Technical Article

An Introduction to Near-Field Communication and the Contactless Communication API

By C. Enrique Ortiz, June 2008

Near-field Communication (NFC) is characterized as a very short-range radio communication technology with a lot of potential, especially when applied to mobile handsets. Imagine yourself using your cellphone to interact with posters, magazines, and even with products while at the store, and with such interaction initiating a request or search for related information in real-time. Other usages of NFC include the electronic wallet to make payments using your handset, the same way you do with your credit card. With NFC all this is possible. But NFC is still a young technology. That said, NFC-enabled handsets are being introduced into the market, and deployments and pilots around the world are occurring. This article explores NFC and how you can leverage it in your Java application by using the *Contactless Communications API*.

Contents

What is NFC?

NFC Modes

The Contactless Communication API

Anatomy of a Contactless Communication API MIDlet

Using the Contactless Communication API

The Contactless Communication and the Generic Connection Framework API

Security Considerations

Resources

Samsung Exhibit 1030
Page 1 of 18

About the Author

What Is NFC?

Near-field Communication or NFC is a standard defined by the NFC Forum, a global consortium of hardware, software/application, credit card companies, banking, network-providers, and others who are interested in the advancement and standardization of this promising technology.

NFC is a short-range radio technology that operates on the 13.56 MHz frequency, with data transfers of up to 424 kilobits per second. NFC communication is triggered when two NFC-compatible devices are brought within close proximity, around four centimeters. Because the transmission range is so short, NFC-based transactions are inherently secure; more on this shortly.

How do NFC compares to the other short-range communication technologies? The following table provides a quick comparison:

	NFC	RFID	IrDa	Bluetooth
Set –up time	<0.1ms	<0.1ms	~O.5s	~6 sec
Range	Up to 10cm	Up to 3m	Up to 5m	Up to 30m
Usability	Human centric Easy, intuitive, fast	Item centric Easy	Data centric Easy	Data centric Medium
Selectivity	High, given, security	Partly given	Line of sight	Who are you?
Use cases	Pay, get access, share, initiate service, easy set up	Item tracking	Control & exchange data	Network for data exchange, headset
Consumer experience	Touch, wave, simply connect	Get information	Easy	Configuration needed

Figure 1: Comparing NFC to other close range communication technologies (Source: NFC Forum)

When compared to the other short-range radio technologies, NFC is extremely short ranged and what I call *people-centric*. Some of the other short-range communication technologies have similar characteristics, for example RFID, while others are completely different yet complimentary to NFC; for example Bluetooth and Infrared. A good scenario of such compliment is the combination of NFC and Bluetooth, where NFC is used for pairing (authenticating) a Bluetooth session used for the transfer of data.

NFC Standards

NFC is a standard, and is ISO standards-based. The **ISO 14443** Type A and Type B standards + **FeliCa** is a four-part international standard for contact-less smart cards operating at 13.56 MHz in close proximity with a reader antenna. The **ISO 18092** standard defines communication modes for NFC Interface and Protocol.

NFC Modes

The NFC forum defines three communication modes, as illustrated next:

Samsung Exhibit 1030
Page 2 of 18

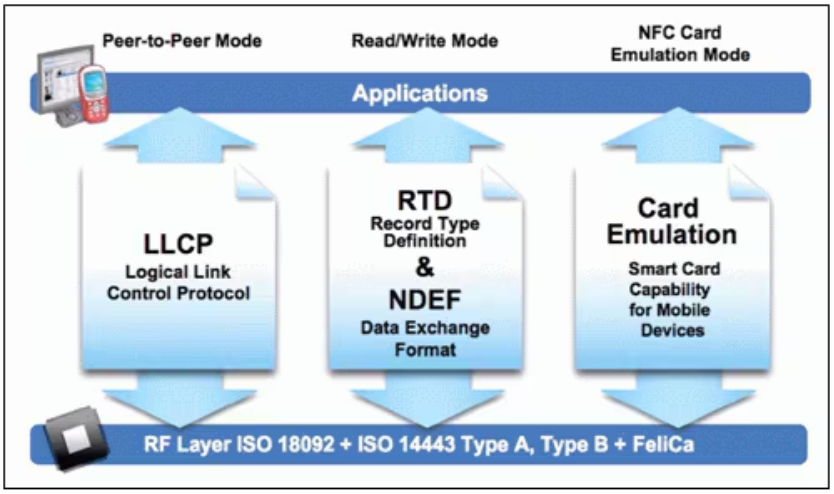


Figure 2: NFC Communication Modes (Source: NFC Forum)

where:

- **Peer-to-Peer** mode is defined for device to device link-level communication. Note that this mode is not supported by the Contactless Communication API.
- **Read/Write** mode allows applications for the transmission of NFC Forum-defined messages. Note that this mode is not secure. This mode is supported the Contactless Communication API.
- **NFC Card Emulation** mode allows the NFC-handset behave as a standard Smartcard. This mode is secure. This mode is supported by the Contactless Communication API.

NFC Terminology

- **NDEF** - NFC Data Exchange Format - standard exchange formats for URI, Smart Posters, other
- **RTD** - Record Type Definition - An NFC-specific record type and type name which may be carried in an NDEF record
- **NDEF message** - Basic message construct defined by this specification. An NDEF message contains one or more NDEF records
- **NDEF record** - Contains a payload described by a type, a length, and an optional identifier
- **NDEF payload** - The application data carried within an NDEF record

Next let's take a look at the Contactless Communication API, the anatomy of a NFC-based application, and how to use the API for NFC communication.

The Contactless Communication API

The Contactless Communication API Java specification, led by Nokia and defined under the Java Community Process as JSR-257, defines a set of APIs for proximity, contactless-based communication. The API consists of five Java packages, as follows:

Table 1 - JSR 257 Java Packages

Samsung Exhibit 1030
Page 3 of 18

Java Package	Interfaces	Classes	

`javax.microedition.contactless`
A **mandatory** package that contains all the target discovery and classes common to all targets
	`TagConnection`
	`TargetListener`
	`TargetProperties`
	`TransactionListener`
		`DiscoveryManager`
		`TargetType`
			`Co`

`javax.microedition.contactless.ndef`
An optional package for communicating with NDEF formatted data tags
	`NDEFRecordListener`
	`NDEFTagConnection`
		`NDEFMessage`
		`NDEFRecord`
		`NDEFRecordType`	

`javax.microedition.contactless.rf` An optional package for communicating with RFID (no NDEF formatted data) tags	`PlainTagConnection`		
`javax.microedition.contactless.sc` An optional package for communicating with external smartcards	`ISO14443Connection`		
`javax.microedition.contactless.visual` An optional package for reading and generating visual tags	`ImageProperties` `VisualTagConnection`	`SymbologyManager`	`Visu`

* Note that the only mandatory package is `javax.microedition.contactless`

The Contactless Communication API allows you to **Discover** and **Exchange** data with contactless targets such as NDEF tags, RFID tags, and external smartcards. The API also provides support for visual tags. The following diagram illustrates the relationships between the different API classes and interfaces:

Samsung Exhibit 1030
Page 4 of 18

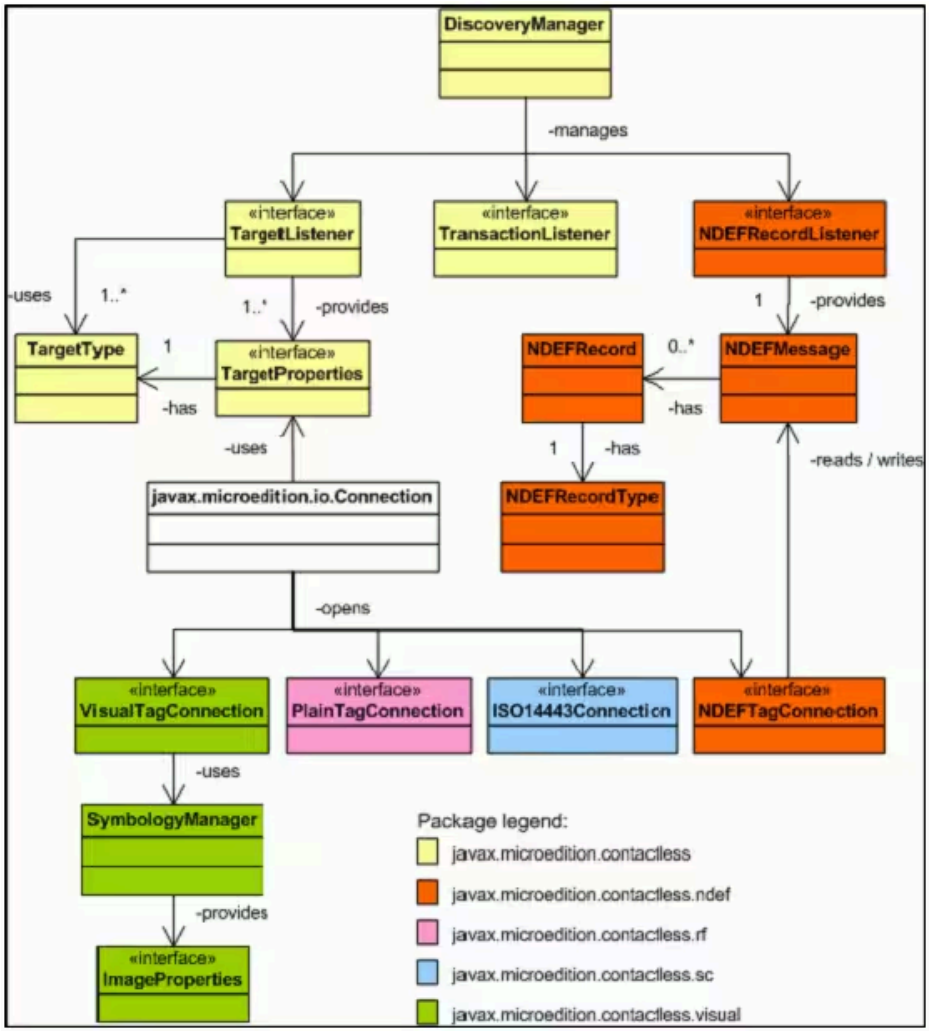


Figure 3: Contactless Communication API Relationships (Source: JSR-257 Specification)

Next let's cover the anatomy of a typical contactless/NFC-based MIDlet.

Anatomy of a Contactless Communication API MIDlet

Let's now cover the elements of a typical mobile Java application (MIDlet) that uses the Contactless Communication API looks. This is illustrated next:

Samsung Exhibit 1030
Page 5 of 18

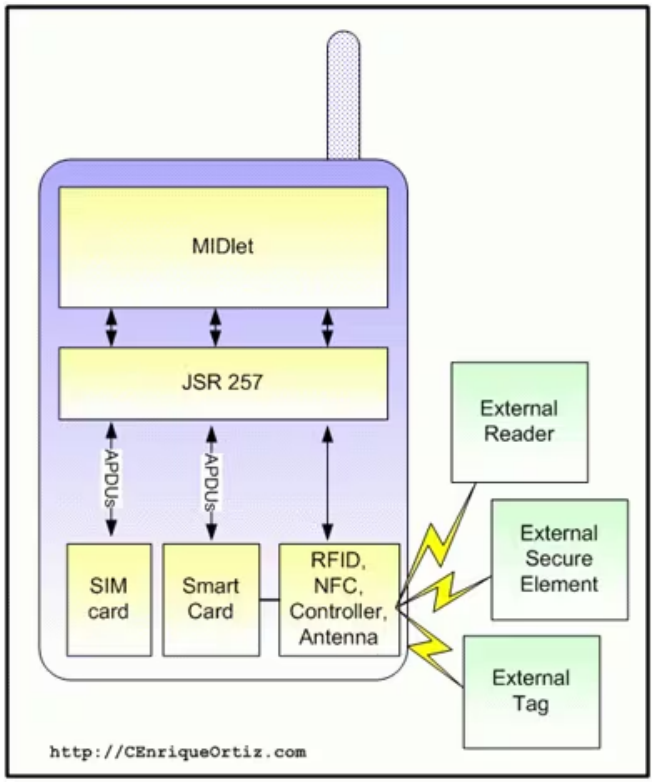


Figure 4: Anatomy of a Contactless Communication API-based Java MIDlet (Source: CEnriqueOrtiz.com)

where we have the following typical elements:

- The Java Runtime with JSR-257 implementation,
- the MIDlet application running on a handset,
- RFID/NFC transponder, controllers, and baseband,
- a SIM card, as well as secure and external elements.

External Readers	Secure Elements
External readers include contactless payment readers in Point of Sale stations, ticketing systems on transportation systems, external radio, visual tags such as NFC, RFID and barcodes, or Smartcards	Secure elements (SE) can be interna elements; example of a secure elem based smartcard. MIDlets can acces using the Security and Trust Servic the Contactless Communication AP readers access internal secure elem RFID circuitry (using the Card Emul

Next, let's take a look at how at how to use the Contactless Communication API.

Using the Contactless Communication API

Samsung Exhibit 1030
Page 6 of 18

The Contactless Communication API allows you to *discover* and *exchange data* with supported contactless radio and visual targets (tags). Applications using the Contactless Communication API typically follow the flow illustrated next:

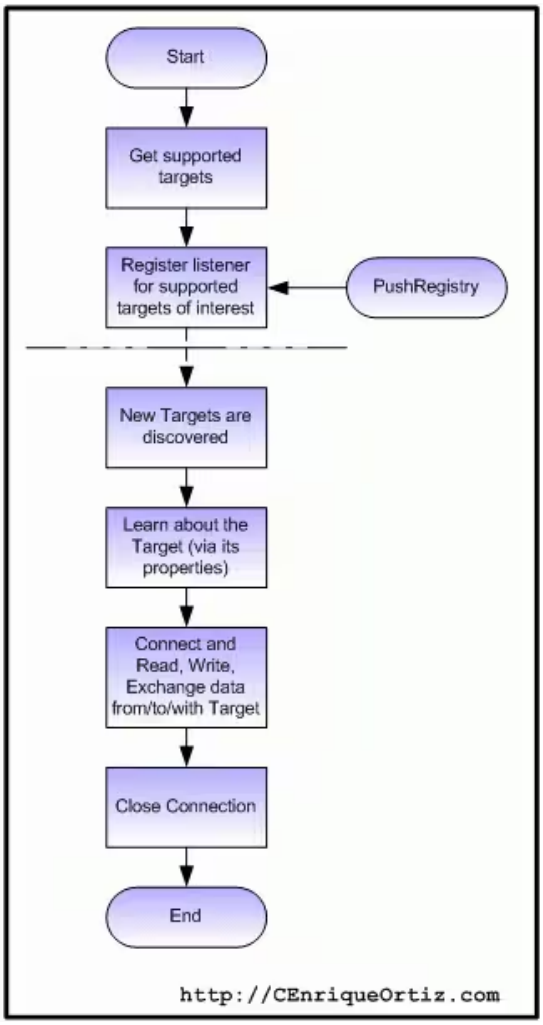


Figure 5: Typical Flow of a Contactless Communication Application

where:

1. The first step is for the application to query the implementation to discover the target types that are supported by the handset,

2. For each supported target type, the application can register a target listener to receive activity notifications. Alternatively, the application can register with the *PushRegistry* for activation due to target activity; supported target activities are NDEF and secure element in card emulation mode activities,

3. As targets come into proximity, they are detected (discovered) by the implementation, which in turn notifies the application(s) by invoking the appropriate activity listeners. Alternatively the PushRegistry activates the MIDlet,

4. For each discovered target, the application can learn the target's properties,

5. The application can connect to, read, write and exchange data with the discovered target,

Samsung Exhibit 1030
Page 7 of 18

6. When done, to release resources, the application closes any opened connections.

The following section covers the relationship between the Connectionless Communication API and the Generic Connection Framework.

The Contactless Communication API and the Generic Connection Framework

The Contactless Communication API extends the Generic Connection Framework (GCF) as shown next:

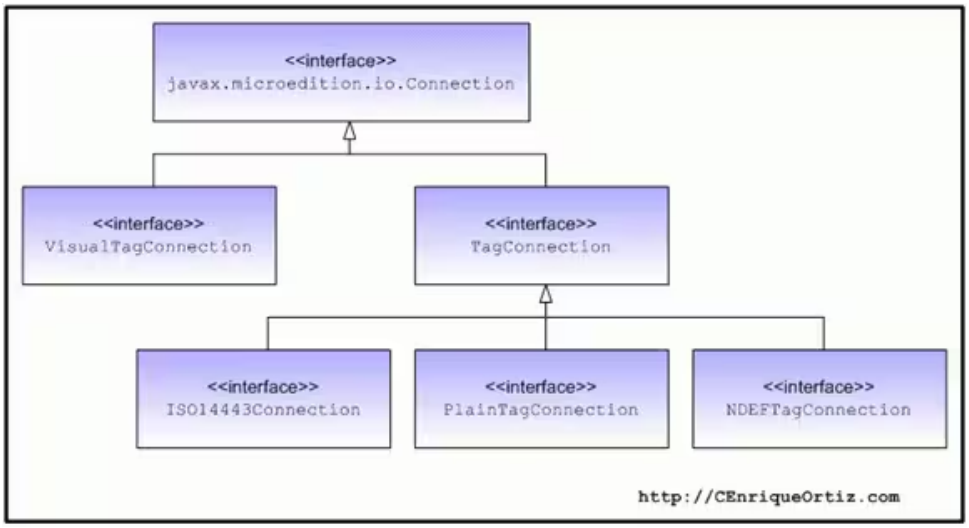


Figure 6: The Contactless Communication API and the Generic Connection Framework

The Contactless Communication API defines the following connection-related interfaces:

- `TagConnection` defines the base interface for all RFID, smartcards, and NFC connections:
  - `ISO14443Connection` extends `TagConnection` and is used for contactless connection with ISO 14443-4 compliant contactless smart card
  - `PlainTagConnection` extends `TagConnection` and is used for contactless connection with RFID (non-NFC) tags
  - `NDEFTagConnection` extends `TagConnection` and used for contactless connection with NFC Forum formatted RFID tags and contactless smart cards
- `VisualConnection` is used for contactless connection to a visual tag such as a bar code

To learn more about this connection framework see the article The Generic Connection Framework.

Discovering and Listening for Supported Targets To discover targets that are supported by the platform invoke the method `DiscoveryManager.getSupportedTargetTypes()`, which returns an array of `TargetTypes`. Then for each target type of interest it registers a target listener. The following code snippet shows the method `registerTargetListener()` that queries the platform for the supported targets, then for targets of type ISO 14443-4 (compliant contactless smart card), it registers a target listener:

Listing 1 - Discovering Targets and Registering Target Listeners Copy

Samsung Exhibit 1030
Page 8 of 18

```
import javax.microedition.contactless.TargetListener;


:

DiscoveryManager dm = DiscoveryManager.getInstance();

:

/**
 * Discover supported targets, registers listeners
 *
 * @param targetListener the target listener
 */
public void registerTargetListeners(TargetListener targetListener) {
    // Discover supported types
    TargetType[] tp = DiscoveryManager.getSupportedTargetTypes();
    try {
        // Register listener for each of the supported types
        for (int i=0; i<tp.length; i++) {
            if (tp[i].equals((TargetType.ISO14443_CARD))) {
                dm.addTargetListener(
                    targetListener, TargetType.ISO14443_CARD);
            } else...
               :
               :
            }
        }
    } catch (Exception e) {
        // ...
    }
}
```

Note that the Contactless Communication API defines the following target types: `ISO14443_CARD`, `NDEF_TAG`, `RFID_TAG` and `VISUAL_TAG`.

Once the targets of interest have been discovered and the appropriate target listeners registered, as illustrated above, the implementation will invoke the method `TargetListener.targetDetected(TargetProperties[])` as new targets are detected:

Listing 2 - Listening for Registered Target Listeners

Copy

```
import javax.microedition.contactless.TargetListener;


:
```


Samsung Exhibit 1030
Page 9 of 18

```
/**
 * A new target has been detected. This method is invoked by
 * the platform.
 *
 * @param prop the properties for the detected target
 */
public void targetDetected(TargetProperties[] prop) {
    for (int i = 0; i < prop.length; i++) {
        // Get UID
        String uid = prop[i].getUid();
        // Get Connection Classes
        Class[] classes = prop[i].getConnectionNames();
        // Get Target Types
        TargetType[] types = prop[i].getTargetTypes();
        // Connect to each Target
        String url = prop[i].getUrl();
        try {
            // Open NDEFTagConnection to the target
            NDEFTagConnection conn =
                    (NDEFTagConnection) Connector.open(url);
            :
            :
        } catch (IOException e) {
            // ...
        }
    }
}
```

The method `targetDetected()` typically performs the following steps:

- Method `targetDetected` receives the `TargetProperties` for detected targets,
- For each detected target, its URL is retrieved from its properties object,
- A connection to the target is made using the GCF,
- Data as appropriate is exchanged,
- Incoming messages are processed accordingly to its message attributes,
- When done, all previously acquired resources are released, and connections are closed.

Please note that reading RFID tags require understanding how ISO14443 I/O and ADPU commands work; this is outside the scope of this article.

Listening for NDEF-specific Targets

The Contactless Communication API also provides for the discovery of NDEF-specific targets without having to know the target details; all it needs to know are the record (message) types and how to process them, resulting in a much simpler communication and data-exchange logic. To listen to NDEF-targets, implement the interface `NDEFRecordListener` and its method

Samsung Exhibit 1030
Page 10 of 18

`recordDetected(NDEFMessage ndefMessage)`. Register an NDEF-record listener by invoking the Discovery manager method `addNDEFRecordListener(listener, recordType)`. The following code snippet illustrates this:

Listing 3 - Listening for NDEF Targets

Copy

```
import javax.microedition.contactless.ndef.NDEFRecordListener;

:

DiscoveryManager dm = DiscoveryManager.getInstance();

:

// Register NDEF_TAG target (smart poster) to discover
try {
    NDEFRecordType rt = new NDEFRecordType(
        NDEFRecordType.NFC_FORUM_RTD, "urn:nfc:wkt:Sp");
    dm.addNDEFRecordListener(this, rt);
} catch (IllegalStateException e) {
    :
} catch (Exception e) {
    :
}
```

The following are the defined NDEF record types:

- **EMPTY**- Record type name format identifier for the empty record.
- **EXTERNAL_RTD** - Record type name format identifier for application specific record type names that follow the NFC Forum naming conventions.
- **MIME** - Record type name format identifier for the MIME type defined in the RFC 2046.
- **NFC_FORUM_RTD** - Record type name format identifier for the NFC Forum Record Type Description.
- **UNKNOWN** - Record type name format identifier for the unknown record type.
- **URI** - Record type name format identifier for the URI type defined in the RFC 3986.

From the JSR 257 Specification: If the format is EMPTY or UNKNOWN the name must be null. The record type names of NFC_FORUM_RTD and EXTERNAL_RTD record types must follow the naming rules defined in the NFC Forum RTD specification. The record type name must only contain characters in the US_ASCII character set.

Processing `NDEFMessages`

Samsung Exhibit 1030
Page 11 of 18

Once the NDEF listener has been registered, the platform will invoke the method `recordDetected(NDEFMessage ndefMessage)` for the requested NDEF, passing as argument an NDEF message for the NDEF target that became active and visible. The method `recordDetected()` then gets record and record type, and other information from the NDEF message, and processes the message according to message attributes.

Listing 4 - Processing NDEF Target Messages

Copy

```
/**

 * Called by the platform, when the requested NDEF record type is
 * discovered by the device from the contactless target.
 *
 * @param ndefMessage the NDEF message to process
 */
public void recordDetected(NDEFMessage ndefMessage) {
    // Get records and record types from NDEF Message
    NDEFRecordType[] rTypes = ndefMessage.getRecordTypes();
    NDEFRecord[] records = ndefMessage.getRecords();
    for (int i=0; i<records.length; i++) {
        // Handle data, based on type of NDEFMessage
        NDEFRecordType t = recordTypes[i];
        NDEFRecord r = records[i];
        byte[] id = r.getId();
        long len = r.getPayloadLength();
        byte[] p = r.getPayload();
        // Process the record
        // ...
    }
}
```

The processing of the message payload will depend on your application; for example, after reading a URL, the application may post some information to that URL, or perhaps will open such URL in the web browser.

Using the Card Emulation Activity Notifications

In card emulation mode, a secure element on the device communicates and transacts with an external reader over RFID hardware. Applications are notified of such transactions, but don't participate on them. This is illustrated next:

Samsung Exhibit 1030
Page 12 of 18

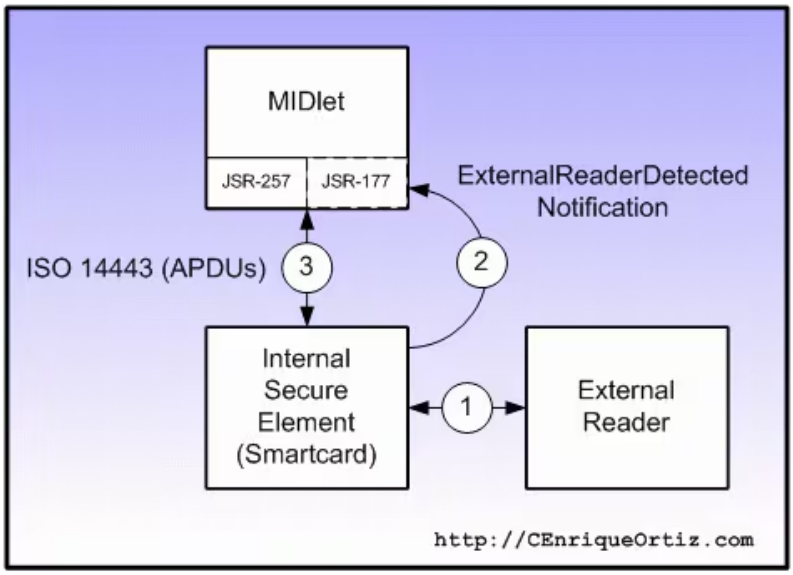


Figure 7: Card Emulation Activity Notifications

where:

1. The internal security element interacts with an external reader,

2. The application (MIDlet) is notified when the external reader has been detected,

3. If needed, the application communicates with the secure element, using the Contactless Communication API ISO14443 connection interface, or SATSA if available.

The communication between secure element and the external readers is transparent to the application itself, and occurs using APDU commands similarly to how Java Cards communicate with external readers; for more information see An Introduction to Java Card Technology - Part 1. Note that the communication between the application and the secure element requires internal knowledge of the secure applet within the secure element.

The APDU commands are defined in ISO7816-4 specification.

The next diagram, taken from the article An Introduction to Java Card Technology - Part 1, illustrates the typical relationships of a Java Card application (in this case from the perspective of MIDlet playing the role of the "reader"), and the Secure Elements (playing the role of the "card-side"):

Samsung Exhibit 1030
Page 13 of 18

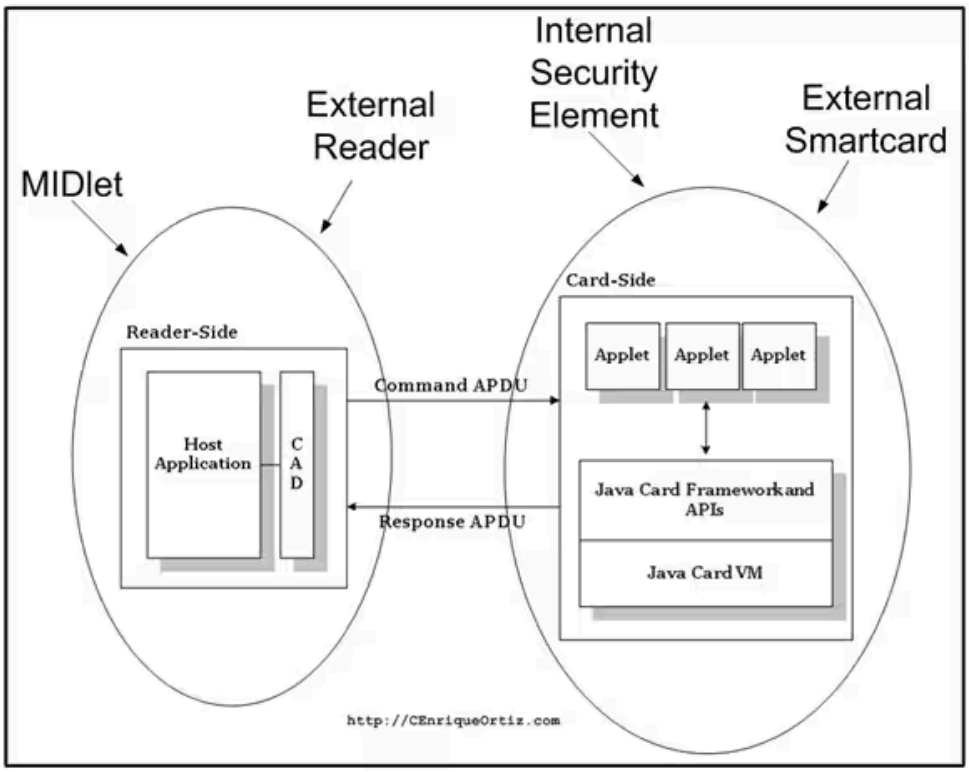


Figure 8: Typical Elements of a Java Card Application; Contactless Communication Application Scenario (Source: CEnriqueOrtiz.com)

where:

- On a Contactless (NFC) handset, the left-side could be an internal reader, the MIDlet itself, or and external reader (via NFC Card Emulation Mode),
- The right-side, the "card", could be an internal or an external secure element, that is accessible via SATSA or JSR-257, or over RFID hardware,
- All communication occur using APDU commands.

Registering for and Processing Card Emulation Activity

As previously mentioned, when in Card Emulation mode, applications can get notified when an external reader is detected, but the application itself doesn't participate in the transaction itself. Also, if needed, the application is responsible for communicating with the secure element using APDU commands, and for this the application must have internal knowledge of the applet running on the secure element. To register for card emulation activity notifications implement the interface `javax.microedition.contactless.TransactionListener` and its method `externalReaderDetected(byte slot)`. Register a transaction listener by calling the Discovery manager method `dm.addTransactionListener(...)`. The following code snippet illustrates this:

Listing 5 - Registering and Processing Card Emulation Activity

Copy

Samsung Exhibit 1030
Page 14 of 18

```
import javax.microedition.contactless.TransactionListener;


:
:

// Register Transaction Listener
try {
    dm.addTransactionListener(this);
} catch (IllegalStateException e) {
    ...
} catch (Exception e) {
    ...
}

:
:

/**
 * Called by the platform, when a card emulation event
 * has happened on the RFID hardware.
 *
 * @param slot is the slot needed to open the APDUConnection defined
 *             in JSR 177 to the external secure element, may be
 *             UNKNOWN_SLOT constant defined in this interface, if the
 *             slot can not be identified.
 */
public void externalReaderDetected(byte slot) {

    // Based on slot number above, using ISO14443Connection or SATSA
    // connect to applet, query applet, update screen, etc.
    ...
}
```

For more information about Java Card and SATSA, see Resources section and the end of this article.

Using PushRegistry to Launch NFC Applications

Automatic application activation or startup is a very important aspect for the success and acceptance of NFC-based applications, as it allows for a better user experience; by just coming into proximity of (or touching) a reader or another NFC device, the application is automatically started. The PushRegistry facility that is part of the Mobile Information Device Profile (MIDP) provides for the automatic application-launch based on timers and connection activity. Note that the Contactless Communication API extends the PushRegistry for application startup only for 1) NDEF record types and 2) for secure element (activity) card emulation mode communication.

The connection URLs used for the registration must follow proper naming convention, as specified by specification:

Samsung Exhibit 1030
Page 15 of 18

- URL format for NDEF push connections: "`ndef:`"<record_type_format> `?name=`<record_type_string> Where:
  - <record_type_format> is "`rtd`", "`external_rtd`", "`mime`", or "`uri`",
  - <record_type_string> is a UTF-8 String is the fully qualified name of the record type, as specified by the implementation, for example, urn:nfc:wkt:Sp (Nokia Smart-posters)
- URL format for card emulation activity: "`secure-element:`"`?aid=`<aid_string> Where:
  - <aid_string> is a String containing the applet identifier (numerical characters) for the applet of interest, per ISO7816-5 specification

To ensure that the application when launched receives notifications related to discovered NDEF records or card emulation activity, it must register its `NDEFRecordListener` and `TransactionListener` as soon as it is started. To learn more about the PushRegistry see The MIDP 2.0 Push Registry.

Security Considerations

The Contactless Communication API is subject to the security policies of the underlying mobile platform. For MIDP, the following methods or operations require permissions:

Table 2 - Contactless Communication API MIDP Security Permissions

Method or Operation	**Permission**
`DiscoveryManager.getInstance()`	`javax.microedition.contactl`
Write `NDEFMessage message`	`javax.microedition.contactl`
Opening `NDEFTagConnection`	`javax.microedition.io.Conne`
Opening `PlainTagConnection`	`javax.microedition.io.Conne`
Opening `ISO14443Connection`	`javax.microedition.io.Conne`
Opening `VisualTagConnection`	`javax.microedition.io.Conne`

Samsung Exhibit 1030
Page 16 of 18

In this model, you request permissions via the JAD file or the JAR manifest, by creating `MIDlet-Permissions` property entries, using the permission names in the table above. Attempting to use a restricted operation without proper permission causes the system to throw a `SecurityException`.

For implementations of the Contactless Communication API on CDC profiles, the application must use `java.security.Permission`-based security checks. The following methods require permissions:

- `DiscoveryManager.getInstance()`
- `NDEFTagConnection.write(NDEFMessage message)`

Resources

- JSR 257: Contactless Communication API
- JSR 177: Security and Trust Services API for J2ME
- An Introduction to Java Card Technology - Part 1
- An Introduction to Java Card Technology - Part 2, The Java Card Applet
- An Introduction to Java Card Technology - Part 3, The Smart Card Host Application
- Near Field Communication (Wikipedia)

About the Author

C. Enrique Ortiz is a long time mobile technologist, developer, and writer. He maintains the About Mobility Weblog. He has been author or co-author of many publications, and has been an active participant in the Java mobility community. Enrique holds a B.S. in Computer Science from the University of Puerto Rico and has more than 18 years of software engineering, product development, and management experience.

Resources for
Careers
Developers
Investors
Partners
Researchers
Students and Educators

Why Oracle
Analyst Reports
Best cloud-based ERP
Cloud Economics
Social Impact
Culture and Inclusion
Security Practices

Learn
What is cloud computing?
What is CRM?
What is Docker?
What is Kubernetes?
What is Python?
What is SaaS?

News and Events
News
Oracle CloudWorld
Oracle CloudWorld Tour
Oracle Health Summit
Oracle Dev Tour
Search all events

Contact Us
US Sales: +1.800.633.0738
How can we help?
Subscribe to emails
Integrity Helpline
Accessibility

© 2025 Oracle Privacy / Do Not Sell My Info Cookie Preferences Ad Choices Careers