Appendix A
Google's Secure Element API

This appendix lists the interface definition of Google's proprietary secure element API as included in version 4.2.1 of the Android system. See Sect. 6.2.4.1 for a detailed analysis.

A.1 Class NfcAdapterExtras

```
/*
 * Copyright (C) 2011 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License. You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed under the License is distributed on
 * an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied. See the License for the
 * specific language governing permissions and limitations
 * under the License.
 */

package com.android.nfc_extras;

public final class NfcAdapterExtras {
  /**
   * Broadcast Action: RF field ON has been detected.
   * This is an unreliable signal, and will be removed.
   */
public static final String ACTION_RF_FIELD_ON_DETECTED =
        "com.android.nfc_extras.action.RF_FIELD_ON_DETECTED";

  /**
   * Broadcast Action: RF field OFF has been detected.
   * This is an unreliable signal, and will be removed.
   */
  public static final String ACTION_RF_FIELD_OFF_DETECTED =
        "com.android.nfc_extras.action.RF_FIELD_OFF_DETECTED";
```


M. Roland, *Security Issues in Mobile NFC Devices*, T-Labs Series in Telecommunication Services, DOI 10.1007/978-3-319-15488-6

```
34
35
36      /**
37       * Get the NfcAdapterExtras for the given NfcAdapter.
38       */
39      public static NfcAdapterExtras get(NfcAdapter adapter) { ... }
40
41
42      /**
43       * Immutable data class that describes a card emulation route.
44       */
45      public final static class CardEmulationRoute {
46        /**
47         * Card Emulation is turned off on this NfcAdapter.
48         */
49        public static final int ROUTE_OFF = 1;
50
51        /**
52         * Card Emulation is routed to nfcEe only when the screen
53         * is on, otherwise it is turned off.
54         */
55        public static final int ROUTE_ON_WHEN_SCREEN_ON = 2;
56
57        /**
58         * A route such as ROUTE_OFF or ROUTE_ON_WHEN_SCREEN_ON.
59         */
60        public final int route;
61
62        /**
63         * The NFcExecutionEnvironment that Card Emulation is
64         * routed to.
65         */
66        public final NfcExecutionEnvironment nfcEe;
67
68
69        public CardEmulationRoute(
70              int route, NfcExecutionEnvironment nfcEe) { ... }
71      }
72
73
74      /**
75       * Get the current routing state of the secure element.
76       */
77      public CardEmulationRoute getCardEmulationRoute() { ... }
78
79      /**
80       * Set the routing state of the secure element.
81       */
82      public void setCardEmulationRoute(
83            CardEmulationRoute route) { ... }
84
85
86      /**
87       * Get the NfcExecutionEnvironment for the embedded secure
88       * element.
89       */
90      public NfcExecutionEnvironment getEmbeddedExecutionEnvironment(
91            ) { ... }
92
93
94      /**
95       * Authenticate the client application.
96       * Some implementations of NFC Adapter Extras may require
97       * applications to authenticate with a token, before using
98       * other methods.
99       * This method is not used on Nexus S/Galaxy Nexus.
100      */
```

```
    public void authenticate(byte[] token) { ... }

    /**
     * Returns the name of this adapter's driver.
     */
    public String getDriverName() { ... }
}
```

A.2 Class NfcExecutionEnvironment

```
/*
 * Copyright (C) 2011 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License. You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed under the License is distributed on
 * an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied. See the License for the
 * specific language governing permissions and limitations
 * under the License.
 */

package com.android.nfc_extras;

import java.io.IOException;

public class NfcExecutionEnvironment {
    /**
     * Broadcast Action: An ISO-DEP AID was selected.
     */
    public static final String ACTION_AID_SELECTED =
            "com.android.nfc_extras.action.AID_SELECTED";
    /**
     * Mandatory byte array extra field in ACTION_AID_SELECTED.
     */
    public static final String EXTRA_AID =
            "com.android.nfc_extras.extra.AID";

    /**
     * Broadcast action: A filtered APDU was received.
     */
    public static final String ACTION_APDU_RECEIVED =
            "com.android.nfc_extras.action.APDU_RECEIVED";
    /**
     * Mandatory byte array extra field in ACTION_APDU_RECEIVED.
     */
    public static final String EXTRA_APDU_BYTES =
            "com.android.nfc_extras.extra.APDU_BYTES";

    /**
     * Broadcast action: An EMV card removal event was detected.
     */
    public static final String ACTION_EMV_CARD_REMOVAL =
            "com.android.nfc_extras.action.EMV_CARD_REMOVAL";

```

```
51    /**
52     * Broadcast action: An adapter implementing MIFARE Classic
53     * via card emulation detected that a block has been accessed.
54     */
55    public static final String ACTION_MIFARE_ACCESS_DETECTED =
56          "com.android.nfc_extras.action.MIFARE_ACCESS_DETECTED";
57    /**
58     * Optional integer extra field in ACTION_MIFARE_ACCESS
59     * _DETECTED that provides the block number being accessed.
60     */
61    public static final String EXTRA_MIFARE_BLOCK =
62          "com.android.nfc_extras.extra.MIFARE_BLOCK";
63
64
65    /**
66     * Open the NFC Execution Environment on its contact
67     * interface.
68     */
69    public void open() throws IOException { ... }
70
71    /**
72     * Close the NFC Execution Environment on its contact
73     * interface.
74     */
75    public void close() throws IOException { ... }
76
77    /**
78     * Send raw commands to the NFC Execution Environment
79     * and receive the response.
80     */
81    public byte[] transceive(byte[] in) throws IOException { ... }
82  }
```

Appendix B
Modifications to Google's Secure Element API Library

This appendix lists a modified version of Google's proprietary secure element API that outputs debug information to the Android debug log. This version is based on `com.android.nfc_extras` as included in Android 4.1.1. Most comments were removed from these listings.

B.1 Class NfcAdapterExtras

```
1  /*
2   * Copyright (C) 2011 The Android Open Source Project
3   * Modifications (debug output): (C) 2012 Michael Roland
4   *
5   * Licensed under the Apache License, Version 2.0 (the
6   * "License"); you may not use this file except in compliance
7   * with the License. You may obtain a copy of the License at
8   *
9   *      http://www.apache.org/licenses/LICENSE-2.0
10  *
11  * Unless required by applicable law or agreed to in writing,
12  * software distributed under the License is distributed on
13  * an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
14  * KIND, either express or implied. See the License for the
15  * specific language governing permissions and limitations
16  * under the License.
17  */
18
19 package com.android.nfc_extras;
20
21 import java.util.HashMap;
22
23 import android.content.Context;
24 import android.nfc.INfcAdapterExtras;
25 import android.nfc.NfcAdapter;
26 import android.os.RemoteException;
27 import android.util.Log;
28 import java.lang.reflect.Method;
```


M. Roland, *Security Issues in Mobile NFC Devices*, T-Labs Series in Telecommunication Services, DOI 10.1007/978-3-319-15488-6

```
37 public final class NfcAdapterExtras {
38   private static final String TAG = "NfcAdapterExtras";
```

```
48   public static final String ACTION_RF_FIELD_ON_DETECTED =
```

```
59  public static final String ACTION_RF_FIELD_OFF_DETECTED =
60     "com.android.nfc_extras.action.RF_FIELD_OFF_DETECTED";
```

```
65   private static INfcAdapterExtras sService;
66   private static final CardEmulationRoute ROUTE_OFF =
67         new CardEmulationRoute(CardEmulationRoute.ROUTE_OFF,
68                                null);
```

```
71   private static final HashMap<NfcAdapter, NfcAdapterExtras>
72         sNfcExtras = new HashMap();
73
74   private final NfcExecutionEnvironment mEmbeddedEe;
75   private final CardEmulationRoute mRouteOnWhenScreenOn;
76
77   private final NfcAdapter mAdapter;
78   final String mPackageName;
79
80   /** get service handles */
81   private static void initService(NfcAdapter adapter) {
82     try {
83       Method getNfcAdapterExtrasInterface =
84             NfcAdapter.class.getMethod(
85                   "getNfcAdapterExtrasInterface");
86       final INfcAdapterExtras service = (INfcAdapterExtras)
87             getNfcAdapterExtrasInterface.invoke(adapter);
88       if (service != null) {
89         // Leave stale rather than receive a null value.
90         sService = service;
91       }
92     } catch (Exception e) {}
93   }
```

```
104   public static NfcAdapterExtras get(NfcAdapter adapter) {
105     Context context = null;
106     try {
107       Method getContext =
108             NfcAdapter.class.getMethod("getContext");
109       context = (Context)getContext.invoke(adapter);
110     } catch (Exception e) {}
111     if (context == null) {
112       throw new UnsupportedOperationException(
113             "You must pass a context to your NfcAdapter to use the
                  NFC extras APIs");
114     }
115
116     synchronized (NfcAdapterExtras.class) {
117       if (sService == null) {
118         initService(adapter);
119       }
120       NfcAdapterExtras extras = sNfcExtras.get(adapter);
121       if (extras == null) {
122         extras = new NfcAdapterExtras(adapter);
```

```
123           sNfcExtras.put(adapter,  extras);
124         }
125         return extras;
126       }
127     }
128
129     private NfcAdapterExtras(NfcAdapter adapter) {
130       mAdapter = adapter;
131       String packageName = null;
132       try {
133         Method getContext =
134               NfcAdapter.class.getMethod("getContext");
135         packageName = ((Context)getContext.invoke(adapter))
136               .getPackageName();
137       } catch (Exception e) {}
138       mPackageName = packageName;
139       mEmbeddedEe = new NfcExecutionEnvironment(this);
140       mRouteOnWhenScreenOn = new CardEmulationRoute(
141             CardEmulationRoute.ROUTE_ON_WHEN_SCREEN_ON,
142             mEmbeddedEe);
143     }
```

```
148   public final static class CardEmulationRoute {
```

```
153     public static final int ROUTE_OFF = 1;
```

```
159     public static final int ROUTE_ON_WHEN_SCREEN_ON = 2;
```

```
164     public final int route;
```

```
170     public final NfcExecutionEnvironment nfcEe;
171
172     public CardEmulationRoute(int route,
173                               NfcExecutionEnvironment nfcEe) {
174       if (route == ROUTE_OFF && nfcEe != null) {
175         throw new IllegalArgumentException(
176               "must not specifiy a NFC-EE with ROUTE_OFF");
177       } else if (route != ROUTE_OFF && nfcEe == null) {
178         throw new IllegalArgumentException(
179               "must specifiy a NFC-EE for this route");
180       }
181       this.route = route;
182       this.nfcEe = nfcEe;
183     }
184   }
```

```
189   void attemptDeadServiceRecovery(Exception e) {
190     Log.e(TAG,
191           "NFC Adapter Extras dead - attempting to recover");
192     try {
193       Method attemptDeadServiceRecovery =
194             NfcAdapter.class.getMethod(
195                   "attemptDeadServiceRecovery",
196                   Exception.class);
197       attemptDeadServiceRecovery.invoke(mAdapter, e);
198     } catch (Exception ee) {}
```

```
    initService(mAdapter);
  }

  INfcAdapterExtras getService() {
    return sService;
  }
```

```
  public CardEmulationRoute getCardEmulationRoute() {
    try {
      int route = sService.getCardEmulationRoute(mPackageName);
      Log.d(TAG,
            "getCardEmulationRoute() for " +
            ((mPackageName != null) ? mPackageName : "[null]") +
            " = " + route);
      return route == CardEmulationRoute.ROUTE_OFF ?
            ROUTE_OFF :
            mRouteOnWhenScreenOn;
    } catch (Exception e) {
      attemptDeadServiceRecovery(e);
      return ROUTE_OFF;
    }
  }
```

```
  public void setCardEmulationRoute(CardEmulationRoute route) {
    try {
      Log.d(TAG,
            "setCardEmulationRoute(" + route.route + ") for " +
            ((mPackageName != null) ? mPackageName : "[null]"));
      sService.setCardEmulationRoute(mPackageName, route.route);
    } catch (Exception e) {
      attemptDeadServiceRecovery(e);
    }
  }
```

```
  public NfcExecutionEnvironment getEmbeddedExecutionEnvironment() {
    Log.d(TAG,
          "getEmbeddedExecutionEnvironment() for " +
          ((mPackageName != null) ? mPackageName : "[null]"));
    return mEmbeddedEe;
  }

  /**
    * Convert a byte array into its hexadecimal string form.
    * @param b  Byte array.
    * @return   Hexadecimal string representation.
    */
  private static String convertByteArrayToHexString(byte[] b) {
    if (b != null) {
      StringBuilder s = new StringBuilder(2 * b.length);

      for (int i = 0; i < b.length; ++i) {
        final String t = Integer.toHexString(b[i]);
        final int l = t.length();
        if (l > 2) {
          s.append(t.substring(l - 2));
        } else {
          if (l == 1) {
            s.append("0");
          }
          s.append(t);
        }
```

```
285         }
286
287         return s.toString();
288       } else {
289         return "";
290       }
291     }
```

```
302     public void authenticate(byte[] token) {
303       try {
304         Log.d(TAG,
305               "authenticate() for " +
306               ((mPackageName != null) ? mPackageName : "[null]") +
307               ": " + convertByteArrayToHexString(token));
308         sService.authenticate(mPackageName, token);
309       } catch (Exception e) {
310         attemptDeadServiceRecovery(e);
311       }
312     }
313 }
```

B.2 Class NfcExecutionEnvironment

```
1  /*
2   * Copyright (C) 2011 The Android Open Source Project
3   * Modifications (debug output): (C) 2012 Michael Roland
4   *
5   * Licensed under the Apache License, Version 2.0 (the
6   * "License"); you may not use this file except in compliance
7   * with the License. You may obtain a copy of the License at
8   *
9   *       http://www.apache.org/licenses/LICENSE-2.0
10  *
11  * Unless required by applicable law or agreed to in writing,
12  * software distributed under the License is distributed on
13  * an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
14  * KIND, either express or implied. See the License for the
15  * specific language governing permissions and limitations
16  * under the License.
17  */
18
19 package com.android.nfc_extras;
20
21 import android.os.Binder;
22 import android.os.Bundle;
23 import android.os.RemoteException;
24 import android.util.Log;
25
26 import java.io.IOException;
27
28 public class NfcExecutionEnvironment {
29   private static final String TAG="NfcExecutionEnvironment";
30
31   private final NfcAdapterExtras mExtras;
32   private final Binder mToken;
```

```
46    public static final String ACTION_AID_SELECTED =
47          "com.android.nfc_extras.action.AID_SELECTED";
```

```
55    public static final String EXTRA_AID =
56          "com.android.nfc_extras.extra.AID";
```

```
70    public static final String ACTION_APDU_RECEIVED =
71          "com.android.nfc_extras.action.APDU_RECEIVED";
```

```
80    public static final String EXTRA_APDU_BYTES =
81          "com.android.nfc_extras.extra.APDU_BYTES";
```

```
88    public static final String ACTION_EMV_CARD_REMOVAL =
89          "com.android.nfc_extras.action.EMV_CARD_REMOVAL";
```

```
102   public static final String ACTION_MIFARE_ACCESS_DETECTED =
103         "com.android.nfc_extras.action.MIFARE_ACCESS_DETECTED";
```

```
113   public static final String EXTRA_MIFARE_BLOCK =
114         "com.android.nfc_extras.extra.MIFARE_BLOCK";
115
116   NfcExecutionEnvironment(NfcAdapterExtras extras) {
117     mExtras = extras;
118     mToken = new Binder();
119   }
```

```
135   public void open() throws IOException {
136     try {
137       Log.d(TAG,
138             "open() for " +
139             ((mExtras.mPackageName != null) ?
140                   mExtras.mPackageName : "[null]"));
141       Bundle b = mExtras.getService().open(mExtras.mPackageName,
142                                            mToken);
143       throwBundle(b);
144     } catch (Exception e) {
145       mExtras.attemptDeadServiceRecovery(e);
146       throw new IOException("NFC Service was dead, try again");
147     }
148   }
```

```
158   public void close() throws IOException {
159     try {
160       Log.d(TAG,
161             "close() for " +
162             ((mExtras.mPackageName != null) ?
163                   mExtras.mPackageName : "[null]"));
164       throwBundle(mExtras.getService().close(
165             mExtras.mPackageName,
166             mToken));
167     } catch (Exception e) {
```

```
        mExtras.attemptDeadServiceRecovery(e);
        throw new IOException("NFC Service was dead");
      }
    }

    /**
     * Convert a byte array into its hexadecimal string form.
     * @param b  Byte array.
     * @return   Hexadecimal string representation.
     */
    private static String convertByteArrayToHexString(byte[] b) {
      if (b != null) {
        StringBuilder s = new StringBuilder(2 * b.length);

        for (int i = 0; i < b.length; ++i) {
          final String t = Integer.toHexString(b[i]);
          final int l = t.length();
          if (l > 2) {
            s.append(t.substring(l - 2));
          } else {
            if (l == 1) {
              s.append("0");
            }
            s.append(t);
          }
        }

        return s.toString();
      } else {
        return "";
      }
    }
```

```
    public byte[] transceive(byte[] in) throws IOException {
      Bundle b;
      try {
        Log.d(TAG,
              "transceive() for " +
              ((mExtras.mPackageName != null) ?
                    mExtras.mPackageName : "[null]") +
              ": C-APDU=" + convertByteArrayToHexString(in));
        b = mExtras.getService().transceive(mExtras.mPackageName,
                                            in);
      } catch (Exception e) {
        mExtras.attemptDeadServiceRecovery(e);
        throw new IOException(
              "NFC Service was dead, need to re-open");
      }
      throwBundle(b);
      byte[] out = b.getByteArray("out");
      Log.d(TAG,
            "transceive() for " +
            ((mExtras.mPackageName != null) ?
                  mExtras.mPackageName : "[null]") +
            ": R-APDU=" + convertByteArrayToHexString(out));
      return out;
    }

    private static void throwBundle(Bundle b) throws IOException {
      if (b.getInt("e") == -1) {
        Log.d(TAG, "IOException: " + b.getString("m", "[null]"));
        throw new IOException(b.getString("m"));
      }
    }
}
```

Index

M. Roland, *Security Issues in Mobile NFC Devices*, T-Labs Series in Telecommunication Services, DOI 10.1007/978-3-319-15488-6