associated widget ID to be manipulated by the java objects.
- The ***src*** folder contains the java activity files and the *NFCCommandIso14443A* java object.

Figure 5. Root file list



2.2 Importing Android application source code

The application has been developed under Eclipse IDE. In order to parse code, rebuild the application and integrate new functions, the developer has to import the whole source code project.

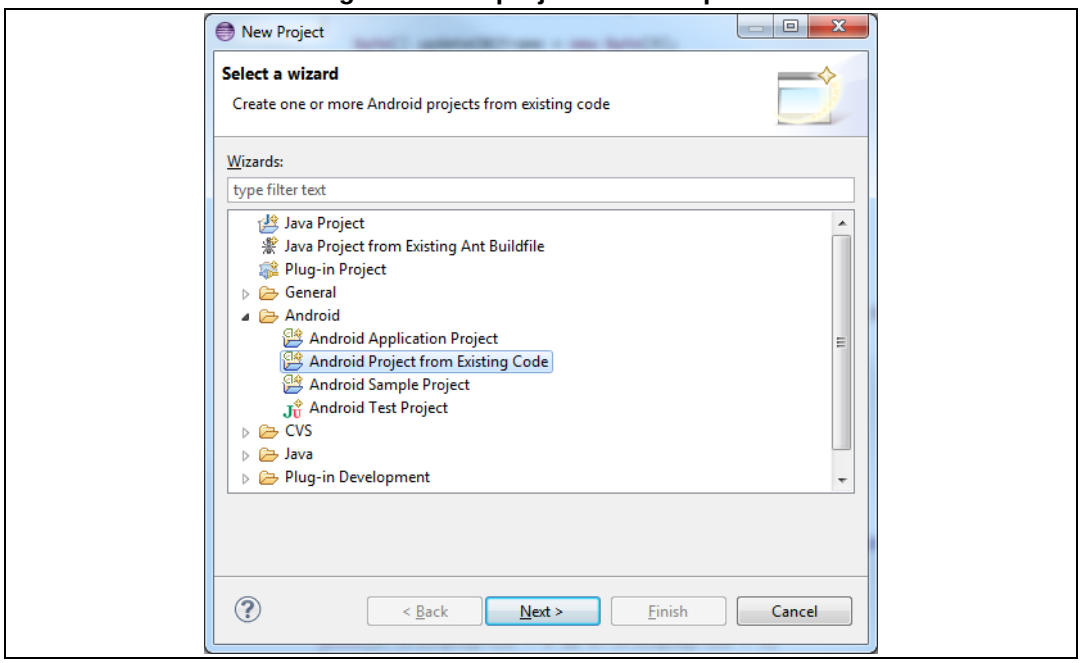
Prerequisites are as follows:

- Eclipse IDE has been installed.
- Android SDK & ADT (Android Development Tool) plug-in has been installed.
  The way to install ADT bundle and Android SDK components is explained on the official Android developer web site.

To go further, retrieve the source code package available on the ST web site (add the logical link of the application) and unzip the package under a temporary folder.

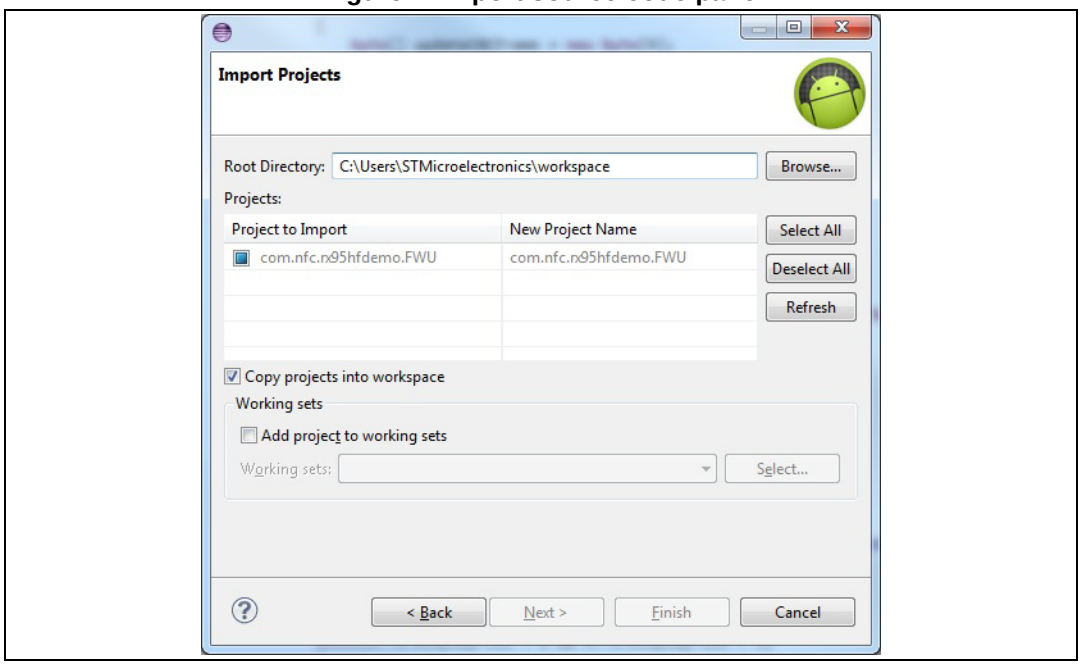
After decompressing the application project zip package, open the installed Eclipse IDE and create a new project (File/New/Project) to display the ***New Project*** panel (see *Figure 6*).

Figure 6. New project creation panel



Developers can select ***Android Project from Existing Code*** then click on ***Next*** to reach the ***Import Projects*** panel (see *Figure 7*).

Figure 7. Import source code panel



Once the root directory of the temporary project location has been chosen and the ***Copy projects into workspace*** option has been checked, the user can click on ***Finish***. The project is then ready to be built and executed on the target phone.

2.3 Main activity class

A ***Main activity class*** is implemented in ***NFCappsActivity_Menu.java***. The ***NFCappsActivity_Menu*** class extends the Android Activity object and must implement at least *OnCreate()* and *onPause()* methods (inheritance concept).

2.3.1 Class members

NFC and Android relay attributes

- private *NfcAdapter mAdapter*;
- private *PendingIntent mPendingIntent*;
- private *IntentFilter[] mFilters*;
- private *String[][] mTechLists*;

These attributes are used to retrieve the Android native NFC handler from the NFC stack provided by the Android system. To get more details on the way to manage NFC components on an Android system, refer to the detailed connectivity NFC API description from http://developer.android.com/guide/topics/connectivity/nfc/index.html

User interface attributes

- Button *btnwww*;
- private *ImageView imgScan*;
- private *int drawableImageID*;
- private *Timer rollImage*.

These fields give the java object the capabilities to manage graphical interfaces. ***NFCappsActivity_Menu*** controls GUI events such as pressing a button to start a new browsing activity, or requesting the graphic user interface system to perform a graphical animation to simulate the NFC field activation.

File System management attributes

- private *dataApplicationDir* string;
- public *FirmwareApplicationDirPath* static string;
- private *FirmwareApplicationDir* file.

File system attributes are declared and used to extract firmware from the ***APK*** Android package to the ***appData*** folder. The Android application ***appData*** folder comes from the Android application structure defined by google. The first time the application is installed, the user can immediately start a Firmware upload to his EVAL-RX95HF board.

2.3.2 NFCappsActivity_Menu.java methods

Activity methods

- public void onCreate(*savedInstanceState* bundle).

  The inheritance from the activity class implies to declare and implement the ***onCreate*** method. This method is defined in the activity class; as NFCappsActivity_Menu.java extends the activity class, the developer must redefine this method. It is called at the activity creation by the Android system. Once the ***super.OnCreate*** is done (***super.OnCreate*** is a way to call an ***onCreate*** method defined in the activity class), the

OnCreate call proceeds to initialize the ***NFCappsActivity_Menu.java*** attributes, as described above. This method checks the availability of the NFC interface with the following request:

pm.hasSystemFeature(PackageManager.FEATURE_NFC)

If the NFC feature is not available, the activity configures a button widget to let the user browse to a specific internet web page. Clicking on it, the user requests the Android system to start a browser activity with the URL during the button configuration. For demonstration purposes, the URL is set to *www.st.com/memories*.

In case the NFC feature is available, the activity ensures the initialization steps. In case the APK embedded firmware is not installed yet, the activity requests to extract the firmware from the ***appData*** folder by calling the *installbinaryfromapk()* method.

The ***OnCreate*** method initializes the Android provided NFC adapter (mAdapter) to get an NFC handler which ensures the command and data exchange with the NFC Android stack. The configuration of the intent category is implemented during the activity creation. The activity is then able to manage such a kind of intent (mPendingIntent).

If the activity has already a pending intent and if it corresponds to the expected intent category configured during the activity creation step, ***NFCappsActivity_Menu*** creates a new intent to broadcast to the *FileManagement* activity and starts the *FileManagement* activity.

Intent intentScan = new Intent(this, FileManagement.class);

startActivity(intentScan);

- protected void onNewIntent(Intent intent)

  The user has to override this method to specify the activity behavior on receiving an intent. In case of an ACTION_TECH_DISCOVERED category received intent, while the activity is in an active state, the activity verifies the tag validity detected from the NFC and sends a first Select Application request.

  checkUID (must also be updated in the source code provided) = DecodeTagUID (Helper.ConvertHexByteArrayToString (tagFromIntent.getId()));

  byte[] selectAppliAnswer = NFCCommandIso14443A.APDUsendSelectAppli (dataDevice.getCurrentTag());

  If both commands succeed, the activity creates a new intent and sends it to the FileManagement activity:

  Intent intentScan = new Intent(this, FileManagement.class);
  startActivity(intentScan);

- protected void onResume()

  The activity method called on activity restart declares the current activity as a grabber for all incoming Android system events of mPendingIntent category, as initialized in the OnCreate() method.

- protected void onPause()

  The activity method called on activity pause unregisters the current activity as a manager of mPendingIntent intent category.

NFC relay methods

- public boolean DecodeTagUID (String TagUID)

  This method is called, at the application level, to store the UID tag. A parsing function can be introduced here to verify that the tag involved in the NFC field is the expected one.

APK embedded relay methods

- private void copyFile(InputStream in, OutputStream out) throws IOException
- private void copyFirmwares()
- private void installbinaryfromapk() throws NameNotFoundException

2.4 File management activity class

The *FileManagement* (*FileManagement.java* file) class extends the activity class. Then, the same methods overridden in the *NFCappsActivity_Menu* class must also be overridden. This class handles user input events, user interface updates and the Firmware upload process. The *FileManagement* object uses the *StartLoadFromFileTask* object to handle the upload process by itself.

2.4.1 Class members

Status members

In order to know which state the activity reaches, the latest status of the upload request is stored using the following attributes:

- public static boolean *statusErrorWrite* = *false*;
- public static boolean *statusErrorWrite_continue* = *false*;

User interface members

As the activity gets its own android graphical surface view, the class needs to have some specific user interface members in order to update the graphical widget state and to manage the user action linked to it.

- Button *buttonWriteFromFile*

  This button is used to handle events coming from the user interface system and to launch a write action. If *buttonWriteFromFile* is pressed, the *Filemanagement* object requests the *StartLoadFromFileTask* object to begin the upload process.

- Button *buttonContinueWriteFromFile*

  This button is used to handle the user request to start the resume process. Then, the *Filemanagement* object delegates this task to the *StartLoadFromFileTask*. This button is only activated if there is something to resume (i.e. a previous upload had been interrupted).

- private *TextView* selection

  The selection member is used to manage the firmware upload selection. This widget is populated using the *Filemanagement*'s File List member listed below.

NFC linked member

As the *Filemanagement* activity requests NFC data exchange using the StartLoadFromFileTask, the corresponding object class needs to store the NFC relay information.

The definition of the following members is the same as the one used in the main activity class:

- private *NfcAdapter mAdapter*;
- private *PendingIntent mPendingIntent*;
- private *IntentFilter[] mFilters*;
- private *String[][] mTechLists*;
- public *NFCCommandIso14443A uploaderHandler = null*.

  The *NFCCommandIso14443A* class attribute is declared here. This object performs NFC communications and sends binary chunks of the file to upload.
- *StartLoadFromFileTask*

  Extend the *AsyncTask* which handles the digital communication protocol to control the Firmware upload.

  In order to keep the user interface active and to avoid an application freeze feeling, an *AsyncTask* object is used. This Android object is detailed on http://developer.android.com/reference/android/os/AsyncTask.html.

  All exchange requests to the EVAL-RX95HF board are done by delegating the upload activity to this object.
- private *long CRC =0*;

  The CRC member is used to store the CRC of the file currently uploaded. Once the upload is done, the CRC is sent to the EVAL-RX95HF board. The EVAL-RX95HF embedded firmware can then verify that it received the complete firmware binary before starting it.

File List managed member

The firmware files to be uploaded can be stored in two different folders. An APK embedded firmware delivered with the application is copied during the application installation in the Android application's ***appData*** folder. The ability to copy the user specific firmware using a PC and a USB connection on MMC has also been implemented. The user can plug the Android phone to a PC using a simple USB cable and then copy his own firmware in the */Download/ fwrx95hf* directory on the Android phone mounted mass storage seen from the PC file explorer.

The full list of files that could be uploaded is then displayed using a listview widget. The user can select the file he wants to upload by expanding the listview object.

The following members are used to implement this File List management.

- private byte[] bufferFile = null;
- private boolean FileError = false;
- private EditText textProcessStatus;
- public static File [] firmwarelist = null;
- public static int nbFWinAppDataDir=0;
- public static File [] firmwareSDlist = null;
- String [] listFWFileName = null;
- public static File firmwareRepo = null;
- public static File firmwareSDRepo = null;
- public static int currentFw2UploadId;
- private string fwextMemDir = "fwrx95hf";

2.4.2 File management method members

Activity methods

- protected void onCreate(Bundle savedInstanceState)

  Called during an activity creation, this method instantiates the *NFCCommandIso14443A* command handler object to perform a digital communication with the EVAL-RX95HF board. It retrieves the current NFC Android stack handler to manage Android native messages (intents). This method also initializes the GUI widget belonging to the Activity's view, such as the firmware file list and notification widgets. The method terminates by calling the *initListener()* method which configures the **Upload FW** and **Resume** buttons.
- protected void onResume()

  This method is called when the activity is resumed and is registered as an NFC intent receiver.
- protected void onPause()

  This method is called when the activity is paused and is not registered as an NFC intent receiver.

FileManagement specific methods

- private void initListener()

  This method initializes the buttons to start and resume the upload. When the button is pressed, the expected action is launched by dispatching the request to the *StartLoadFromFileTask* object.
- public void onItemSelected(AdapterView<?> parent,View v, int position, long id)

  This method is called when the user selects the file from the listview he wants to upload. If on call, fileID is stored. fileID is used to retrieve the full path of the binary file to upload when the request is delegated to the *StartLoadFromFileTask* object.
- public void onNothingSelected(AdapterView<?> parent)

  The empty method but must be overridden.
- public boolean Verify_RX95_UID (String rx95_UID_answer)

  NFC Helper verifies the tag detected in the field. This method can be easily improved to parse the full UID tag provided by the NFC Android stack.
- StartLoadFromFileTask

  This internal class which extends AsyncTask (details can be found on http://developer.android.com/reference/android/os/AsyncTask.html) is used to start the pure upload activity.

- protected void StartLoadFromFileTask.onPreExecute()

  This method is called before launching the process dedicated to the AsyncTask object and on a button press (Upload FW and Resume). This method retrieves the fileID of the firmware to upload, rebuilds the Firmware path, initializes the NFCCommandIso14443A uploaderHandler with the buffer, and the buffer size to send. Then the doInBackground method is called.

- protected Void StartLoadFromFileTask.doInBackground(Void... params)

  It starts the upload process by calling *APDUsendUpdateBinaryNew* from NFCCommandIso14443A class with the right parameters (resume status, current handled NFC tag, and computed CRC file).

- protected void StartLoadFromFileTask.onPostExecute(final Void unused)

  This method is called when the upload process is stopped (i.e. the upload is successful or interrupted). The goal of this method is to check the upload result (type NFCCommandStatus defined in NFCCommandIso14443A) and to update the user interface of the FileManagement activity accordingly.

2.5 NFCCommandIso14443A class

NFCCommandIso14443A (NFCCommandIso14443A.java file) class defines the object which controls the digital protocol communication with the RX95HF through the Android NFC stack. As the set of commands to send (see next section) is light, this object has only one method to perform the NFC Firmware upload.

2.5.1 Class members

- public enum NFCCommandStatus

  {

  CMD_OK,
  CMD_SELECTAPPLIERR,
  CMD_UPDATESIZEINFERR,
  CMD_UPLOADBUFFEREXCEPTIONERR,
  CMD_SENDCHUNCKERR,
  CMD_CLOSEFILEMSGERR,
  CMD_CRCMSGINGERR,
  CMD_LAUNCHACTIONERR,
  CMD_TAGUNREACHABLEERR,
  CMD_STATUSUNKNOWN

  }

  The error enumeration is used by the caller to update an object status or a graphic user interface, and to notify the user of the upload progress.
- public static int *lastChunkIDsent*;

  Updated when ACK is received from RX.
- public static int *lastBuffOffsetSent*;

  Updated when ACK is received from RX.
- public static int *bufferSize*;

  Size in bytes of the buffer to send.
- public static byte [] bufferData;

  Raw data to send.
- public static int chunkSize;

  The chunk size to send to the EVAL-RX95HF must be initialized according to the capabilities of the EVAL-RX95HF (MLe field).
- public static int nbChunk;

  Number of chunks to send (BuffSize/chunkSize).

NFCCommandIso14443A method members

- public *NFCCommandIso14443A()*

  NFCCommandIso14443A object-oriented programming concept initializes the object attributes and, more specifically, the chunk size.

- public void init(byte [] abufferData)

  This init method must be called with a new buffer as parameter every time a new firmware is selected.

- public static byte[] *APDUsendSelectAppli (Tag myTag)*

  This function member requests to send an APDU send select application. The myTag argument is extracted from the intent triggered on the dual interface EEPROM device detection by the NFC Android stack.

- public *NFCCommandStatus APDUsendUpdateBinaryNew* (boolean resume, Tag myTag,long CRC)

  This function member is called to execute the firmware upload process. Depending on the last upload status, the command starts a new firmware upload or resumes the previous one, if it has been interrupted. The sequence of commands to send must follow the command set defined in the tag emulator firmware (see EVAL-RX95HF board firmware user manual from *www.st.com*).

  Following the status of each command sent, this member returns a specific *NFCCommandStatus* error to let the caller layer decide if a resume is necessary or not. The command sequence is detailed in *Section 3*.

3 Digital protocol description

This section describes the simple digital protocol used by the *NFCCommandIso14443A* object to ensure the firmware (or binary file) upload task.

The digital protocol is based on the 7816 standard part 4 *(Organization, security and commands for interchange)* reused in 14443-4 document.

3.1 APDU Overview

As described in ISO/IEC 7816-4, an application protocol data unit (APDU) contains either a command message or a response message, sent from the interface device (i.e. a phone) to the card (EVAL-RX95HF) or conversely.

In a command-response pair, the command message and the response message can contain data, thus inducing four cases:

Table 2. Command and response data

Case	Command data	Expected response data
1	No data	No data
2	No data	Data
3	Data	No data
4	Data	Data

The command APDU consists of:

- a mandatory header of 4 bytes (CLA INS P1 P2),
- a conditional body of a variable length.

Table 3. Header and body

Header	Body
CLA INS P1 P2	[Lc field] [Data field] [Le field]

3.1.1 APDU command content details

The APDU command sent by the phone is defined as in *Table 4*.

Table 4. APDU command sent by the phone

Code	Name	Length	Description
CLA	Class	1	Class of instruction
INS	Instruction	1	Instruction code
P1	parameter 1	1	Instruction parameter 1
P2	parameter 2	1	Instruction parameter 2

Table 4. APDU command sent by the phone

Code	Name	Length	Description
Lc field	Length	variable 1 or 3	Number of bytes present in the data field of the command
Data field	Data	variable = Lc	String of bytes sent in the data field of the command
Le field	Length	variable 1 or 3	Maximum number of bytes expected in the data field of the response to the command

3.1.2 APDU response content details

The APDU response content sent by the EVAL-RX95HF is defined as in *Table 5*.

Table 5. APDU response content

Code	Name	Length	Description
Data field	Data	variable = Lr	String of bytes received in the data field of the response
SW1	Status byte 1	1	Command processing status
SW2	Status byte 2	1	Command processing qualifier

3.2 EVAL-RX95HF digital protocol APDU description

In order to build a specific firmware upload digital protocol, two classes of instructions are used in the EVAL-RX95HF firmware upgrade application

3.2.1 Class and Instruction code

- 0x00: structure and coding of command and response according to 7816-4 / No SM or no SM indication class.
  In this case, the instructions used are defined by the following codes:
  - 0xA4: Select File
  - 0xD6: Update binary
- 0xA2: unless otherwise specified by the application context, structure and coding of command and response according to 7816-4 / No SM or no SM indication associated to channel 2 format. This class is associated to the following proprietary instruction codes:
  - 0x41 - Send a buffer size update command
  - 0x42 - Send a CRC update command
  - 0xFF - Send a start request

3.2.2 Detailed commands

Table 6. Select the application frame

CLA	INS	P1	P2	LC Field	Data	Data	Data	Data
0x00	0xA4	0x04	0x00	0x10	0xF0	0x02	0x46	0x57
					0x55	0x5F	0x58	0x58
					0x4F	0x5F	0x76	0x30
					0x00	0x00	0x00	0x00

Table 7. Buffer size update

CLA	INS	P1	P2	LC Field	Data	Data	Data	Data
0xA2	0x41	0x80	0x00	0x02	0xXX	0xXX		

Table 8. Update by chunk

CLA	INS	P1	P2	LC Field	Data	Data	Data	Data
0x00	0xD6	0xX1	0xX1	0xX3	data[0]	data[1]	data[3]	...

Table 9. Update remaining data

CLA	INS	P1	P2	LC Field	Data	Data	Data	Data
0x00	0xD6	0xX1	0xX1	0xX3	data[0]	data[1]	data[3]	…

Table 10. Close file

CLA	INS	P1	P2	LC Field	Data
0x00	0xD6	0xFF	0xFF	0x01	0xAA

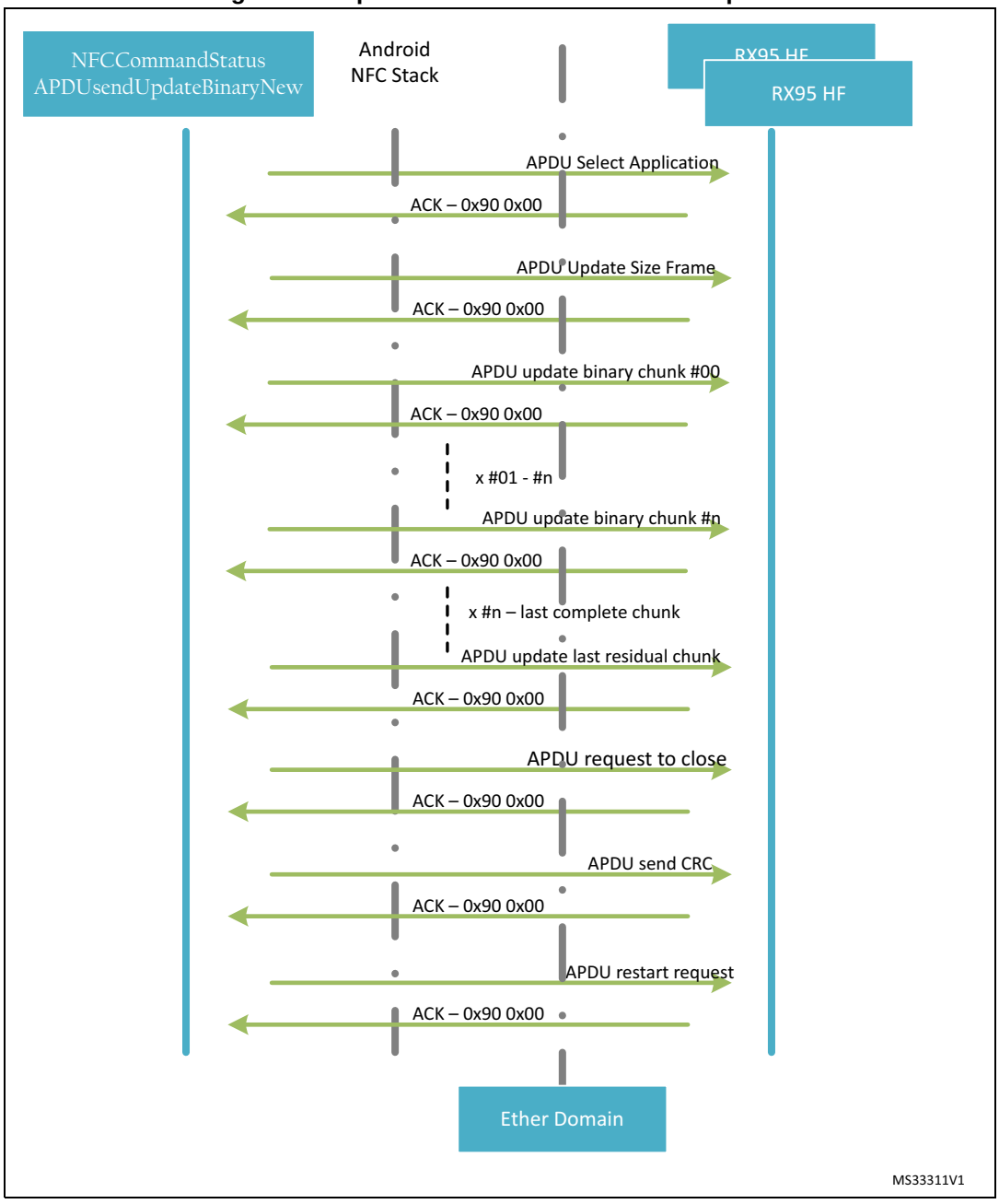
Table 11. Send CRC

CLA	INS	P1	P2	LC Field	Data	Data	Data	Data
0xA2	0x42	0x00	0x00	0x04	0xCRC0	0xCRC1	0xCRC2	0xCRC3

Table 12. Send start request

CLA	INS	P1	P2	LC Field	Data	Data
0xA2	0xFE	0x80	0x00	0x02	0x70	0x69

Figure 8. Sequence chart of a full firmware upload



4 Revision history

Table 13. Document revision history

Date	Revision	Changes
05-Dec-2013	1	Initial release.

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

ST PRODUCTS ARE NOT DESIGNED OR AUTHORIZED FOR USE IN: (A) SAFETY CRITICAL APPLICATIONS SUCH AS LIFE SUPPORTING, ACTIVE IMPLANTED DEVICES OR SYSTEMS WITH PRODUCT FUNCTIONAL SAFETY REQUIREMENTS; (B) AERONAUTIC APPLICATIONS; (C) AUTOMOTIVE APPLICATIONS OR ENVIRONMENTS, AND/OR (D) AEROSPACE APPLICATIONS OR ENVIRONMENTS. WHERE ST PRODUCTS ARE NOT DESIGNED FOR SUCH USE, THE PURCHASER SHALL USE PRODUCTS AT PURCHASER’S SOLE RISK, EVEN IF ST HAS BEEN INFORMED IN WRITING OF SUCH USAGE, UNLESS A PRODUCT IS EXPRESSLY DESIGNATED BY ST AS BEING INTENDED FOR “AUTOMOTIVE, AUTOMOTIVE SAFETY OR MEDICAL” INDUSTRY DOMAINS ACCORDING TO ST PRODUCT DESIGN SPECIFICATIONS. PRODUCTS FORMALLY ESCC, QML OR JAN QUALIFIED ARE DEEMED SUITABLE FOR USE IN AEROSPACE BY THE CORRESPONDING GOVERNMENTAL AGENCY.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.
Information in this document supersedes and replaces all information previously supplied.
The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2013 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com